UAV-based Forest Fire Detection and Localization Using Visual and Thermal Cameras

Mohsen Sadi

A Thesis

in

The Department

of

Mechanical, Industrial and Aerospace Engineering

Presented in Partial Fulfillment of the Requirements

for the Degree of

Master of Applied Science (Mechanical Engineering) at

Concordia University

Montréal, Québec, Canada

January 2021

CONCORDIA UNIVERSITY

School of Graduate Studies

This is to certify that the thesis prepared

By: **Mohsen Sadi**

Entitled: **UAV-based Forest Fire Detection and Localization Using Visual and Thermal Cameras**

and submitted in partial fulfillment of the requirements for the degree of

Master of Applied Science (Mechanical Engineering)

complies with the regulations of this University and meets the accepted standards with respect to originality and quality.

Signed by the Final Examining Committee:

______________________________ Chair
Dr. Chun-Yi Su

______________________________ External Examiner
Dr. Jun Yan

______________________________ Examiner
Dr. Chun-Yi Su

______________________________ Co-supervisor
Dr. Youmin Zhang

______________________________ Co-supervisor
Dr. Wen-Fang Xie

Approved by ______________________________
Dr. Mamoun Medraj
Chair of Department or Graduate Program Director

__________ 2021 ______________________________
Dr. Mourad Debbabi, Dean
Gina Cody School of Engineering and Computer Science

Abstract

UAV-based Forest Fire Detection and Localization Using Visual and Thermal Cameras

Mohsen Sadi

In this research, a UAV (unmanned aerial vehicle) system for detecting and locating forest fires is developed. This system uses the information from two cameras: one visual camera and one thermal camera for fire detection and navigation. Two images from these cameras are aligned before they can be used. An alignment process is created and a homography matrix is computed so that it is assured that the two images are aligned and every pixel is the same in both images. By combining data extracted from these cameras, it can verify whether there is no fire or a real fire is happened or a fake fire. A two-degree-of-freedom (2DOF) frame is implemented for testing the tracking and locating capabilities of the UAV. Both cameras are mounted in this frame. This system is based on the concept of image based thermo-visual servoing (IBTVS). That is, it extracts thermal and visual information of a scene and then it computes the position of a desired object (fire) and commands the servos to move to the desired position. This frame can simulate 2DOF movement of a UAV and can be used to test the developed fire detection algorithms. Finally, a co-simulation system is presented to verify the application of fire detection in a simulated UAV. The experimental tests demonstrate that the developed algorithms can guide the UAV to fly on a predefined path and look for any possible fire. As soon as a fire is detected, the system will alarm and calculate the location of fire and fly the UAV over the fire for further investigations.

Acknowledgments

First and foremost, I thank God for everything that I have. As our scholars have told us, one who does not express gratitude to the creatures, does not express it to the Creator, I would like to thank everyone who has helped me accomplish this achievement. I will give my sincere gratitude to both of my advisors. Dr. Youmin Zhang and Dr. Wen-Fang Xie for their support throughout the days of research. Dr. Zhang, thank you for all of your kindness and positivity. Dr. Xie, thank you for the times that you put for me. For all of the efforts that you have for my research. Thank you both of you for your thorough supports without which this research would have never been done. I would not be able to do this project without the motivation and support of my lovely wife. Thank you dear Massi for standing beside me and for bearing the hardship of a student life far from your family. You are the best. Then, special thank goes to my mother whose prays solves a lot of my problems. Thank you Meysam, my dear brother, for your financial support throughout my studies. Thank you everyone in my family. Last but not the least, I would like to thank all of my friends who have helped me in this project. Anim, thank you for your precious help in this research. Mikail thank you for patiently answering my questions.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

The importance of forests is inevitable. Their existence is necessary for our survival. Forests purify the air we breathe, make the wood we need, purify the water we drink, moderate the climate, and stabilize the soil etc. They cover almost one-third of our planet's surface and are home to two-third of its species. They can be a source of wealth for a country. Personally, I feel calm and relaxed anytime I go to the woods.

Unfortunately, every year, millions of hectares of forests are destroyed by fires. These fires can have natural causes or can be human made. The main natural causes of forest fires are lightning, dry climate and volcanic activities. Human-caused fires are mainly because of small fires, discarded cigarettes, power-line sparks etc. Each year, millions of dollars are spent to extinguish forest fires and many lives are taken as a result of these fires [1–4]. These fires are constantly threatening the infrastructures, ecological systems and public safety [5]. They have the potential of doubling the temperature of lower atmosphere of Earth [6]. Unwanted fires can spread rapidly to places that threaten the human life as well as the ecological system of the forests. Hence, it is crucial to prevent these fires in the early stage [7].

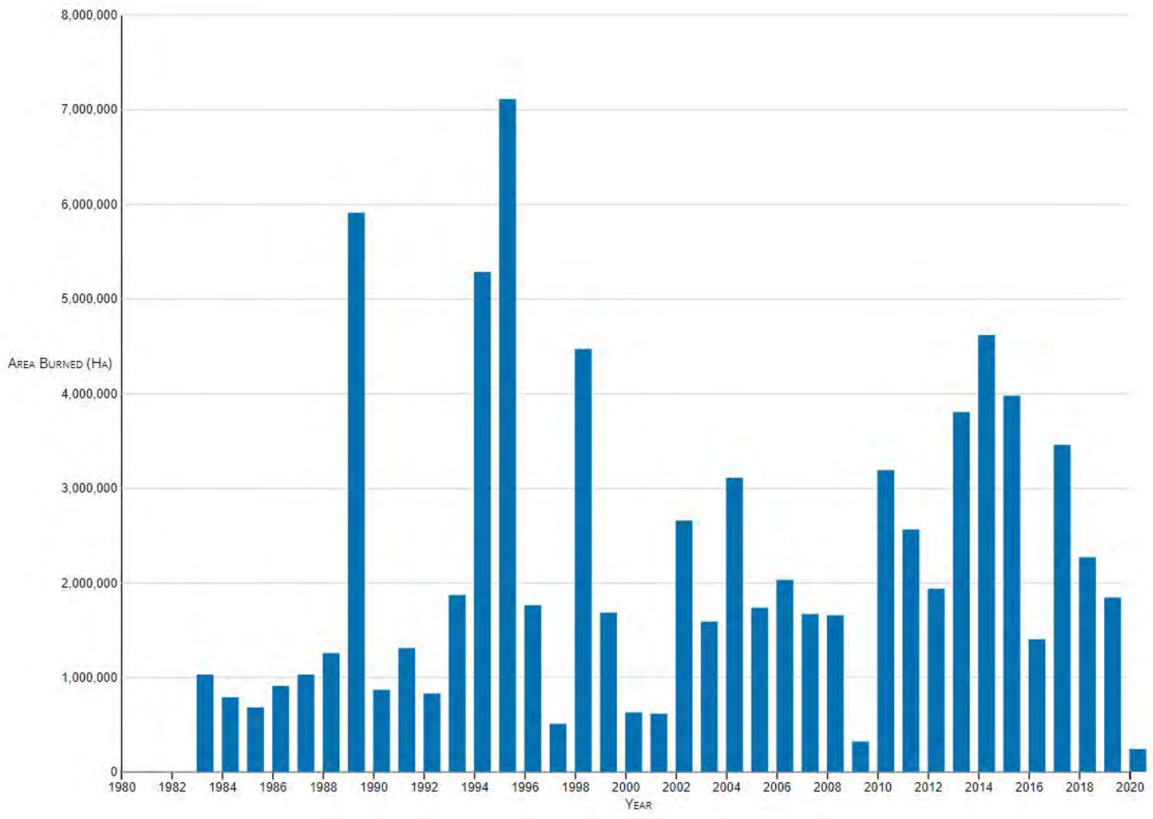


Figure 1.1: Annual area burned in the forests of Canada [8].

Traditionally, the approaches to detect forest fires are: human patrol, smoke and thermal detectors, ground-based equipment, manned aircraft and satellite imagery [9], [10]. Each of these methods has its own drawback. For example, smoke and thermal sensors require proximity to the fire and cannot provide information on the size or location of the fire [11]. Ground-based equipment may have limited surveillance ranges. Human patrol is not practical in large and remote forests. Satellite images are not vivid enough to detect early stage fires and they lack the ability of continuously monitoring forests because they have less flexibility in their path planning [9], [12]. Manned aircraft operations are expensive and require skilled pilots. Moreover, this can potentially threaten the crews' lives because of hazardous environments and operators fatigue [13]. Over the last 20 years, At least 80 pilots have died during firefighting operations only in the USA [14].

Unmanned Aerial Vehicles (UAV) are nowadays widely used in many different military and civilian operations. They can carry different payloads with different usages. With a camera as payload, one UAV can perform numerous different surveillance operations. For example, it can do geographic mapping of inaccessible locations and terrain or it can be used for pipeline or power-line monitoring. Nowadays they are even used in movie industries to create unique view of the scene. Furthermore, if we use a thermal camera instead of a visual camera, even more data can be revealed from the scene. With the new developments in UAV technology, cheaper commercial UAVs are available for numerous research projects. UAVs can access high-risk zones, provide over-the-hill view, perform night time mission with no risk of human lives [15]. They can provide these benefits:

(1) wide area coverage

(2) work at any time of the day and long duration

(3) cheap operation cost and easily recoverable

(4) minimum disturbance to wildlife

(5) various sensors can be mounted and therefore different missions can be done

(6) minimum need of operator's involvement [16]

In recent years, a huge amount of researches have been carried out in the field of forest fire monitoring and detection by UAVs. In forest fire detection operations, UAVs can play many roles. The initial usage of UAVs was to send them to forests and record a video and by watching the video later, an operator can define if a fire has happened. In this method, finding the exact location of a fire can be challenging. Moreover, further investigations require additional operations. Later, UAVs are used to send real-time visual data to their bases so that an operator can check whether a fire exists or not in real-time. A reliable communication method is necessary in this method. If the received video quality is not acceptable, the method can lead to false results. Moreover, with intelligent devices onboard, UAVs can detect fires in early stages and signal to its operator the possibility of a fire. Also, they can be used to diagnose the detected fire in term of location and size of the fire. This can be done with the help of UAV's inertial measurement unit (IMU) and global positioning system (GPS) sensors. They can predict the fire behavior and aid firefighters in making decisions for rapid suppression and contamination. Various sensors and mathematical methods are used to perform these missions. Gas sensor, charged coupled device (CCD) camera, infrared (IR) and thermal cameras and smoke sensor are a few of the instruments used. Median filtering, color space conversion, the Otsu threshold segmentation, neural networks, and many other mathematical techniques are used to detect and localize forest fires. Fig. 1.2 shows two examples of UAVs used in the process of forest fire detection.

1.2 Thesis objective

The aim of this study is to develop an onboard system that uses both visual and thermal cameras for finding forest fire. This system can be installed on any UAV and it will do all of the computations on its powerful onboard mini computer. A predefined path is loaded to the autopilot of UAV and it will follow that path. Once a valid fire is detected, it will signal the base the existence of the fire and move towards it until it reaches over the fire (in a safe altitude). Meanwhile it can locate the exact position of the fire and the area exposed to it. The whole process should be done while the fire is in its early stages. As a result, it opens the possibility to detect forest fires in the early stages using a low-cost UAV with high precision and low error.

Thesis goals:

(1) Design a fire detection algorithm that uses thermal and visual cameras' data to detect, track and locate the true forest fire;

(2) Develop a fire localization algorithm to guide the UAV to fly over the detected fire;

(3) Software-in-the-loop simulation testing of the system.

1.3 Thesis organization

This thesis is organized in six chapters.

- The first chapter mainly talks about the importance of the problem and motivation and objectives behind doing this research.
- The second chapter reviews the previous works done by others in this field. This includes different sensors and techniques used by other researchers in this field.
- The third chapter introduces the system model and the hardware and software used in this research including the model of 2DOF frame.
- The fourth chapter presents the developed fire detection and localization algorithms and the techniques of fusing the data from visual and thermal cameras. It also discusses the IBTVS system on the 2DOF frame.
- In the fifth chapter, experimental tests on two different test platforms are carried out to validate the developed algorithms. The first one is the 2DOF frame in which the cameras are mounted. This is used to test the fire detection and tracking capabilities of the system. Then the second test platform is a software-in-the-loop system which is used to test if the system can look for a possible fire and if a fire is found it will locate the fire location and move over it.
- Finally, in chapter six, the concluding remarks are presented and recommended future works are described. In the end, the appendix including all of the codes written and used in this research is presented.

Figure 1.2: Usage of UAVs in forest fire monitoring

Chapter 2

Literature Review

In order to develop a state-of-the-art system, it is necessary to study and understand the methodologies of the past and current systems that are conducted in this field. The literature review aims at determining the drawbacks and advantages of the existing systems and decide in which way they can be improved.

2.1 General methodology and architecture of UAV-based forest fire detection systems

UAVs are classified as fixed-wing or rotary-wing [9]. The benefits of using fixed-wing UAVs over rotary-wing ones is that they are cheaper, less complex and have longer endurance and can carry larger and heavier payloads. Rotary-wing UAVs, on the other hand, can hover in a stationary spot in the air and do not need a runway to take-off and land. Choosing the type of UAV depends on the defined mission. There are experiments that have used rotary-wing or fixed-wing UAVs or a combination of both. Generally, if the area of surveillance and the fire itself are small enough for a rotary-wing UAV to fly over and check, a rotary-wing UAV is a better option. Otherwise, in a large forest or for a vast fire, a fixed-wing UAV will be a better choice.

A UAV that performs the mission of forest fire detection consists of multiple parts which are mainly in these categories [16]:

(1) The frame of UAV and its payload (which includes all of the sensors and devices used for fire detection and also communication to the base);

(2) Image processing methods for fire detection and possibly tracking and localization;

(3) Autopilot system for control and navigation;

(4) Ground station for ground processes.

The payload of a UAV can be from wide range of sensors and electrical devices. They may have a temperature, gas or smoke sensor, a GPS, an IMU, visual, infrared and/or thermal camera and communication devices for transmitting data to their base. In forest fire detection operations, the main focus is on using different cameras (rather than temperature or gas or smoke sensors). Although these sensors are made specially for fire detection in small environments, they are not suitable for fire detection in an open environment like a forest.

Fig. 2.1 is a typical UAV and its components used in forest fire detection. The mission of forest fire monitoring can be defined in three steps: fire detection, fire diagnosis and fire prognosis [17]. In the fire detection step, a single UAV or a fleet of them [18], [19] look for a fire in the surveillance region. Once the existence of a fire is confirmed, it will trigger an alarm to notify firefighting team. In the case of multiple UAVs, other UAVs are sent to the location to verify the existence of fire. Once a fire is verified, the stage of diagnosis starts. Now, the UAV will determine the fire's location and its boundaries. Finally, in prognosis it predicts the fire's future evolution based on wind and other conditions. Not all of the UAV-based systems have all these three stages. They can have one or more of these.

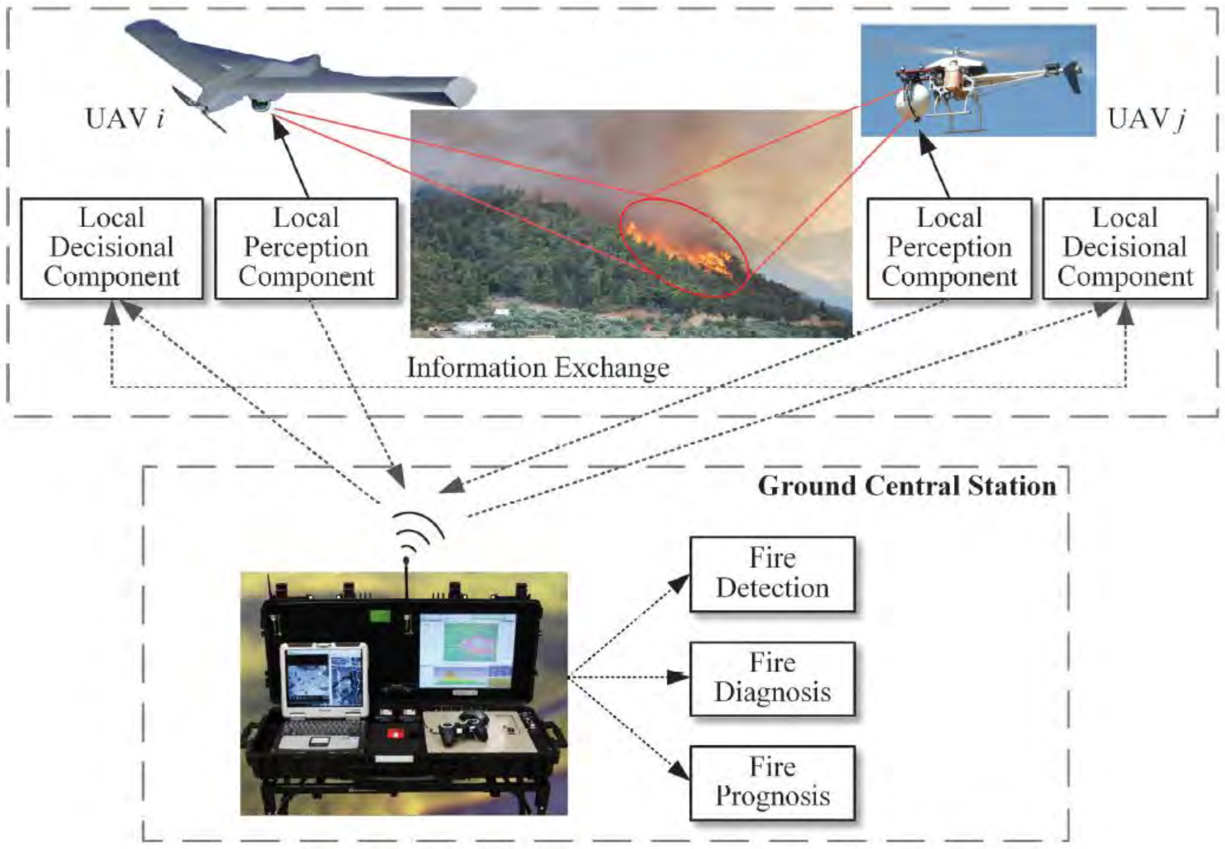

UAV i
UAV j
Local Decisional Component
Local Perception Component
Local Perception Component
Local Decisional Component
Information Exchange
Ground Central Station
Fire Detection
Fire Diagnosis
Fire Prognosis

Figure 2.1: UAV-based forest fire detection components [17]

.

2.2 Benefits of using UAVs in forest fire detection missions

In the case of forest fire, a huge layer of smoke can cover the affected area. This may cause firefighters problems in terms of impaired vision and hazardous conditions [20]. This situation can be harmful for manned aircraft and firefighters to approach the fire. In this case, a UAV can be launched quickly to the fire region and it can monitor the fire from the desired position. Another benefit is it can look from above to give the firefighters better understanding of the extent of fire. Furthermore, firefighters can send it to hazardous positions without any danger to their own lives [21]. Moreover, the fact that using UAVs is the cheaper, safer and more flexible way of forest fire detection operation [22], [23]. UAVs can also perform long-time missions that are over a person's capabilities. However, there are some drawbacks for using UAVs. Unclear images due to vibrations and turbulences, lack of the power of decision making and unstable communication in deep valley regions. The benefits of using a UAV in forest fire monitoring operations lead to more research in this field over the last two decades. In recent years, a great progress has been done in this field to a point that nowadays they are used for even firefighting.

2.3 Use of visual, infrared and thermal cameras in forest fire monitoring systems

In the recent years, researchers have put massive efforts on image processing and computer vision methods for forest fire monitoring systems. The reason is that these methods have numerous merits namely, ability to monitor wide range object, intuitive and real-time imagery and also ability to record videos. A camera can be categorized into visual/CCD, IR or thermal based on the spectrum of light that it captures. Respectively, fire monitoring systems are classified into visual, infrared or thermal categories [11]. A system can either detect flame or smoke of a fire. To detect a fire (flame or smoke), one needs to extract the features of it which are mainly color, motion, and geometry [24]. Color and motion features of fire are used more than the geometry feature. Table 2.1 provides a summary of current UAV-based forest fire detection systems [17].

2.3.1 Visual-based systems

A camera-based fire detection system can be either offline or online. Chen et al. [34] use both chromatic and dynamic features of an offline video to detect fire. They make use of Red, Green and Blue (RGB) channels as features of each frame to detect flame. They also extract the features of smoke from the sequence of frames. Töreyin et al. [35] use color, motion information and fire flicker analysis to detect flame. In order to detect flicker process, they used hidden Markov model.

Table 2.1: UAV-based forest fire detection systems. [17]

Test Types	**UAV Class**	**Onboard Cameras (Resolution**	**Engine Power**	**Payload Capacity**	**References**
Near Operational	1 fixed-wing	1 thermal (720 $\times$640)	Fuel	340kg	[25]
Operational	1 fixed-wing	4 mid-IR (720 $\times$640)	Fuel	>1088kg	[26]
Near Operational	2 rotary-wing; 1 airship	1 visual(320 $\times$240); 1 IR (160 $\times$120)	Fuel; Electric	3.5kg	[27]
operational	1 fixed-wing; 1 rotary-wing	1 visual; 1 IR	Fuel	-	[28]
Near Operational	1 fixed-wing	1 visual; 1 IR	Fuel	<34kg	[29]
Near Operational	2 fixed-wing	1 visual; 1 IR; 1 visual (1920 $\times$1080)	Fuel	25kg; 250kg	[7]
Near Operational	2 fixed-wing	1 thermal (160 $\times$120); 1 NIR (752 $\times$582); 1 VNIR (128 $\times$128)	Electric	<2.6kg	[30]
Near Operational	1 fixed-wing	1 visual (720 $\times$480)	Gas	0.68kg	[21],[31]
Near Operational	1 rotary-wing	2 visual (4000 $\times$2656; 2048 $\times$1536); 1 thermal (320 $\times$240)	Fuel	907kg	[20]
Near Operational	1 fixed-wing	1 visual	Electric	-	[32]
Near Operational	1 fixed-wing	1 visual (656 $\times$492)	Electric	5.5kg	[33]

Note: (-) not mentioned; IR: Infrared; NIR: Near IR; VNIR: Visible-NIR.

Their method can be used online or offline. In [36], Töreyin et al. added temporal and spatial wavelet analysis and they were able to reduce the false alarms considerably. Then they used the same method to detect smoke in the early stage of fire in order to use it as an early signal of fire [37]. In [24], foreground information is extracted from background with an adaptive subtraction method in real-time. They also use a statistical color model for checking fire existence. They continue their work and in [38] they develop a generic color model for flame pixel classification. They use YCbCr color space instead of RGB. YCbCr color space make it possible to separate the chrominance from the luminance. Therefore, the algorithm can easily distinguish between fire pixels and fire-like ones. They achieve a high rate of fire detection. Phillips et al. [39] create a lookup table for colors. They then trained the table with massive images. They use both color and temporal variations to distinguish fire from non-fire objects. Yuan et al. [16] introduce a fire detection method that uses Lab color model. It extracts fire-pixels in the channel A of the color model by using Otsu method. They improve the performance of their work in [40] and develop an algorithm that uses both color and motion features. Color-based decision rules and optical flow method are the main feature of their work.

Many studies that use visual camera look for flame features in the image. However, some others look for smoke or a combination of both. Smoke feature can be used to the early and precise detection of fire. Chen et al. in [41] combine two decision rules to extract smoke pixels. Those rules are color-based static rule and diffusion-based dynamic rule. Their results show a robust solution for smoke classification. In [42], they use a texture analysis to detect smoke in real-time. This is done by using a back-propagation neural network. Experimental results validate that the algorithm is able to differentiate smoke and non-smoke images with high rate of true alarms. Yuan et al. [43] use the color feature for the candidate area. They also utilize optical flow method to calculate the motion vector.

In recent years, with the flourishing of intelligent methods, many works are done to improve the fire detection systems. The algorithms used in [24], [44], [45], [46] are fuzzy logic, artificial neural networks (ANNs) and fuzzy neural networks. Although these methods show effectiveness in experimental validations, most of them have not been used in real practical forest fire scenarios.

2.3.2 Infrared-based systems

Numerous works are done to develop more effective image processing techniques for fire detection. In visual images, the motion and color features are widely used to detect fire. However, utilizing CCD cameras in vision-based fire detection methods is not considered as a robust and reliable method for every outdoor application. In sophisticated, non-structured environments of forests there are many situations in which false fire alarms can be high. For example, smoke blocking fire flames or in situations that there are objects so analogous to fire features such as reddish leaves that sway in the wind or reflections of lights. IR cameras can capture images in weak or no light

Table 2.2: Fire detection methodologies using visual and infrared cameras [17]

Detection method	Spectral bands	OV	IV	OLV	CF	MF	GF	FD	SD	PP	GL	References
Training method	Visual Mid-IR	✖	✔	✖	✔	✖	✖	✔	✖	✔	✔	[47]
Training method	Visual Mid-IR	✔	✖	✖	✔	✖	✖	✔	✖	✔	✔	[1]
Image matching	Visual IR	✔	✖	✖	✔	✖	✖	✔	✖	✔	✔	[48],[49]
Data fusion	Visual IR	✖	✔	✖	✔	-	-	✔	✔	✖	✖	[50]
Neural networks	IR	✖	✔	✖	✔	✔	✖	✔	✖	✖	✖	[51]
Dynamic data-driven	Multi-spectral IR	✖	✖	✔	✔	✖	✔	✔	✖	✔	✖	[52]

Note: (OV) Outdoor validation; (IV) Indoor validation; (OLV) Offline validation; (CF) Color feature; (MF) Motion feature; (GF) Geometry feature; (FD) Flame detection; (SD) Smoke detection; (GL) Geolocation; (PP) Propagation prediction; (✔) considered; (✖) not considered; (-) not mentioned;

situations. Moreover, in some of IR cameras, smoke is transparent and it makes it possible to capture fire flames hiding behind smokes. Also, they can be utilized in both day and night time. Therefore, by using an IR camera it is possible to reduce false fire alarm rate and make use of forest fire detection systems in various environments. Table 2.2 summarizes researches done using IR cameras in their fire detection algorithms.

Infrared spectrum is divided into three regions; the near-, mid- and far- infrared, named for their relation to the visible spectrum. So-called IR cameras use a short wavelength infrared light to illuminate the environment and capture the reflected light to generate an image. Thermal cameras, on the other hand, are in the mid- or far-infrared regions. They only realize temperature difference and can create a temperature map and convert it into an image. One drawback of the IR cameras is that they can capture light from any light source such as headlight. But thermal cameras are not sensitive to those and they only capture temperature difference.

Infrared cameras do not rely on the environmental visual light and some of them (thermal) can work in smoky areas. Therefore, using them would be beneficial in daytime and especially night-time conditions. Refs. [53] and [27] utilize the threshold selection method introduced in [54] to generate binary images from IR images. They report that the false alarms are reduced remarkably. That is because of the significance of fire pixels in IR images. In [55], they fuse IR camera and visual camera images to detect the fire by generating some decision fusion rules. Pastor et al. [56] compute the propagation of forest fire in IR images by using linear transformations. Moreover, a threshold-value-searching criterion is applied to locate flame front position. In [57], they utilize some image processing tools and also a concept called dynamic data-driven application system (DDDAS). A multi-spectral IR image processing algorithm is used. This algorithm can evaluate active fire line, forest fire perimeter and fire propagation tendency. Huseynov et al. [58] propose a multiple artificial neural networks for detecting fire flame in IR images. The result shows that the training time can be reduced and fire detection is more successful. Yuan et al. [59] improve forest fire detection by using both motion and brightness features of fire. They extracted fire pixels from background and non-fire hot objects by taking advantage of optical flow analysis and histogram-based segmentation. Their experiments in IR video sequences show improved performance in the fire detection.

One issue associated with small IR cameras and their images is that they still have low sensitivity [27]. In order to overcome this issue, the exposure period should be increased to have higher-quality images. This requires a stationary base because vibration can have destructive effect on images with high exposure period. As a result, high frequency of vibrations in many UAVs can cause blurring in the images and this still remains a challenge in using IR cameras in forest fire detection.

2.3.3 Fusion of visual and IR images

One way of improving fire detection algorithms in terms of accuracy, robustness and reliability, is by fusing visual and IR images together. These improvements can be done by the use of fuzzy logic, probabilistic, statistical and intelligent methods (As illustrated in Table 2.2).

Arrue et al. [49] develop an algorithm in which IR image processing, fuzzy logic and artificial neural networks are used to improve fire detection operations. They use matching the information of visual and infrared images to confirm forest fires. In [47], authors integrate both visual and infrared cameras for fire front parameter calculation by taking advantage of image processing techniques in both visual and IR images. They did not conduct any experiment out of laboratory. They continue in [1] by introducing a forest fire perception system. By using computer vision techniques, visual and IR images are fused to extract a 3D fire perception. Then the fire propagation can be visualized by remote computer systems.

2.4 Summary and contribution of this thesis work

Although many works have been done in the field of forest fire detection, not many projects pay attention to the fusion of thermal and visual camera information. This work aims to solve the problem of image alignment which is necessary before fusion of cameras' information. Furthermore, an image-based thermo-visual servoing system is developed to track the location of fire and control the UAV respectively. This can mimic the movement of a UAV over a moving fire.

Chapter 3

System Components and Model

The purpose of this research is to develop an on-board real-time forest fire monitoring system that can detect any possible fire and locate it. For doing so, we have used a combination of a visual camera along with a thermal camera. Both cameras are connected to a mini processor (Raspberry Pi 4) and all of the computations are done within the on-board Raspberry Pi. Therefore, there is no need of using a high speed wireless data transfer and a powerful base computer. The system is capable of combining two images (RGB and thermal) and accurately defining whether a fire has been detected or not. A two-degree-of-freedom frame is made in order to simulate the fire detection and tracking capabilities of the system. Due to the limitations imposed by the appearance of COVID-19 pandemic, testing on a UAV was replaced by a software-in-the-loop simulation. This is beneficial because it can reduce the costs of real tests and possible failures. Many tests and scenarios are done and the systems is able to find and locate the fire.

First, the hardware used in this system is introduced in Section 3.1 and then the software part is discussed in Section 3.2. Finally, the model of the two-degree-of-freedom is discussed in Section 3.3.

3.1 System hardware

Selecting a proper set of hardware is an important part of a research. At first, for the core processor, an Arduino Due was chosen. Its performance was not good enough for image processing of both cameras. Therefore, after testing different devices a set of hardware is selected based on the budget and functionality.

3.1.1 Raspberry Pi

Raspberry Pi is a low-cost computer with size of a credit card. It can plug into a monitor or a TV and it can use mouse and keyboard. There are over ten different Raspberry Pi models. Appendix A compares four common Raspberry Pi models. Raspberry Pi 4 has a quad core Cortex-A72 (ARM v8) 64-bit processor with 4GB of DDR4 RAM. It also has wireless and Bluetooth communications. Furthermore, various sensors can be connected to it since it has 40 general-purpose input/output (GPIO) pins. It can be connected to most of the off-the-shelf autopilot boards and command them based on the data it collects from its sensors. In our research, all of the sensors and actuators are connected to the Raspberry Pi. Real-time fire searching is done in the Raspberry Pi by processing the images of two cameras (visual and thermal). This requires innumerable calculations. With the high-power processor and high-speed 4GB RAM of Raspberry Pi 4, it is capable of doing this task in real-time. Previously, a Raspberry Pi zero was used and it was not able to conduct this task in real-time. Because it has low power usage and it is light weight, it can easily be integrated in an unmanned aerial vehicle in forest fire monitoring operations. Finally, with 64 GB memory card on-board, it can save numerous images as well as videos of the fire scene for the further considerations. Fig. 3.1 shows an image of Raspberry Pi 4.

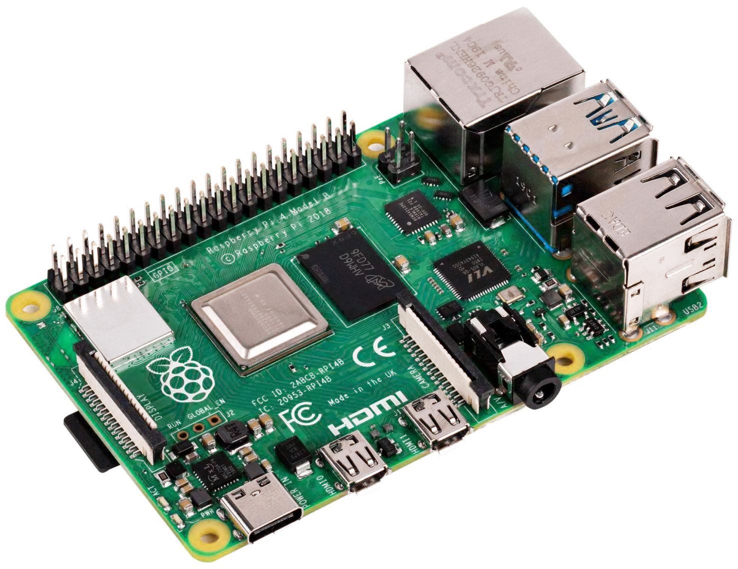


Figure 3.1: Raspberry Pi 4 mini processor

3.1.2 Visual camera

The next primary component is visual camera. It is used for detecting fire in the visual spectrum. It can use different fire features to detect fire. For example, a red threshold can be applied to the image to remove unwanted pixels and areas or the dynamics of the flame is used for the goal. Also, with the characteristics of smoke, a fire can be detected as well. The visual camera should be light weight, yet it should have high resolution and compatible with the Raspberry Pi. The 5 megapixels camera (OV5647 sensor) has 1080p video resolution at 30 FPS. Its field of view (FOV) is $54^\circ \times 41^\circ$. Since its field of view is different than that of the thermal sensor, it is needed to align the images from two cameras. The alignment process will be discussed in the next chapter. Raspberry Pi will capture images from the video of this camera anytime it needs one. Fig. 3.2 shows an image of the visual camera used in this research.

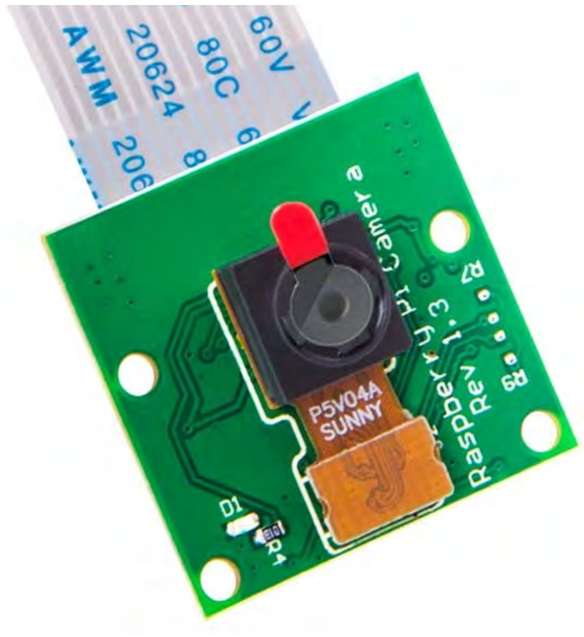


Figure 3.2: 5MP 1080p visual camera

3.1.3 Thermal camera

A thermal camera (sensor) is a device that creates an image from infrared radiation. It is similar to a visual camera that creates an image by visible light. A thermal image consists of many pixels each of which represents temperature of that point. A spectrum of colors can be assigned to the different temperatures. Therefore, a visual image is created from the thermal image. Fig. 3.3 shows a thermal image with the respected color spectrum and temperature. The camera is widely used in many applications such as

- High precision non-contact temperature measurements;
- Movement;
- Person localization;
- Temperature control of moving objects;
- IR thermometers.

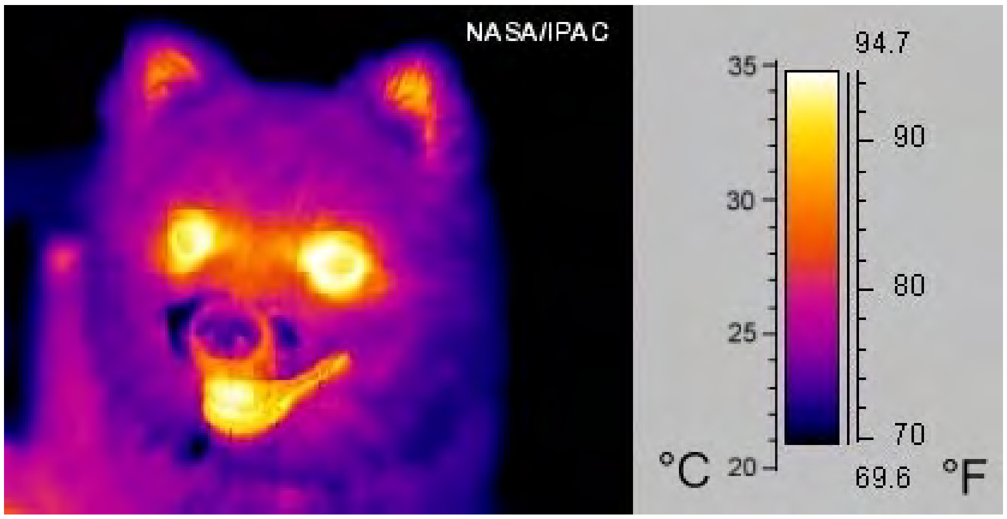


Figure 3.3: A thermal image and its color-temperature spectrum

Choosing the proper thermal camera is crucial because thermal camera is a key point in this system. In many cases like fires covered by smokes or fires under ashes, visual cameras cannot easily detect the existence of a fire and it is the duty of thermal camera to do so. In [60], Wardihani et al have used a 2 by 2-pixel temperature sensor with a FOV of five degrees. Usage of this sensor has two limitations:

1- it cannot calculate the area covered by the fire since it has limited pixels data;

2- it cannot search a wide area because of its limited field of view.

We have used a MLX90640 thermal sensor which has a resolution of 32 by 24 pixels and a FOV of $55° \times 35°$. This is beneficial as it has a FOV close to the visual camera's FOV. In this way, it is possible to align both cameras for further usages. This camera can capture images up to 64 FPS and can detect a wide range of temperatures with high precision (-40 to 300°C with approximately 1°C accuracy [61]). The extracted raw data from the thermal camera needs a computationally expensive process in order to become available as an image. Fig. 3.5 shows an image of a fire taken by the visual camera and Fig. 4.27 shows the same view from the thermal camera. This image is made after the heavy computations are done.

The communication between Raspberry Pi and the thermal sensor is with I^{2}C. I^{2}C is a multi-master, multi-slave serial communication bus. It is widely used in lower-speed ICs and microcontrollers in short-distance. In I^{2}C communication two wires are used to connect up to 128 devices. One wire is the generated clock signal by master (SCL) and the other wire is for data transfer (SDA). The frequency of communication can be up to 1MHz. The MLX90640 thermal sensor can have a refresh rate from 0.5Hz to 64Hz. The noises in the image is proportional to the refresh rate. Hence, choosing the proper refresh rate is necessary.

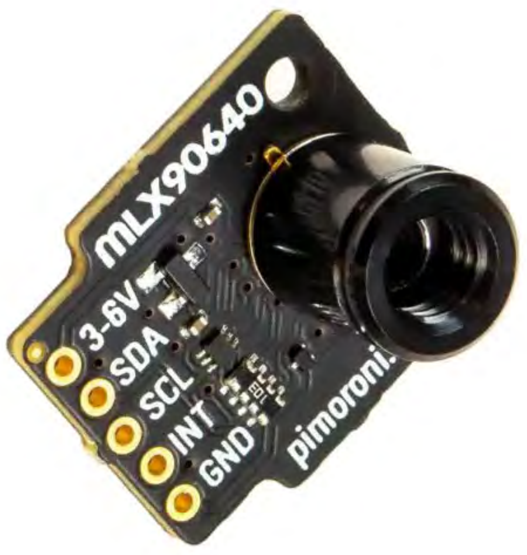


Figure 3.4: MLX90640, 32*24 pixels thermal camera

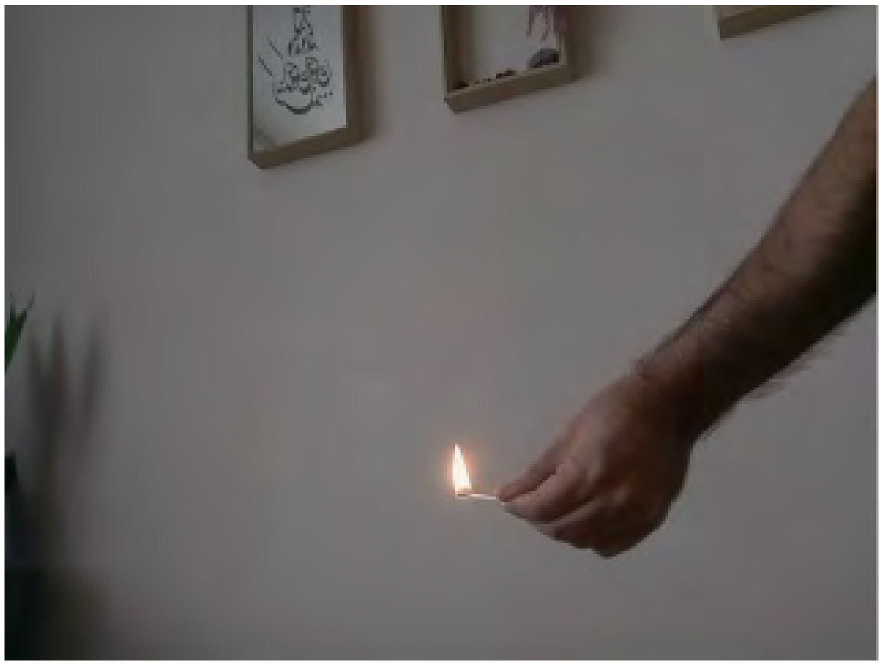


Figure 3.5: Visual image of a fire

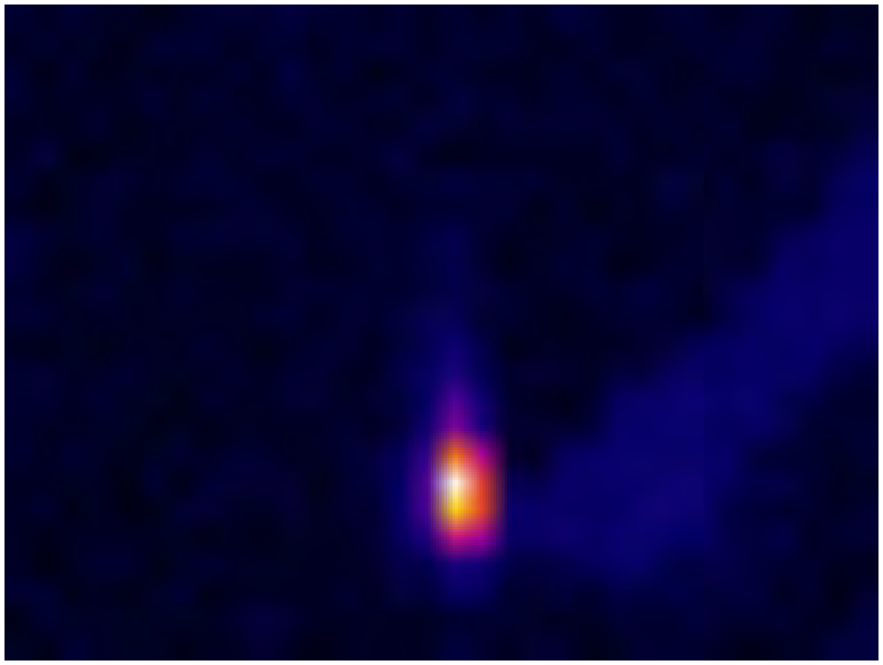


Figure 3.6: Thermal image of a fire

3.2 System software

3.2.1 Operating system

An operating system is software that manages all of the hardware resources associated with computer hardware. To put it simple, the operating system manages the communication between your software and your hardware. Without the operating system (OS), the software would not function. Like other computers, in order to run, Raspberry Pi needs to have an operating system. It runs in Linux. Just like Windows, iOS, and Mac OS, Linux is an operating system. In fact, one of the most popular platforms on the planet, Android, is powered by the Linux operating system. Linux is open-source, free, secure, lightweight, stable and suitable for programmers. Many different distributions are released for Linux. In this research, Rasbian is installed on the Raspberry Pi because it has the most compatibility with the Raspberry Pi's hardware.

3.2.2 Programming language

In order to program any microprocessor, a programming language should be selected. At first, for programming Arduino to connect and capture data from the thermal sensor, C++ language was chosen. Since the Arduino was not capable of heavy image processing of both cameras, a more powerful processor was needed. After considering multiple options, Raspberry Pi was chosen because of its high processing power and low cost. Later, when the Arduino was replaced by Raspberry Pi, a change of language was almost necessary. Despite the fact that C++ is a low-level language with better runtime performance, it is much harder to use available computer vision libraries in it. Python, on the other hand, has numerous libraries and functions that can be used easily. In this manner, more time is put over the methods of implementation rather than dealing with the hardships of the programming language. As a result, the final code is written totally in Python. Also, the whole driver of the thermal sensor is converted from C++ to Python.

3.3 Two-degree-of-freedom (2DOF) frame (Components and model)

A two-degree-of-freedom frame is used in this research to test the tracking ability of the system. This frame is made from stick wood. Four servo motors and a servo motor driver are used to rotate the whole system including Raspberry Pi and cameras in two directions. The servo driver is being commanded by an Arduino nano. The communication between the servo driver and Arduino is through I^2C. Two servos are used to rotate the board in one axis and two other servos are used for the other axis. Arduino receives the relative fire angles in two directions from the Raspberry Pi and rotates towards the fire. This frame is analogous to a UAV. It means, if a UAV (in a constant altitude) can move in two directions (pitch and yaw), this can rotate in two directions. In a UAV, if the system finds a fire, its location is expressed in terms of distance, but in this frame the location is expressed in terms of degrees.

In order to communicate properly with the driver, its related library should be used. The driver is called "Adafruit PCA9685 PWM Servo Driver" and the proper library can be found at [62]. This driver can run up to sixteen servo motors simultaneously. A simple Arduino code for running four servos can be written as below:

```
#include <Adafruit_PWMServoDriver.h>
Adafruit_PWMServoDriver pwm = Adafruit_PWMServoDriver();
void setup() {
    pwm.begin();
    pwm.setPWMFreq(50);}
void loop() {
    pwm.writeMicroseconds(0, 1000); // Servo 0 and 4 are for roll
    pwm.writeMicroseconds(4, 1000);
    pwm.writeMicroseconds(8, 1700); // Servo 8 and 12 are for yaw
    pwm.writeMicroseconds(12, 1700);}
```

Fig. 3.7 shows the diagram of Arduino and the driver, and the wiring and communication between them. The complete Arduino program for reading from Raspberry Pi and running the servos of the frame is available in Appendix B.3.

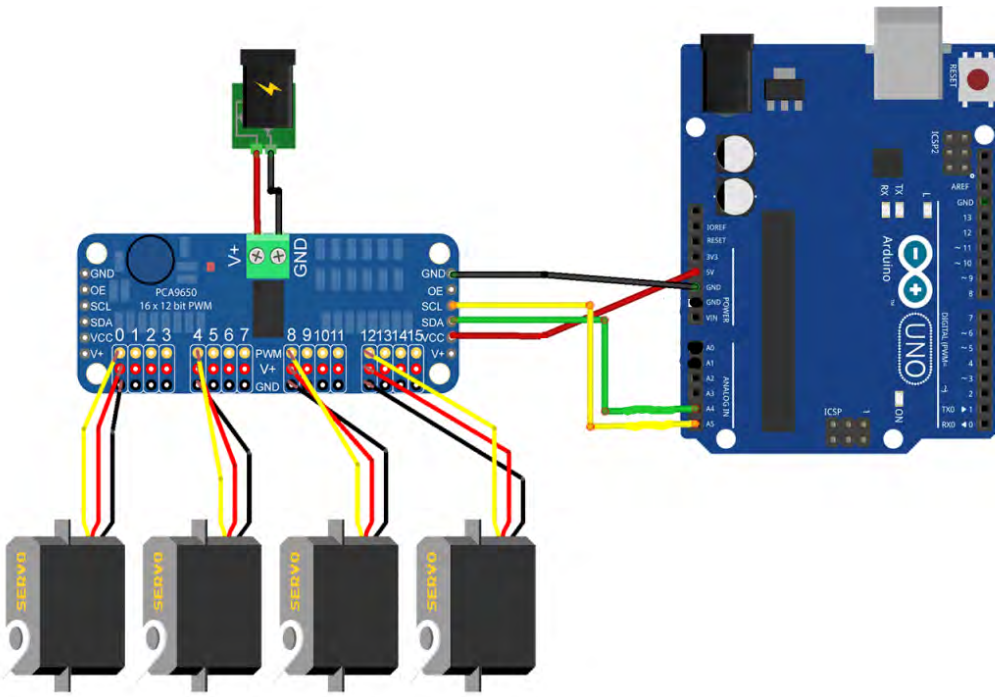
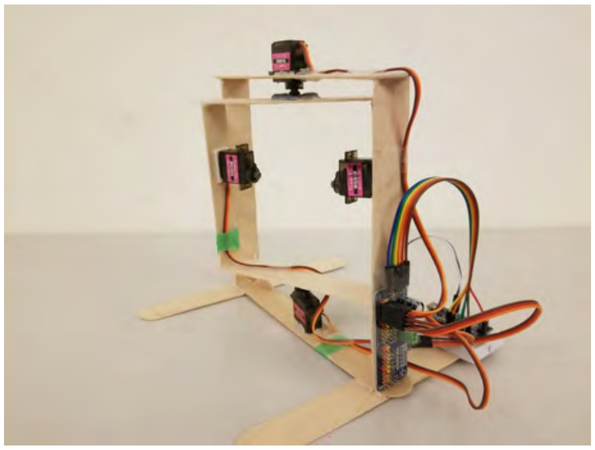


Figure 3.7: Communication of Arduino and Adafruit PCA9685 PWM servo driver

3.3.1 The model of 2DOF frame

Every mechanical and electrical system can be presented with its model. This frame is used in Chapter 4 for the purpose of testing the ability of tracking fire detection. Therefore, it is required to understand the model of this frame before we can design a control system for it. Fig. 3.9 illustrates the model of the 2DOF frame with all of its component. The developed control system is demonstrated in Chapter 4.


(a) The 2DOF frame without Raspberry Pi and cameras

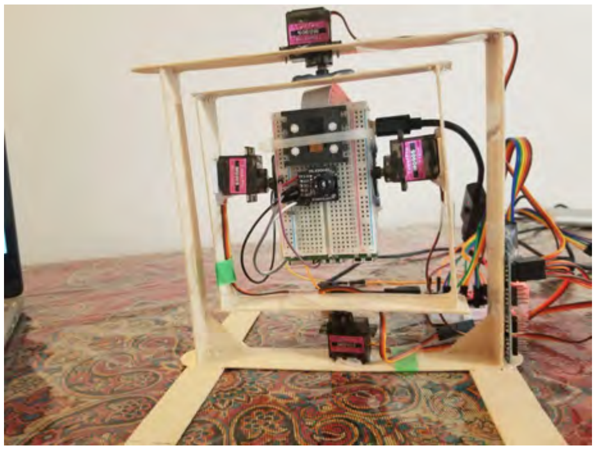
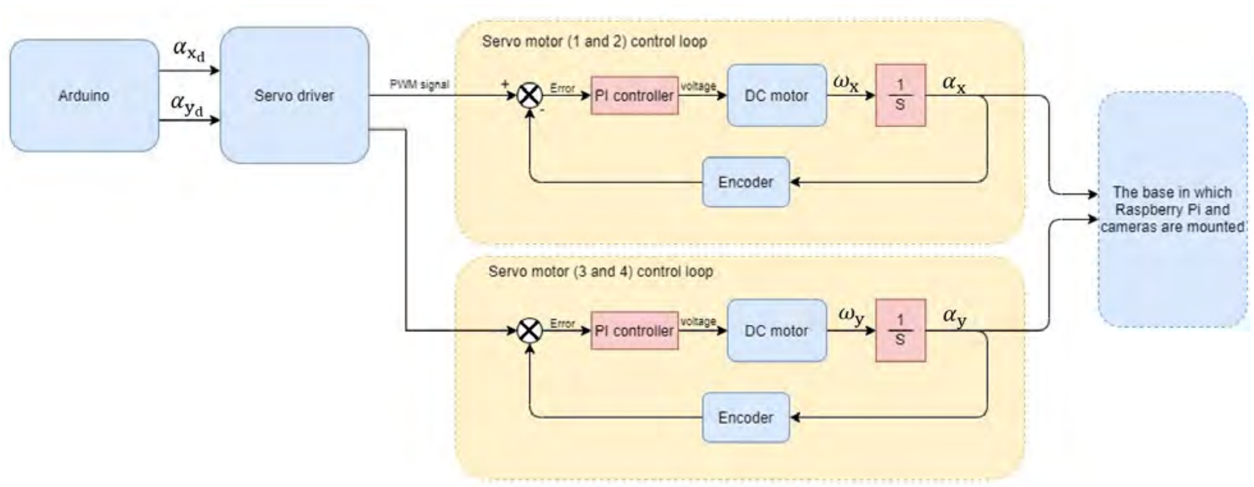

(b) The 2DOF frame with Raspberry Pi and cameras

Figure 3.8: The two-degree-of-freedom-frame



Note: α_{x_d}: desired angle around x-axis, α_{y_d}: desired angle around y-axis, α_x: angle around x-axis, α_y: angle around y-axis, ω_x: rate of rotation around x-axis, ω_y: rate of rotation around y-axis.

Figure 3.9: The model of 2DOF frame

3.3.2 Deriving Jacobian matrix for 2DOF frame

Any object in an image has a position and a velocity. These parameters can be related to the motion of camera. To find this relation, the Jacobian matrix has to be formed. By calculating the Jacobian matrix, the control rule can be built and the system can follow the desired posture. Fig. 3.10

illustrates the camera coordinate system and an point and its projection in the image.

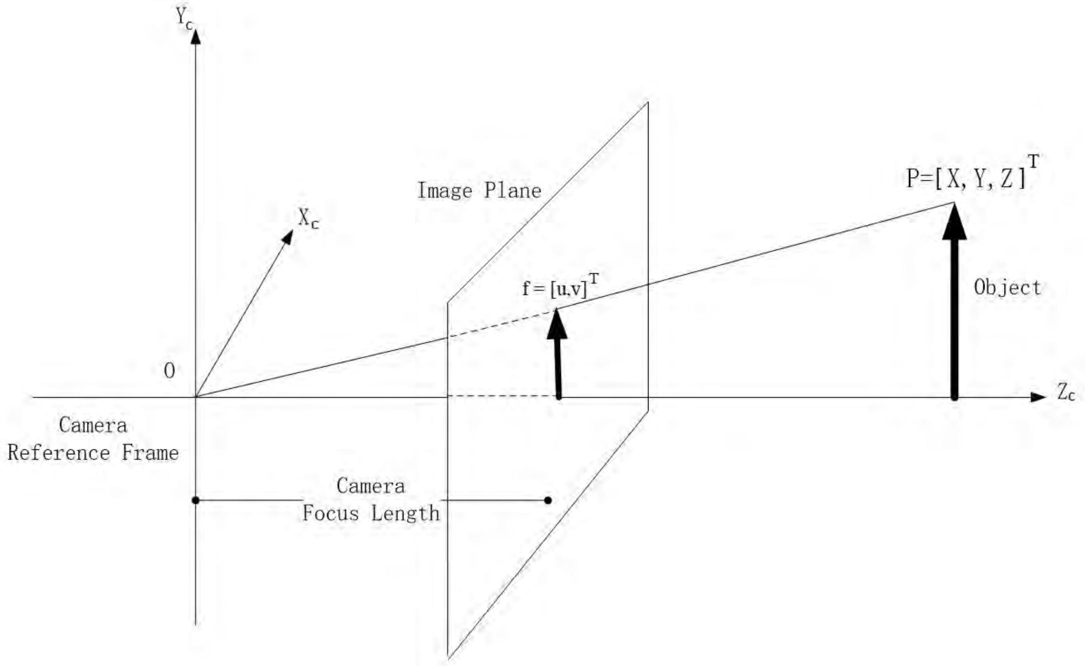


Figure 3.10: Camera coordinate system [63]

A point $P = [X, Y, Z]^T$ is defined in the camera coordinate system. Its projected point in the image frame is $f = [u, v]^T$ where u and v denote pixel indices.

The position of camera is denoted as following:

$$r = [x_c \; y_c \; z_c \; \alpha_x \; \alpha_y \; \alpha_z]^T \tag{3.1}$$

where in 2DOF frame, $x_c = y_c = z_c = \alpha_z = 0$, because the frame and the installed camera cannot move in any direction and can only rotate around x and y axes. α_x and α_x, are the inputs of the system. In other words, the system can rotate to any commanded (α_x, α_y) position.

By differentiating Eq. (3.1), the velocity of camera is:

$$\dot{r} = [v_x \; v_y \; v_z \; \omega_x \; \omega_y \; \omega_z]^T \tag{3.2}$$

where $v_x = v_y = v_z = \omega_z = 0$, because it can only rotate around x and y axes. The velocity of the projected point is defined as:

$$\dot{f} = \begin{bmatrix} \dot{u} \\ \dot{v} \end{bmatrix} \tag{3.3}$$

The velocity of point P is:

$$\begin{cases} \dot{X} = Z\omega_y - Y\omega_z + v_x = Z\omega_y & (\omega_z = v_x = 0) \\ \dot{Y} = X\omega_z - Z\omega_x + v_y = Z\omega_x & (\omega_z = v_y = 0) \\ \dot{Z} = Y\omega_x - X\omega_y + v_z = Y\omega_x - X\omega_y & (v_z = 0) \end{cases} \tag{3.4}$$

As can be seen in Fig. 3.10, the relationship between P and f is given by:

$$\begin{bmatrix} u \\ v \end{bmatrix} = \frac{\lambda}{Z} \begin{bmatrix} X \\ Y \end{bmatrix} \tag{3.5}$$

where λ is the focal length of camera (λ = 3.6mm). By taking derivative from both sides of Eq. (3.5), we have:

$$\begin{cases} \dot{u} = \lambda \frac{Z\dot{X} - X\dot{Z}}{Z^2} \\ \dot{v} = \lambda \frac{Z\dot{Y} - Y\dot{Z}}{Z^2} \end{cases} \tag{3.6}$$

Since u and v are image indices that start from top-left side of image, they need to change to image coordinates. In image coordinates, the origin is the center point. Therefore, we have:

$$\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} -1 & 0 & x_0 \\ 0 & -1 & y_0 \end{bmatrix} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} \tag{3.7}$$

where $(x, y)^T$ is the coordinates of the point f in the image coordinates. $(x_0, y_0)^T$ is the pixel-index of center point. By putting Eq. (3.5) into Eq. (3.4) the velocity components of P are given by:

$$\begin{cases} \dot{X} = Z\omega_y \\ \dot{Y} = Z\omega_y \\ \dot{Z} = \frac{Z}{\lambda}(v\omega_x - u\omega_y) \end{cases} \tag{3.8}$$

Combining Eqs. (3.6), (3.7) and (3.8) will result as following:

$$\begin{bmatrix} \dot{x} \\ \dot{y} \end{bmatrix} = \begin{bmatrix} \frac{xy}{\lambda} & -\frac{\lambda^2+x^2}{\lambda} \\ \frac{\lambda^2+y^2}{\lambda} & -\frac{xy}{\lambda} \end{bmatrix} \begin{bmatrix} \omega_x \\ \omega_y \end{bmatrix} \tag{3.9}$$

Equation (3.9) shows a relation between the velocity of a point in image coordinates and the rotations of camera. This relation is called the Jacobian matrix and is denoted by J_{image}:

$$\dot{f} = J_{image}\dot{r} \tag{3.10}$$

The inverse of Jacobian matrix is calculated as:

$$J_{image}^{-1} = \begin{bmatrix} \frac{\lambda xy}{D} & \frac{\lambda(\lambda^2+x^2)}{D} \\ \frac{\lambda(\lambda^2+x^2)}{D} & \frac{-\lambda xy}{D} \end{bmatrix} \tag{3.11}$$

where:

$$D = \lambda^4 + 2\lambda^2 x^2 + x^4 + x^2 y^2$$

By combining Eqs. (3.11) and (3.9) we find ω_x and ω_y in terms of speed of the point in image:

$$\begin{bmatrix} \omega_x \\ \omega_y \end{bmatrix} = J_{image}^{-1} \begin{bmatrix} \dot{x} \\ \dot{y} \end{bmatrix} \tag{3.12}$$

Equation (3.12) is used in Chapter 4 to form the visual servoing feedback controller.

3.4 Summary

This chapter discussed the hardware and software that are used in this research. Raspberry Pi, visual camera, and thermal camera are the main hardware of this work. The operating system installed on Raspberry Pi is Linux and the main programs are written in Python. Then, the model of the 2DOF frame is depicted and the Jacobian matrix is derived for the purpose of forming the control law in the next chapter.

Chapter 4

Fire Detection and Localization Algorithms

In this chapter, all of the methods on fire detection and localization are presented. We take a deep look at different parts of the program that will run the whole system of fire detection. But, before discussing about methods used in this research, it is beneficial to review some of the preliminary knowledge that are used in this chapter.

4.1 Preliminary knowledge

In vision-based fire detection systems, the main purpose is to extract the possible fire features from the image. This process is called image segmentation technique. This is done by differentiating fire pixels from background pixels. There are many techniques based on the feature that needs to be extracted. e.g. flame, smoke etc.

Flame is the main feature of fire. In an image of flame, its color is the easiest and most popular feature that can be extracted. Therefore, it is mainly used in developing fire detection techniques [11]. A color can be expressed in different ways with different properties. These different expressions are called color space. A detection technique may utilize one of these color spaces. Thus, it is worth looking at different color spaces and figure out the benefits of each of them. There are four predominant models used in digital image processing field:

(1) RGB: red, green and blue color space

(2) HSI: hue, saturation and intensity color space

(3) HSV: hue, saturation and value color space

(4) Lab color space

4.1.1 RGB color space

The RGB color space has three components: Red, Green, Blue. These primary colors can be mixed in various amounts to reform all of the possible colors. This color space is mainly used to display images in electronic displays such as monitors, televisions and cell phones.

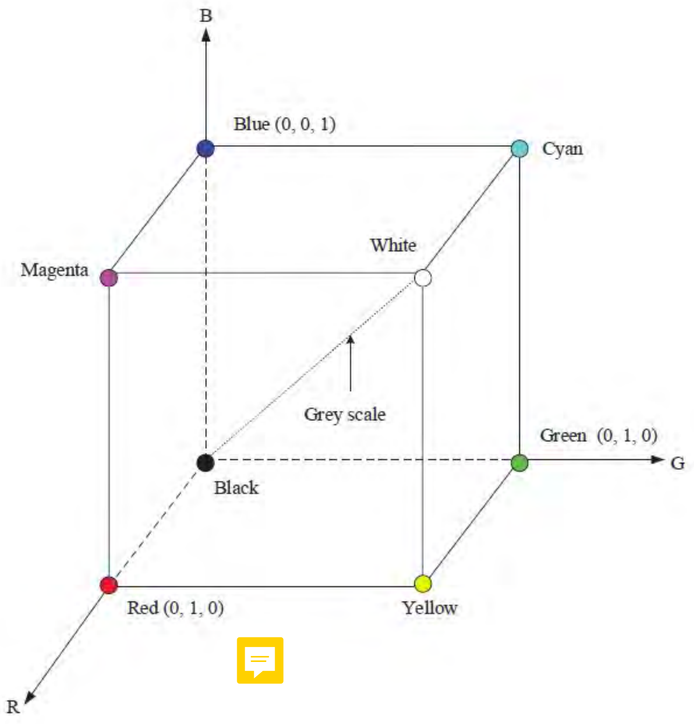


Figure 4.1: RGB color space [64]

As can be seen in Fig. 4.1, RGB color space uses a Cartesian coordinate system to express color. Each axis is representing one on the red, green and blue colors. The values R, G and B can vary within the range of $[0,1]$. A white color can be expressed with a location in the Cartesian system. Location of each color and the origin can be two corners of an imaginary cube. Therefore, three primary colors are this cube's three corners on the axes. Red [1,0,0], Green [0,1,0], Blue [1,0,0], and other three corners are secondary primary colors (or primary colors of pigment which are Yellow [1,1,0], Magenta [1,0,1] and Cyan [0,1,1]). The origin corresponds to Black [0,0,0] and white is the farthest corner from the origin [1,1,1]. A line that connects black to white constitutes the gray-scale spectrum in which the color is represented as [c,c,c] where c $\in[0,1]$.

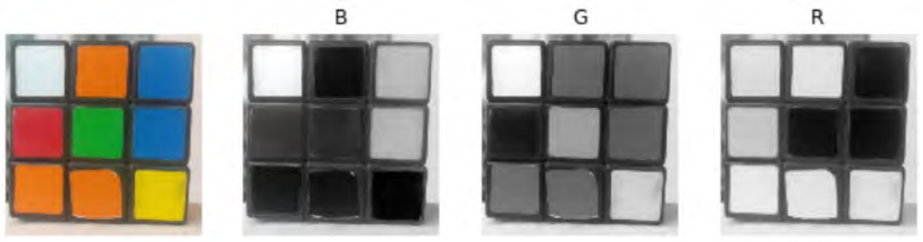


Figure 4.2: A RGB image and its components

4.1.2 HSI color space

The HSI color space is similar to human perception of color. In a human's eyes, a color is described with three elements: hue, saturation and brightness. Same thing virtually happens in the HSI color space; this model considers each color with three components: hue, saturation and intensity. The importance of this color space is that it works exceptionally well in image processing algorithms as the color definitions are natural, intuitive and ideal to human eyes. Fig. 4.3 illustrates this color space.

The hue element (H) corresponds to an angle on a circle which represents the chrominance of a color. $(H \in [0°,360°])$ Red, green and blue as the primary numbers have $120°$ distance and the primary colors of pigment (cyan, magenta and yellow) are exactly in the middle of primary colors. That is, $0°, 120°, 240°$ represent red, green and blue, respectively. Moreover, $60°, 180°$ and $300°$ represent cyan, magenta and yellow. The next component is saturation. Saturation indicate the amount of white mixed in the color and it can vary within the range of $[0, 1]$. Intensity is the last component in this color space and similar to saturation with the range of $[0, 1]$ where 0 means black and 1 means white. As the Fig. 4.3 shows, hue is more meaningful when saturation approaches 1 and less meaningful when saturation approaches 0 or when intensity approaches 0 or 1. Intensity also limits the saturation values.

Any digital image taken from a digital camera or a scanner is originally represented in RGB color space. Therefore, in order to represent the image in any other color space, a conversion formula is needed. The following equations are used to derive HSI color space from RGB [64]:

$$
\begin{aligned}
H &= \begin{cases} \Theta, & \text{if}(B \leq G) \\ 360 - \Theta, & \text{if}(B > G) \end{cases} \\
\text{where} \quad \Theta &= \cos^{-1}\left(\frac{\frac{1}{2}((R-G)+(R-B))}{[(R-G)^2 + (R-B)(G-B)]^{\frac{1}{2}}} \right) \\
I &= \frac{R+G+B}{3} \\
S &= 1 - \frac{3}{R+G+B}\min(R, G, B)
\end{aligned}
\tag{4.1}
$$

where R, G and B in this formula are the values of red, green and blue in the RGB color space.

4.1.3 HSV color space

HSV color space is a cylindrical coordinate system. Its components are hue, saturation and value. This color space converts the RGB Cartesian color space to a cylindrical color space and as a result, the color representation becomes more intuitive and perceptional for human eyes. Therefore, color space can be used in image processing algorithms.

As can be seen in Fig. 4.5, hue is the angle of arc around the vertical central axis. Saturation

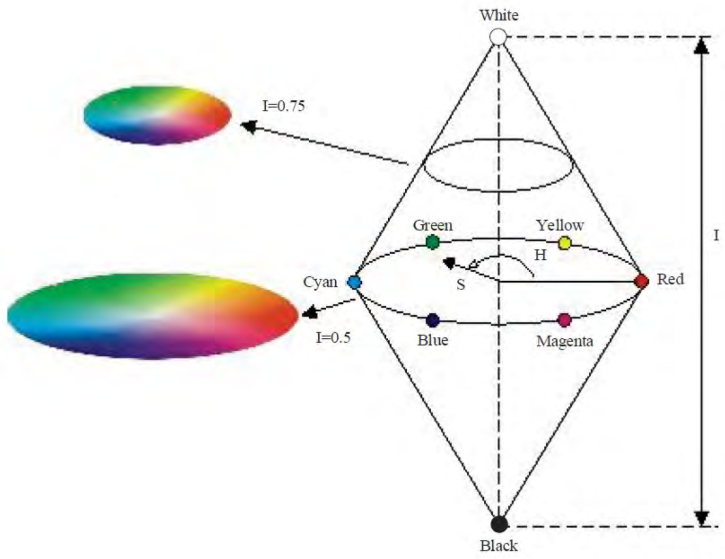


Figure 4.3: HSI color space [64]

can be defined as the horizontal distance from this axis and finally, value is the vertical distance from the base. Red, green and blue colors' locations are similar to HSI color system. That is, $0°, 120°, 240°$ represent red, green and blue. The vertical central axis makes the gray spectrum from black to white (bottom = 0, top = 1).

As mentioned in Section 4.1.2, digital images are generally represented in RGB color space model. Hence, it is needed to formulate HSV-RGB conversion as bellow:

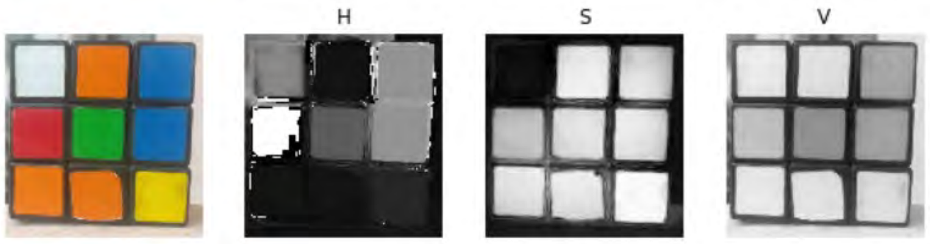


Figure 4.4: HSV components of an image

$$V = \max(R,G,B)$$
$$S = \begin{cases} 0, & \text{if}(\max(R,G,B) = 0) \\ \frac{\max(R,G,B) - \min(R,G,B)}{\max(R,G,B)}, & \text{otherwise} \end{cases}$$
$$H = \begin{cases} \text{undefined}, & \text{if}(S = 0) \\ 60 \times \frac{G-B}{\max(R,G,B)-\min(R,G,B)}, & \text{if}(\max(R,G,B) = R)\&(G \geq B) \\ 60 \times \frac{G-B}{\max(R,G,B)-\min(R,G,B)} + 360, & \text{if}(\max(R,G,B) = R)\&(G < B) \\ 60 \times \frac{B-R}{\max(R,G,B)-\min(R,G,B)} + 120, & \text{if}(\max(R,G,B) = G) \\ 60 \times \frac{R-G}{\max(R,G,B)-\min(R,G,B)} + 240, & \text{if}(\max(R,G,B) = B) \end{cases} \tag{4.2}$$

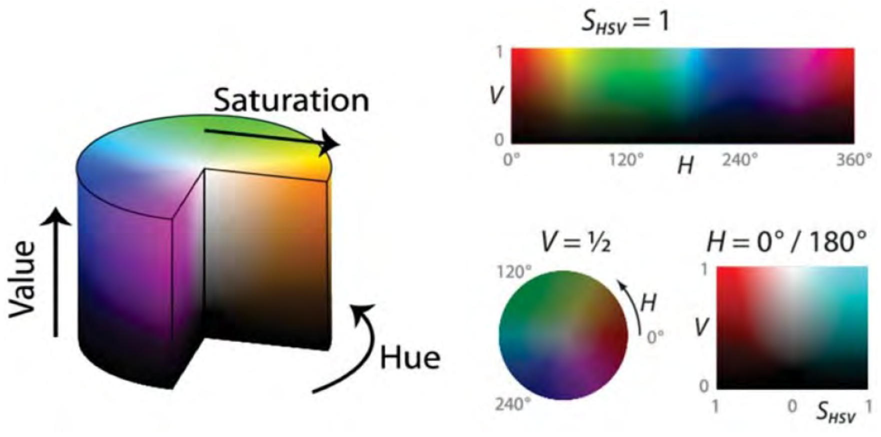


Figure 4.5: HSV color space

4.1.4 Lab color space

The Lab color space (or L*a*b*) is created to mimic the colors in a way that human eyes do. Therefore, its way of expression and its usages outweigh all of other color spaces. This color space does not depend on the device that generates it. A a result, the Lab color space can express colors accurately regardless of the way the images are generated or displayed. As can be seen in Fig. 4.6, the Lab color space is composed of three components: luminance (L), and two chrominance (a and b). Luminance represents the brightness of color, which can range from 0 to 100. Chrominance a indicates the color changes from red to green ranging from -128 to +127. Chrominance b denotes that the color varies from yellow to blue ranging from -128 to +127. Similar to other color spaces, there is a conversion formula that can be used to calculate Lab variables:

$$\begin{aligned} L &= 116 \times (0.299R + 0.587G + 0.114B)^{\frac{1}{3}} - 16 \\ a &= 500 \times [1.006 \times (0.607R + 0.174G + 0.201B)^{\frac{1}{3}} - (0.299R + 0.587G + 0.114B)^{\frac{1}{3}}] \\ b &= 200 \times [(0.299R + 0.587G + 0.114B)^{\frac{1}{3}} - 0.846 \times (0.066G + 1.117B)^{\frac{1}{3}}] \end{aligned} \tag{4.3}$$

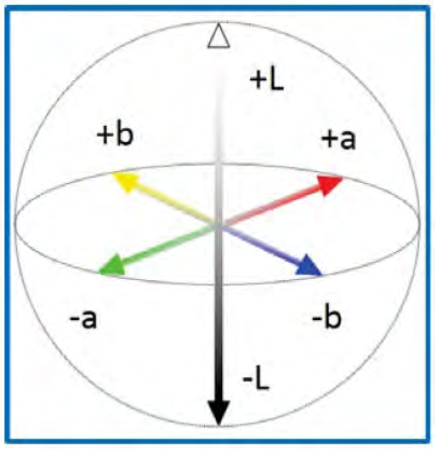


Figure 4.6: Lab color space

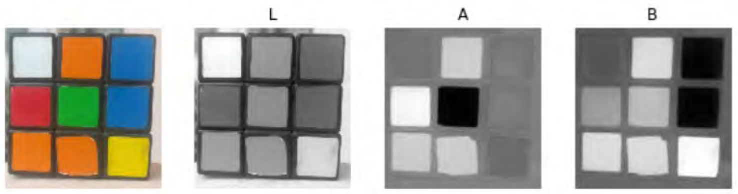


Figure 4.7: Lab components of an image

4.2 Architecture of the thermal sensor and its usage

Based on its datasheet, the thermal sensor needs to run through multiple steps. The first step is properly wiring the sensor to Raspberry Pi. SDA and SCL pins should be connected to the respective pins on the Raspberry Pi. Power supply pins also need to be connected in order to power up the sensor. Fig. 4.8 shows the diagram for connecting the thermal sensor to Raspberry Pi. After hardware connection, it is time to consider the software communication. Using I^2C, one can either read from or write to the sensor. As indicated in the sensor's datasheet [61], in order to read and write, it is needed to generate signals like Fig. 4.9 and Fig. 4.10. It is mainly needed to read from the sensor, but there are sometimes that one should write to the sensor. For example, to configure the sensor's parameters such as refresh rate and resolution of the sensor.

As can be seen in Fig. 4.9 and Fig. 4.10, in order to communicate with the sensor, after calling the device by its 7-bit address (0X33 by default), it is needed to write a 2-Byte internal address. this is required in both reading and writing processes. Every single data stored in the sensor has its unique 2-Byte address and can be accessed only by using its address. Therefore, it is crucial to know how to properly address the data. Fig. 4.11 shows the address map for MLX90640 sensor. ROM is used for internal calculations. RAM is where the data of each pixel is saved. EEPROM holds the constant parameters that are needed for calculating and creating the temperature image. It also contains the individual pixel calibration information. Registers are used for configuring the device. Only some of the registers are writable and other memory locations are read-only.

4.2.1 Writing to the sensor

Based on Fig. 4.10, writing to the sensor is easily done in three steps:

(1) Writing the device address;

(2) Writing the 2-Byte address of the location in which the data should be put;

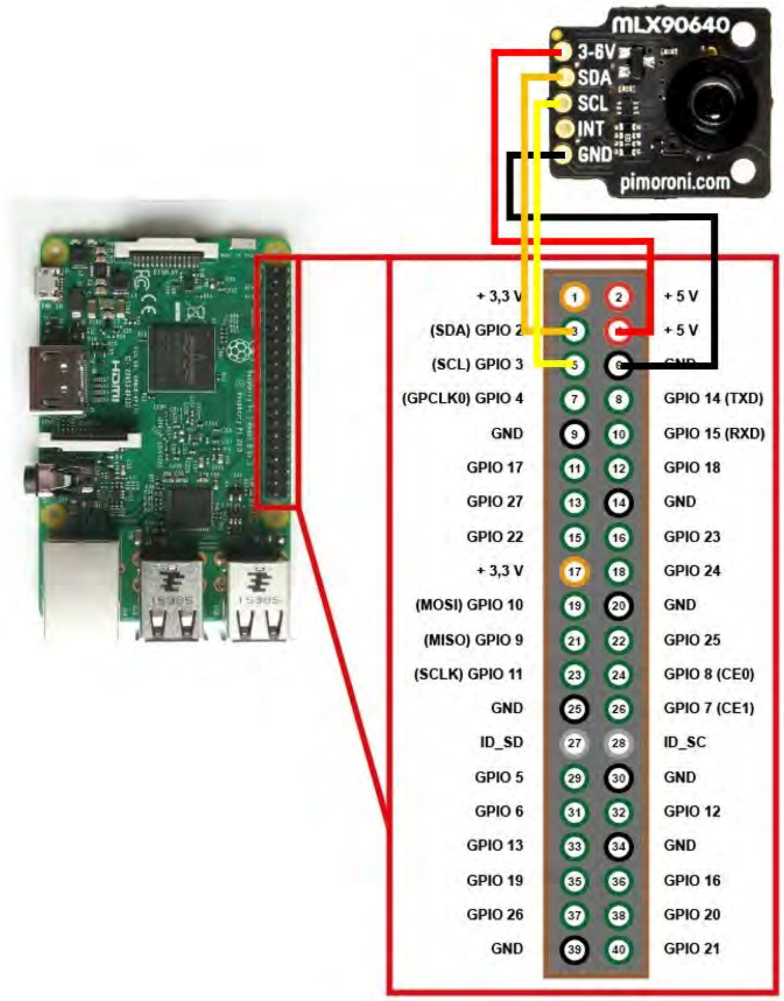


Figure 4.8: Wiring between the thermal sensor and Raspberry Pi

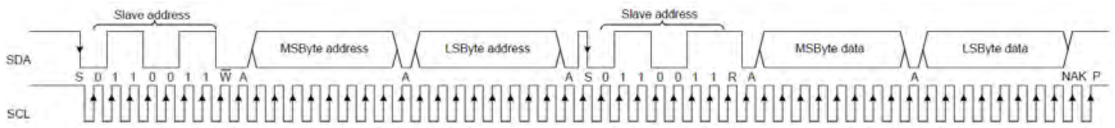


Figure 4.9: I^{2}C read command format

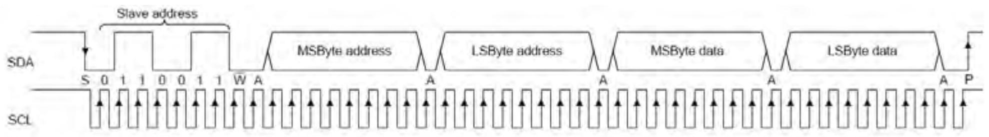


Figure 4.10: I^{2}C write command format

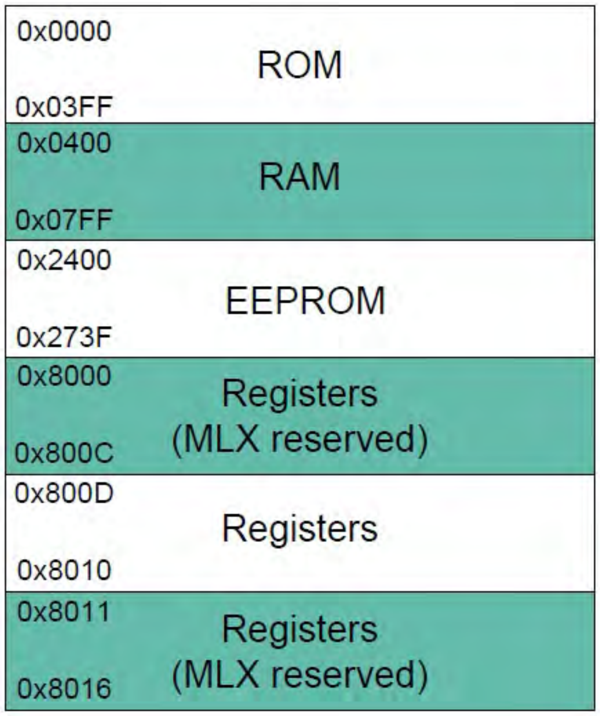


Figure 4.11: MLX90640 memory map [61]

(3) Writing the 2-Byte data.

There are a few internal registers that are customer accessible through which the device performance can be customized. Two of them need to be set properly. They are status register (0x8000) and control register (0x800D)

4.2.1.1 Control register

The control register is used for multiple purposes. First, it is needed to get familiar with the way the sensor captures data. It has a 32×24 pixel IR array sensor and each time only half of the data can be read. Therefore, there are two methods of reading: one is called TV mode and in each reading, it reads either the odd or the even rows (Fig. 4.12). The other method is called the chess reading and each time it reads pixels that are in a hypothetical chess board and have same color (Fig. 4.13). In this research chess mode is chosen.

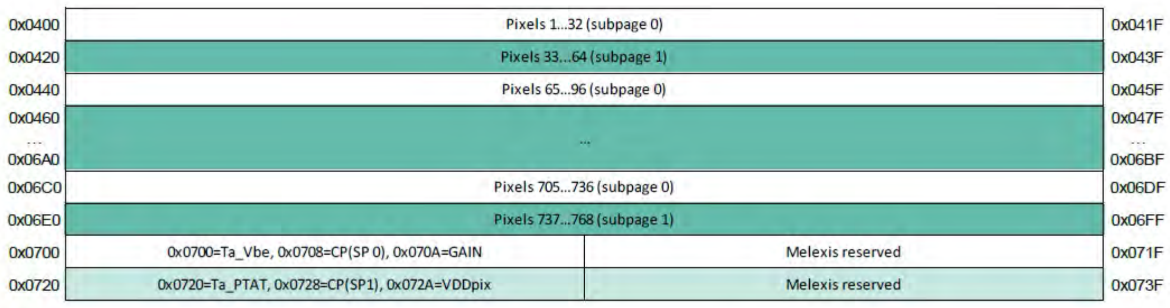

0x0400
Pixels 1...32 (subpage 0)
0x041F
0x0420
Pixels 33...64 (subpage 1)
0x043F
0x0440
Pixels 65...96 (subpage 0)
0x045F
0x0460
0x047F
0x06A0
0x06BF
0x06C0
Pixels 705...736 (subpage 0)
0x06DF
0x06E0
Pixels 737...768 (subpage 1)
0x06FF
0x0700
0x0700=Ta_Vbe, 0x0708=CP(SP 0), 0x070A=GAIN
Melexis reserved
0x071F
0x0720
0x0720=Ta_PTAT, 0x0728=CP(SP1), 0x072A=VDDpix
Melexis reserved
0x073F

Figure 4.12: TV mode reading pattern [61]

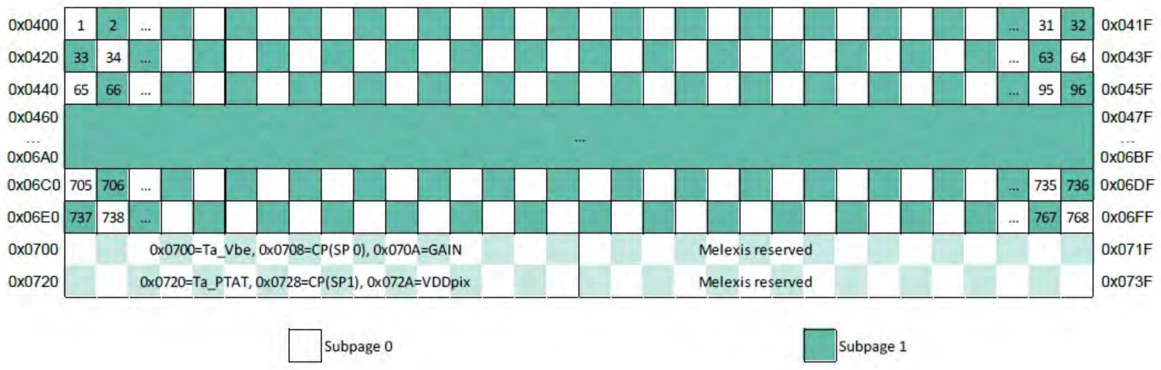
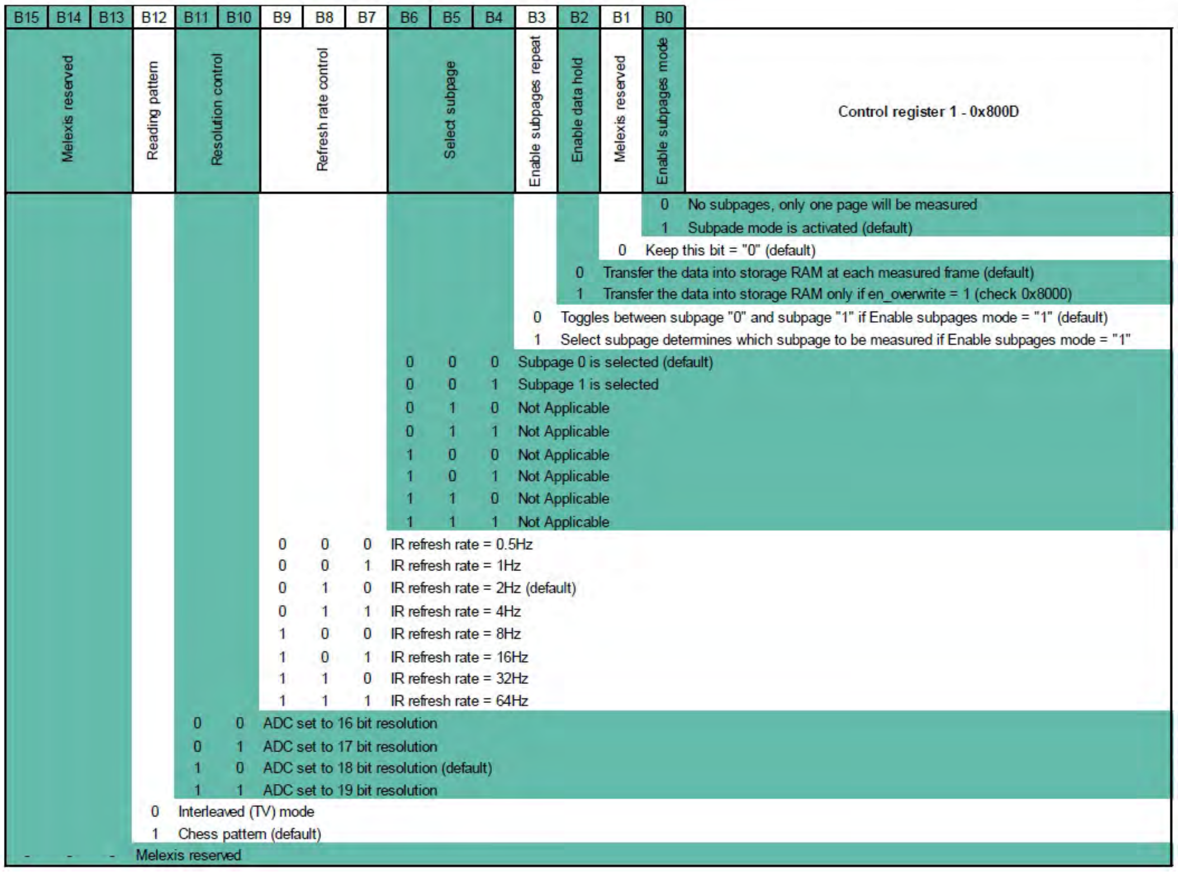


Figure 4.13: Chess reading pattern [61]

In a 2-Byte data, bits can be names as B15 to B0. B15 is the bit with the highest value and B0 is the bit with the lowest value. In the control register, B15 to B13 are reserved. B12 is used to choose the reading pattern. 1 is for TV mode and 0 is for chess mode. B11 and B10 are used for setting the resolution of ADC reading which can vary from 16-bit to 19-bit. B9 to B7 are for refresh rate which is an important factor. The refresh rate can be from 0.5Hz to 64Hz. As it increases, noises in the measurement will rise. Therefore, choosing the proper refresh rate is necessary. The effect of refresh rate on noise level is discussed in Section 4.2.3. B6 to B4 shows which subpage is selected. B3 determines if it will toggle between subpages or not. B2 in used to decide whether to move data automatically to RAM or by setting a register to 1. B1 is reserved and finally B0 is used to see if subpage mode is activated or not.



B15	B14	B13	B12	B11	B10	B9	B8	B7	B6	B5	B4	B3	B2	B1	B0	
Melexis reserved			Reading pattern	Resolution control		Refresh rate control			Select subpage			Enable subpages repeat	Enable data hold	Melexis reserved	Enable subpages mode	Control register 1 - 0x800D
															0	No subpages, only one page will be measured
															1	Subpade mode is activated (default)
														0		Keep this bit = "0" (default)
													0			Transfer the data into storage RAM at each measured frame (default)
													1			Transfer the data into storage RAM only if en_overwrite = 1 (check 0x8000)
												0				Toggles between subpage "0" and subpage "1" if Enable subpages mode = "1" (default)
												1				Select subpage determines which subpage to be measured if Enable subpages mode = "1"
									0	0	0					Subpage 0 is selected (default)
									0	0	1					Subpage 1 is selected
									0	1	0					Not Applicable
									0	1	1					Not Applicable
									1	0	0					Not Applicable
									1	0	1					Not Applicable
									1	1	0					Not Applicable
									1	1	1					Not Applicable
						0	0	0								IR refresh rate = 0.5Hz
						0	0	1								IR refresh rate = 1Hz
						0	1	0								IR refresh rate = 2Hz (default)
						0	1	1								IR refresh rate = 4Hz
						1	0	0								IR refresh rate = 8Hz
						1	0	1								IR refresh rate = 16Hz
						1	1	0								IR refresh rate = 32Hz
						1	1	1								IR refresh rate = 64Hz
				0	0											ADC set to 16 bit resolution
				0	1											ADC set to 17 bit resolution
				1	0											ADC set to 18 bit resolution (default)
				1	1											ADC set to 19 bit resolution
			0													Interleaved (TV) mode
			1													Chess pattern (default)
-	-	-														Melexis reserved

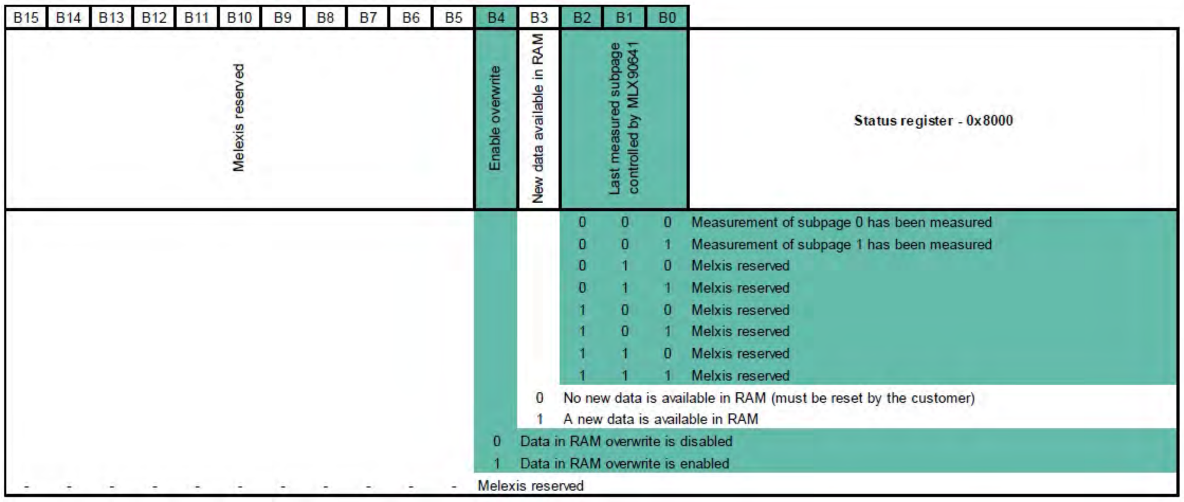
Figure 4.14: Control register bit meaning [61]

4.2.1.2 Status register

The status register is used for three purposes: B15 to B5 registers are reserved. B4 is used to enable or disable overwrite process. B3 is for checking whether a new data is available in RAM or not. If it is equal to 1 it means a new data is available and then it should be reset to zero for further usage. B2 to B1 are to check which subpage is measured.

4.2.2 Reading from the sensor

Unlike writing to the sensor, reading is not as easy as writing. In one reading, either one 2-Byte data or a consecutive set of data can be read from the sensor. The process consists of more steps:



B15	B14	B13	B12	B11	B10	B9	B8	B7	B6	B5	B4	B3	B2	B1	B0	
Melexis reserved											Enable overwrite	New data available in RAM	Last measured subpage controlled by MLX90641			Status register - 0x8000
													0	0	0	Measurement of subpage 0 has been measured
													0	0	1	Measurement of subpage 1 has been measured
													0	1	0	Melxis reserved
													0	1	1	Melxis reserved
													1	0	0	Melxis reserved
													1	0	1	Melxis reserved
													1	1	0	Melxis reserved
													1	1	1	Melxis reserved
												0	No new data is available in RAM (must be reset by the customer)			
												1	A new data is available in RAM			
											0	Data in RAM overwrite is disabled				
											1	Data in RAM overwrite is enabled				
-	-	-	-	-	-	-	-	-	-	-	Melexis reserved					

Figure 4.15: Status register bit meaning [61]

(1) Writing the device address;

(2) Writing the address of the first data to be read;

(3) Writing the device address (again) - here is the starting of the reading process;

(4) Reading 2-Byte data of the first data;

(5) (Optional) if more data needs to be read, a not acknowledge bit (NoACK) is sent to the SDA line by holding it to HIGH;

(6) Reading the next data;

(7) Repeat from 5 until the whole data is read.

4.2.3 Effect of refresh rate on noise level

A test in conducted on the effect of refresh rate on noise level. Generally, as the refresh rate increases, so does the noise level. Therefore, it is important to choose the proper refresh rate based on the allowed noise level. Fig. 4.16 demonstrates 50 consecutive measurements of a point with different refresh rates. Table 4.1, shows the statistical data of this test and it shows that the range and standard deviation increases as the frequency of device rises but, the average measurement will stay unchanged. Therefore, an effective measurement strategy can be averaging between multiple measurements.

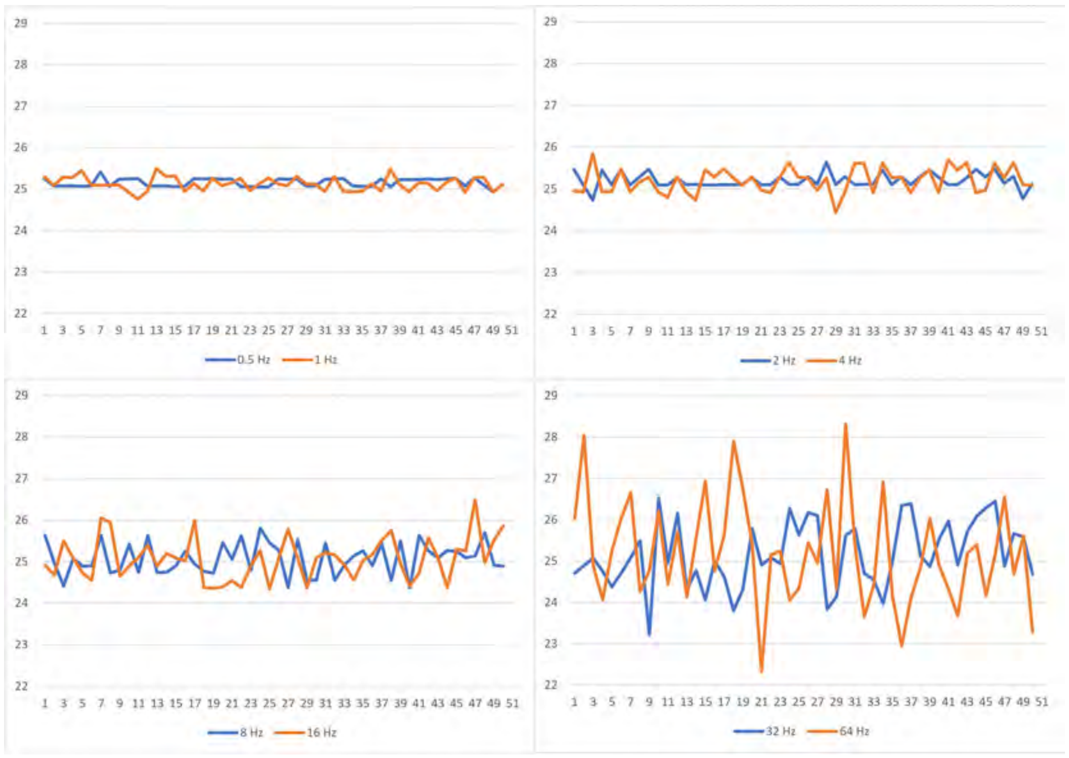


Figure 4.16: Effect of refresh rate on noise in temperature measurement

4.2.4 Temperature calculation

After reading data from a subpage, multiple steps should be done to create the final thermal image. First, the resolution of the sensor should be restored. Then the supply voltage value is calculated. This value is common for all pixels. Next, the ambient temperature is calculated. After this, a series of calculations are done for every pixel in order to find out its temperature data. Fig. 4.17 shows all of the steps to generate final thermal image.

Table 4.1: Effect of refresh rate on the noise level

Refresh rate	Minimum	Average	Maximum	Range	Standard deviation
0.5 Hz	24.93	25.15	25.41	0.48	0.10
1 Hz	24.76	25.13	25.49	0.73	0.16
2 Hz	24.73	25.20	25.64	0.91	0.17
4 Hz	24.43	25.20	25.84	1.40	0.30
8 Hz	24.36	25.07	25.80	1.43	0.39
16 Hz	24.34	25.07	26.48	2.14	0.51
32 Hz	23.22	25.16	26.52	3.30	0.80
64 Hz	22.34	25.19	28.32	5.98	1.26

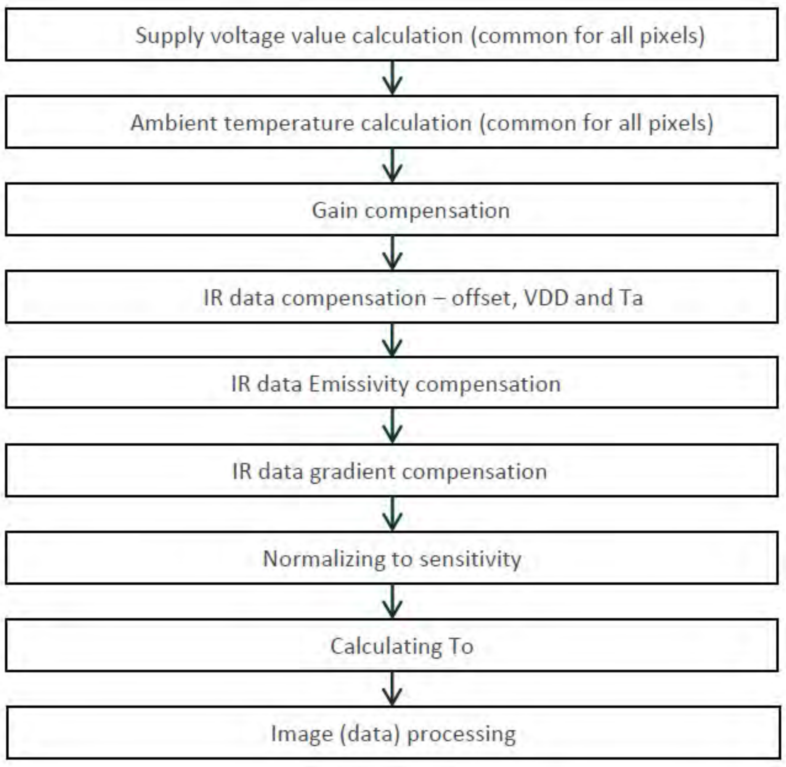

Supply voltage value calculation (common for all pixels)
Ambient temperature calculation (common for all pixels)
Gain compensation
IR data compensation – offset, VDD and Ta
IR data Emissivity compensation
IR data gradient compensation
Normalizing to sensitivity
Calculating To
Image (data) processing

Figure 4.17: Temperature calculation flow

The output of these calculations is an array of temperatures. Table 4.2 shows an example of the temperature image. This image is then converted to either a grayscale image or a spectrum-temperature image. In this case, any color represents a temperature and low temperature pixels tend to be bluer while high temperature ones are more red.

Table 4.2: Temperature data from thermal camera (half-sized)

24	22	22	23	23	23	22	22	23	22	23	22	22	23	22	22
23	22	22	22	22	22	22	22	22	22	23	23	23	23	24	25
23	22	22	22	22	22	23	23	25	31	32	26	25	28	30	30
23	22	22	22	22	22	22	23	38	59	60	39	30	30	30	29
22	22	21	22	22	22	22	23	40	65	65	46	31	29	28	26
22	22	22	22	22	22	22	22	35	59	59	42	28	26	24	23
21	22	22	22	22	22	23	23	25	35	34	26	23	23	22	23
22	22	21	21	22	22	23	23	22	22	22	22	22	22	22	22
22	22	22	22	22	22	22	22	22	22	22	22	22	22	23	22
22	21	21	22	22	22	22	23	23	22	22	23	23	23	23	23
21	22	22	22	22	22	22	22	22	22	22	23	23	23	22	22
21	22	22	22	22	22	23	22	22	22	22	23	22	22	22	22

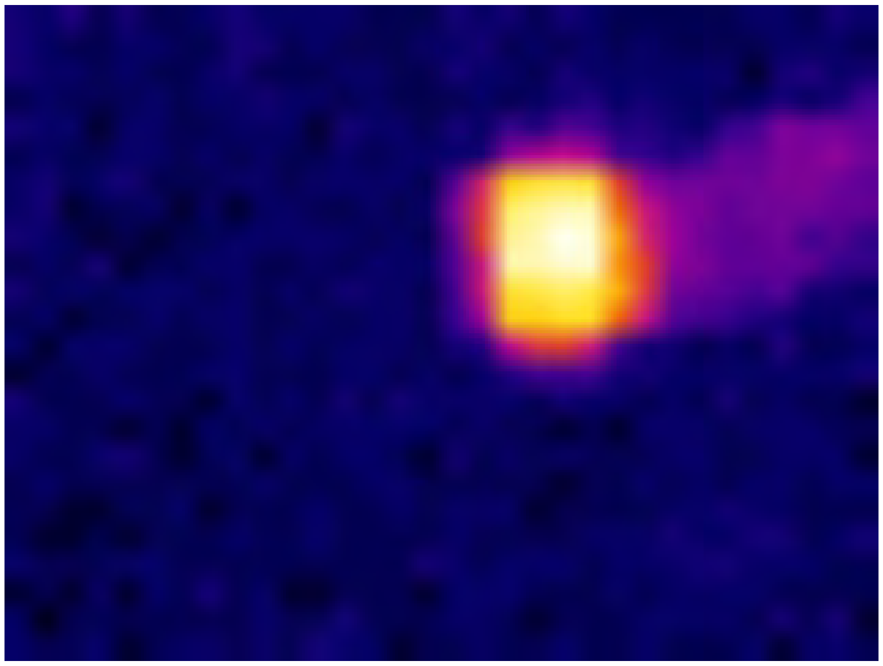


Figure 4.18: Thermal picture of Table 4.2

As can be seen in the Fig. 4.27 and Table 4.2, this is an image of a glass of hot water. At its hottest point, is has a temperature of 65°C and the temperature of environment is around 22°C.

4.3 Fire detection algorithms

In this section, fire detection algorithms used in this research are introduced. First, only visual camera is used to detect the fire. Then, the methods for using only thermal camera are presented. Finally, the fire detection methods by combining both cameras are described. Using the images from both cameras improves the algorithm by reducing the false fire signals. For example, a picture of a fire can be falsely detected as a fire by the visual camera. On the other hand, a hot surface can be detected as fire by the thermal camera. And if properly tuned, the multiple camera system, can improve the fire detection performance by validating the fire signal in two different ways.

4.3.1 Algorithm used in the visual camera

There are numerous different algorithms used with a visual camera to detect forest fires. Algorithms that use color threshold, or flame characteristics, or smoke features. From another point of view, these algorithms can be categorized into color-based and motion-based algorithms. A color-based fire detection algorithm uses the color features of flame, smoke or other fire characteristics to decide whether a fire is detected or not. On the other hand, motion-based fire detection algorithms rely on the motion feature of flame or smoke in their decision-making process. Even recently with the help of more intelligent methods such as machine learning and neural networks, more effective algorithms are implemented. These methods need high power computers and a long process of learning. Therefore, they need either a long time for setup and pre-operation activities or they need more time to process the images to deliver a result. This means that they cannot work in real-time. Using two cameras, will make it possible to combine two simple detection methods to get a fairly effective fire detection system. Therefore, there is no need of learning and pre-operation activities and also it can work in real-time. Fig. 4.19 illustrates the flowchart of color-based fire detection algorithm.

As mentioned above, one of the problems of using visual camera is that it cannot distinguish

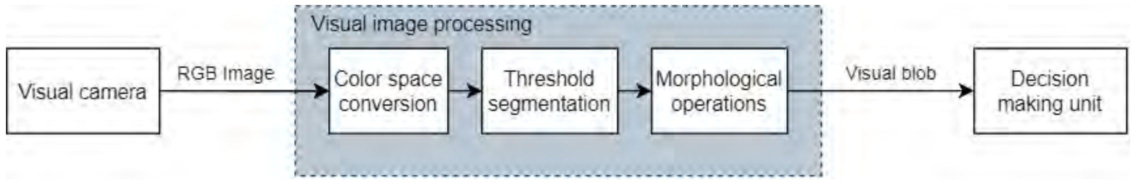


Figure 4.19: Flowchart of color-based fire detection algorithm

between a real fire and an image of a fire. Despite this fact, in many cases, a simple visual camera can be a significant help to firefighters. As discussed in Section 4.1, some of the color spaces that can be used in fire detection algorithms are RGB, HSV, HSI and Lab color spaces. In this section, some of the possible fire detection algorithms are presented. As the HSV color space has many similarities with the HSI color space, only one of them is investigated in this section.

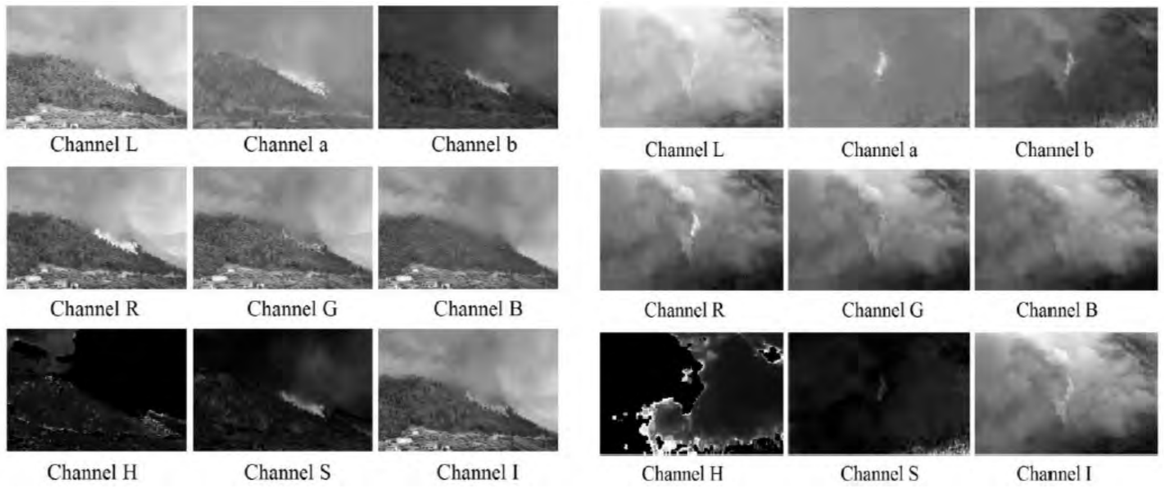


Figure 4.20: Three components of color spaces for two forest fire images [16].

4.3.1.1 Fire detection using RGB color space

An image consists of three color components: red, green and blue, known as RGB. Any color in an image can be presented by its base colors. R, G and B range from 0 to 255. For example, a pixel with RGB factors of (0,0,0) represents black and (255,255,255) is white. A simple fire detection algorithm that can be implemented easily by using a threshold on the red part of an image. For example, if R>200 then fire happens. An example of this, is shown in Fig. 4.21. As can be seen in these figures, this method can generate false fire alarms; because a white pixel has R>200. To improve this method, another condition for R>G>B can be added. This condition is always true for a fire flame because its blue is at the lowest while its red is at its maximum. Using these two conditions can fairly detect most of the visible fires. Fig. 4.22 shows that by having these two conditions, a fire flame can be detected. Although, these methods are not completely reliable, but if combined with a thermal sensor, they can produce acceptable results.

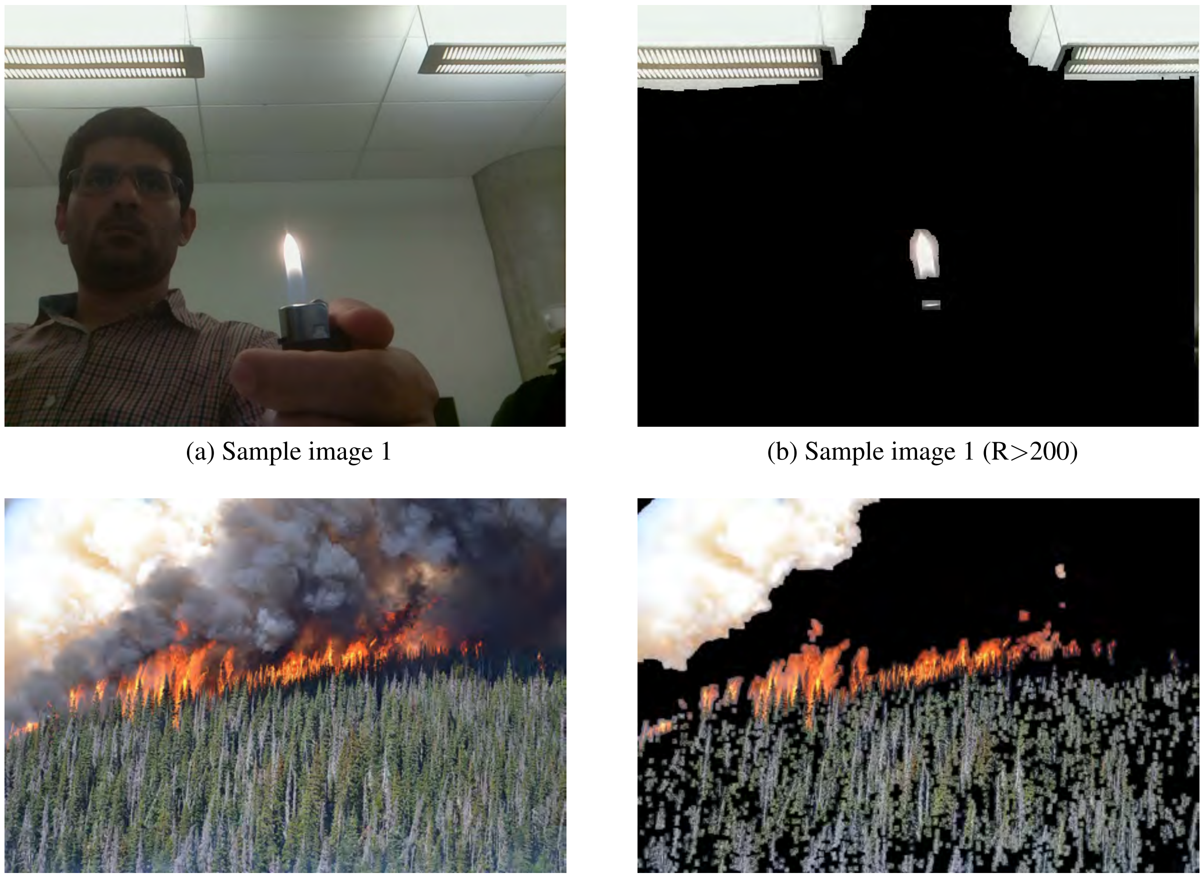
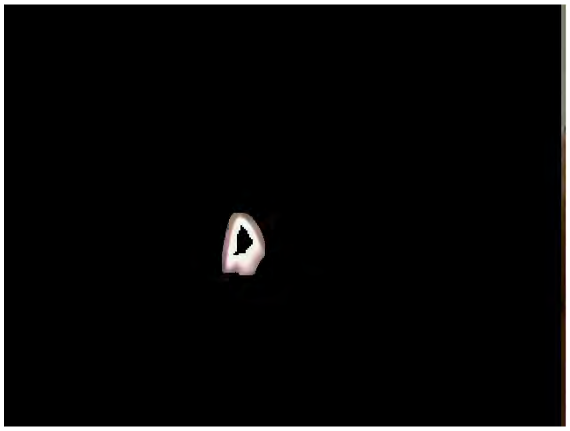
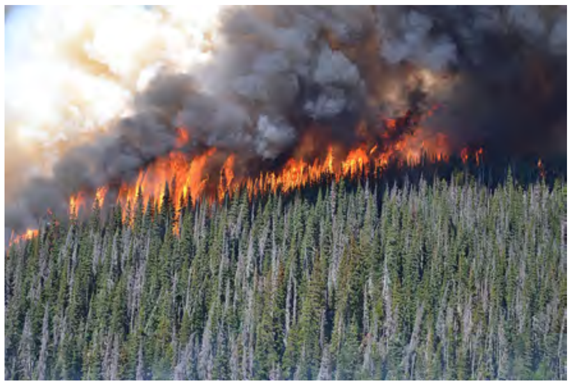

(a) Sample image 1
(b) Sample image 1 (R>200)
(c) Sample image 2
(d) Sample image 2 (R>200)

Figure 4.21: Simple thresholding

(a) Sample image 3


(b) Sample image 3 (R>200 and R>G>B)


(c) Sample image 2

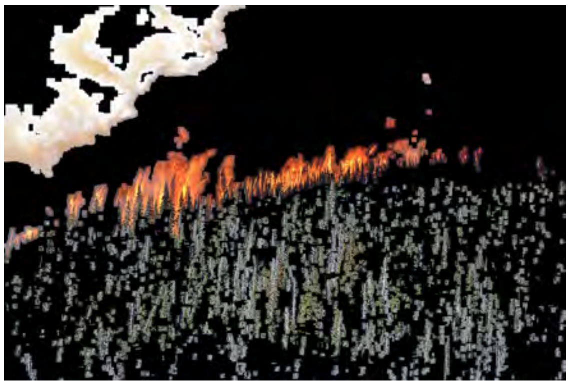

(d) Sample image 2 (R>200 and R>G>B)

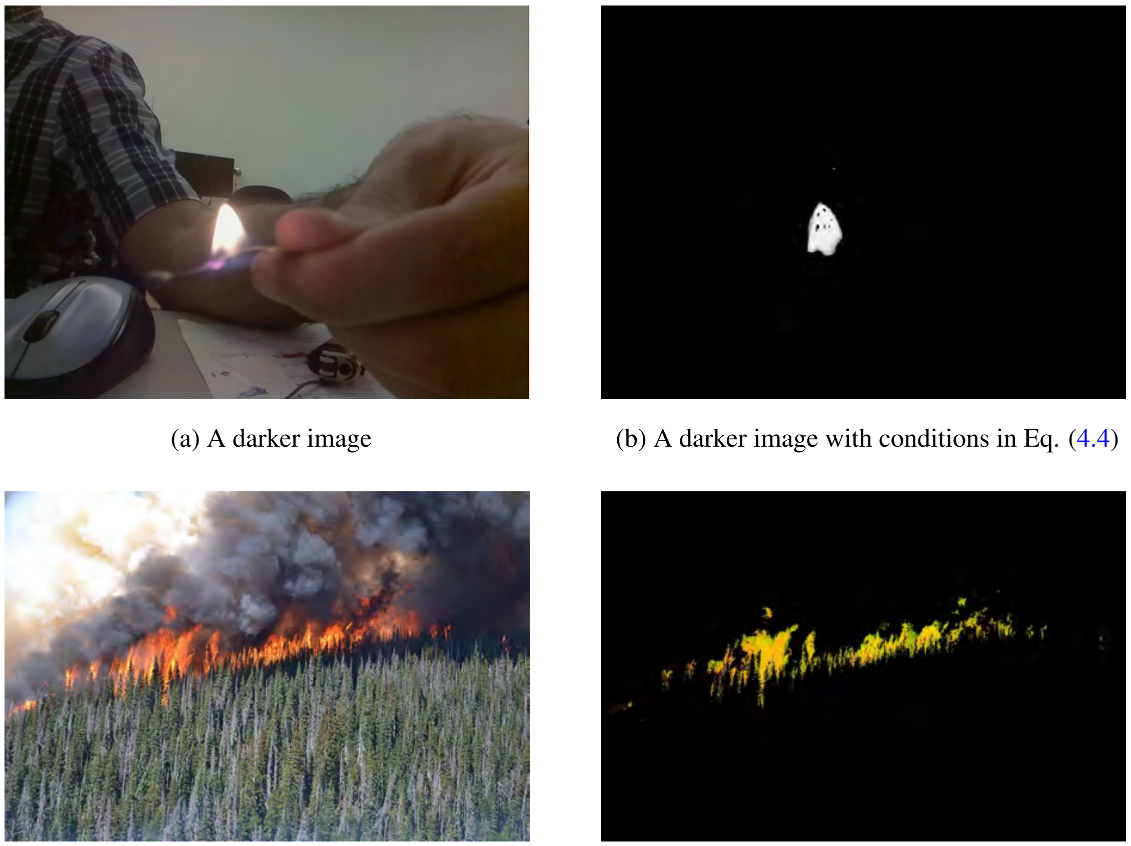
Figure 4.22: Effect of adding R>G>B rule

4.3.1.2 Fire detection using HSV color space

A fire pixel tends to have the color of orange to red. In the HSV color space, unlike the RGB color space, since the lighting is separated from the color, and it is easier to filter pixels based on their color. This can be represented as below:

$$\begin{cases} \text{Condition 1:} & 0^\circ \leq H \leq 60^\circ; \\ \text{Condition 2:} & \text{Brighter images: } 150 \leq S \leq 255, \\ & \text{Darker images: } 0 \leq S \leq 255; \\ \text{Condition 3:} & \text{Brighter images: } 120 \leq V \leq 255, \\ & \text{Darker images: } 170 \leq V \leq 255; \end{cases} \tag{4.4}$$

Fig. 4.23 shows two images with bright and dark backgrounds. With some tuning, fire flame can be detected in this color space, but still it is not considered as a perfect method to do so. The reason is that it relies on tuning for each light and also it might produce false alarms.


(a) A darker image

(b) A darker image with conditions in Eq. (4.4)

(c) A brighter image

(d) A brighter image with conditions in Eq. (4.4)

Figure 4.23: HSV thresholding

4.3.1.3 Fire detection using Lab color space

Lab color space provides better information regarding flame. The reason is it separates the lighting (Luminance) from colors. Therefore, the issue with the HSV color space is not of any concern here (i.e. having two different conditions for different lighting). Refs. [16] and [43] reported reliable and accurate fire detection by using this color space.

As can be seen in Fig. 4.20, there are two forest fire images with different lighting. The only color space in which both fires have same features, is Lab color space. And the reason is- as mentioned above- the ability to separate light from color in image. It can be seen in Fig. 4.20 and Fig. 4.7, the higher values of a and b components can represent yellow and red and it can be said that a fire pixel usually has this feature. Also, a fire pixel owns a high value of luminance (L component). As a result, a decision-making rule in Lab color space can be initiated as follows.

In each component (L, a or b), the average value $\bar{A}_I$ of all pixels can be computed as below [43]:

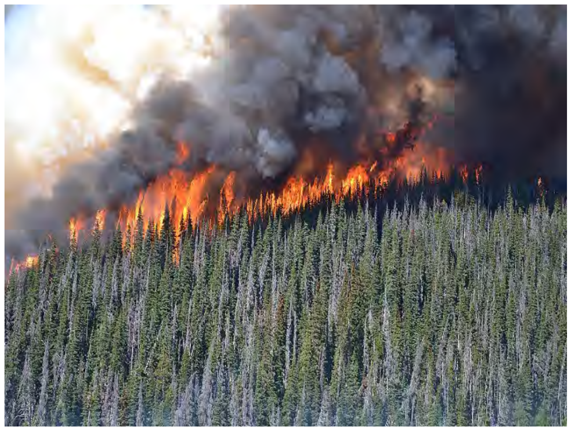
$$\bar{A}_I = \frac{1}{N} \sum_{(x,y)\in I} P_\Phi(x, y), \tag{4.5}$$

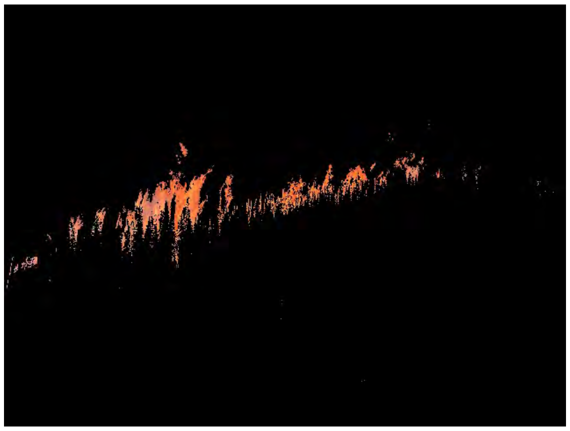
where $P_\Phi(x, y)$ is the value of a pixel at (x, y) position in a component of Lab color space. I is one of the components (L, a or b) and N is the total number of pixels. The decision-making rule (P_F) for fire pixel is formulated as below:

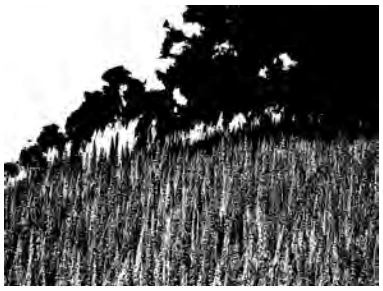
$$\begin{cases} \text{Condition 1:} \quad \bar{A}_L + (MAX_L - \bar{A}_L) \times 0.1 \leq L \leq 255; \\ \text{Condition 2:} \quad \bar{A}_a + (MAX_a - \bar{A}_a) \times 0.2 \leq a \leq 255; \\ \text{Condition 3:} \quad \bar{A}_b + (MAX_b - \bar{A}_b) \times 0.2 \leq b \leq 255; \end{cases} \tag{4.6}$$

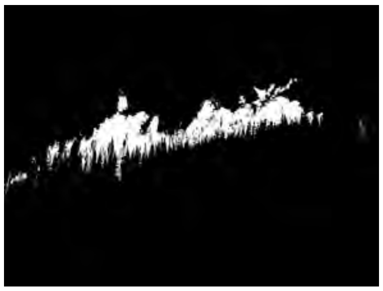
where $\bar{A}_I$ is the average value of all pixels in the I^{th} component and MAX_I is the maximum amount of a pixel value in I. This decision-making rule show acceptable result with different lighting images. Furthermore, since the final goal is to fuse visual image and thermal image, acceptable result in this step leads to a significant result in the next step.

Fig. 4.24 and Fig. 4.26 show two fire images that use Lab fire detection algorithm. They are different in many aspects. First, they have different lighting values. Second, in the first image, there is a white smoke that can be misinterpreted as fire. Third, in the second image, although it seems that fire is obvious, but this image suffers from reddish background. But this method is able to detect fire pixels with a set of rules that are fixed for any condition.


(a) Image 1


(b) Final result with conditions in Eq. (4.6)


(c) Condition 1 in Eq. (4.6)


(d) Conditions 2 in Eq. (4.6)

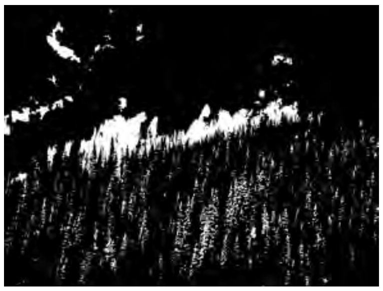
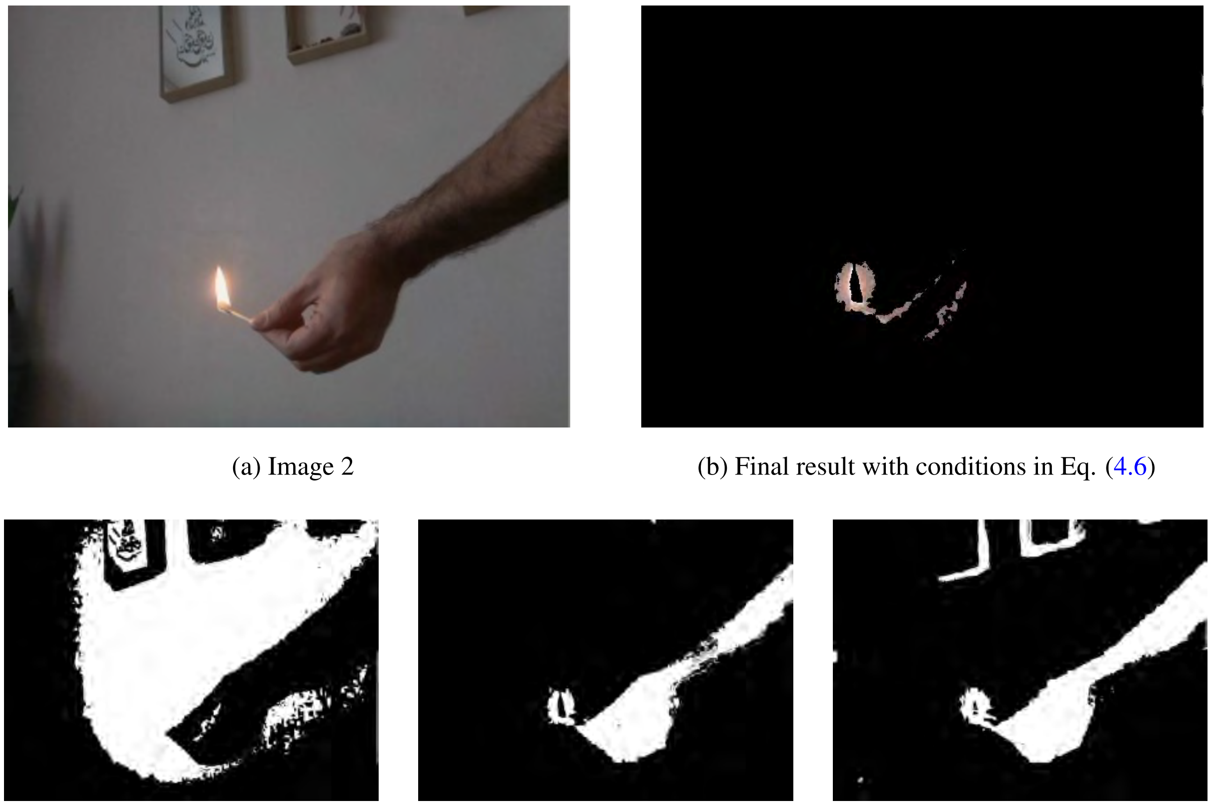

(e) Conditions 3 in Eq. (4.6)

Figure 4.24: Lab decision-making results



(a) Image 2

(b) Final result with conditions in Eq. (4.6)

(c) Condition 1 in Eq. (4.6)

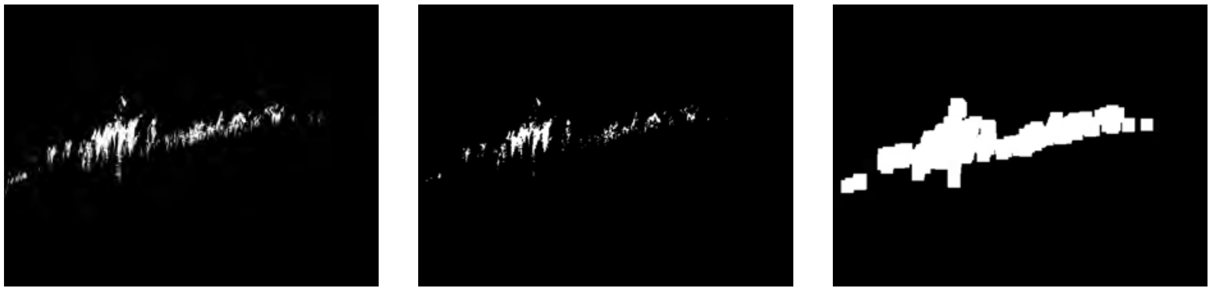
(d) Conditions 2 in Eq. (4.6)

(e) Conditions 3 in Eq. (4.6)

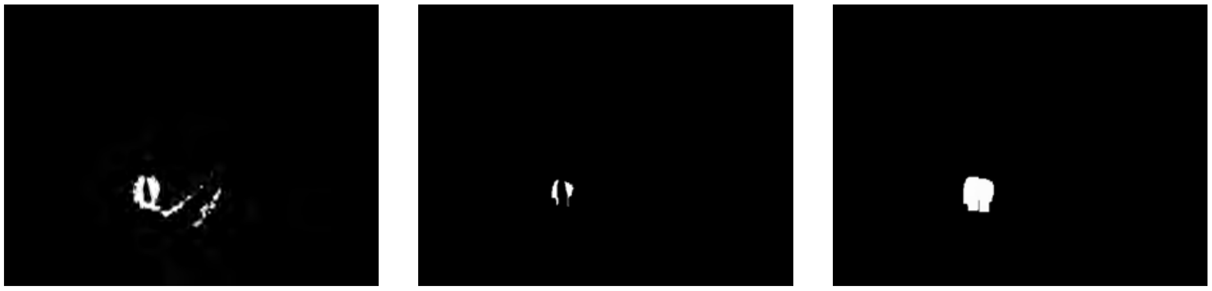
Figure 4.25: Lab decision-making results

4.3.1.4 Morphological operations

The final result of aforementioned procedures can have some small unconcerned areas and pixels. These, pixels can be removed. On the contrary, there are some areas and pixels that need to be added to the fire contour. For example, in Fig. 4.24b and 4.26b the mentioned pixels exist. One way to perform these tasks is by using morphological operations. Morphological operations have a good performance in eliminating uninterested areas and pixels in the thresholding images. Some of the operations are dilation, erosion, opening and closing. In this research, erosion and dilation operations are used. First, by using erosion operation, unwanted small pixels in the boundaries are removed. Next, the dilation operation is used to add wanted pixels that were missed before.



(a) Lab thresholding output (b) Erosion operation (c) Dilation operation



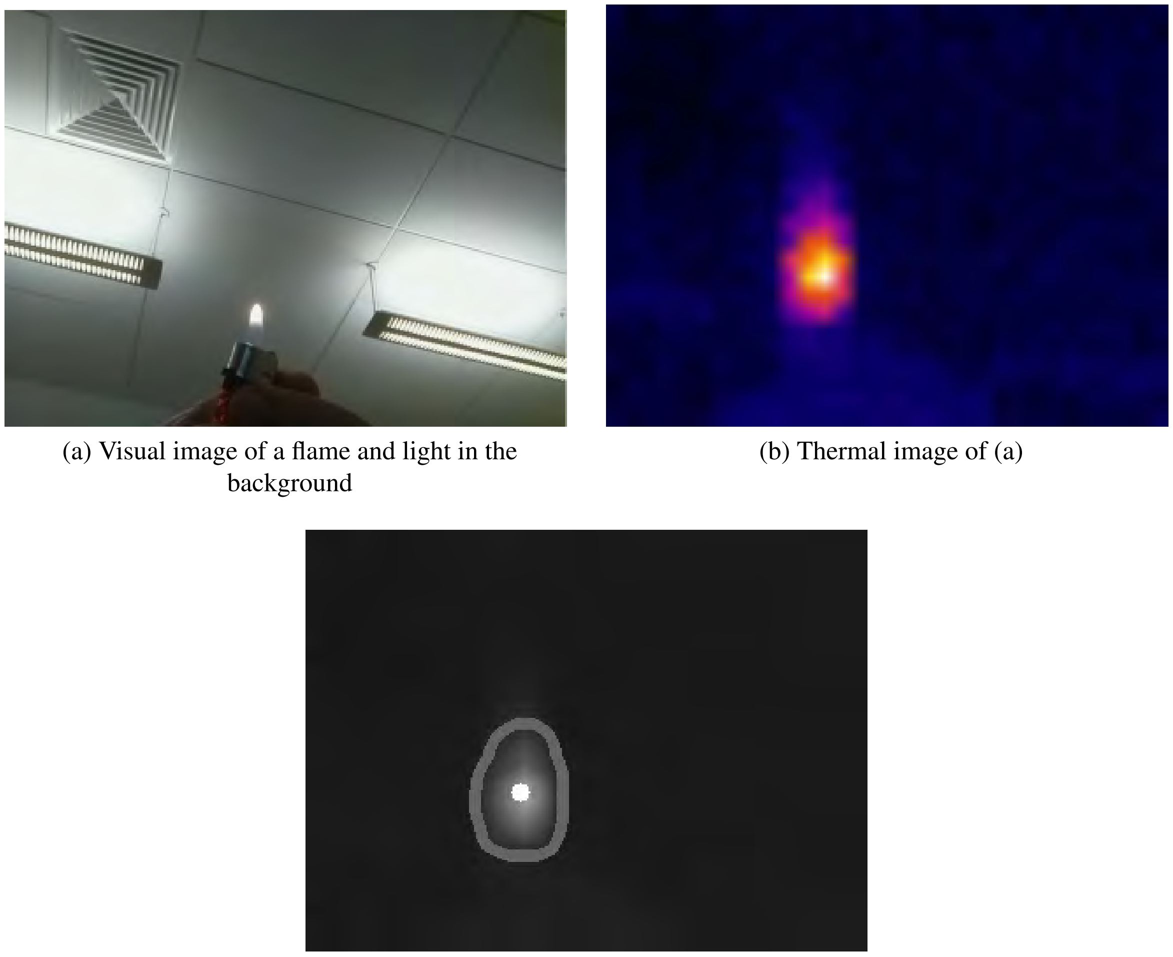
(d) Lab thresholding output (e) Erosion operation (f) Dilation operation

Figure 4.26: Morphological operation's results

4.3.2 Algorithm used in the thermal camera

Unlike a visual camera, fire features in a thermal camera can be easily detected. Therefore, there is no need of expensive computations in the thermal camera. A threshold in the thermal image is applied to filter out low temperature pixels. Also, if there are more than one fire, it can detect them faster. Knowing the maximum temperature of a fire can help us in deciding better how to extinguish it. Finally, it is possible to know the exact area covered by fire even if there are ashes above the fire.

The important point about working with the thermal sensor is to correctly compute the thermal image. i.e. it is needed to write the required code in Python. The written code is in Appendix B.1. A comprehensive program that can detect fire, locate its location with respect to the camera, compute the area covered by the fire, and show the fire in both thermal and visual cameras is presented in Appendix B.2. Although the images from visual camera are presented, but there is no other use of this camera at this point. In the next section, a method for combining data from both cameras is introduced and then its implementation is discussed.


(a) Visual image of a flame and light in the background

(b) Thermal image of (a)

(c) Calculated fire area and its center

Figure 4.27: Thermal fire detection method

4.3.3 Fusing visual and thermal images

One of the main goals of this research is to make use of two cameras to find forest fire with minimum false alarms. In order to do so, there should be a method to fuse the information from both cameras and check the result. Two cameras have different field of views and they may point to different angle. Therefore, an alignment process is needed to make sure an area in an image points to the same area in another image. Fig. 4.28 demonstrates the pinhole camera model. The projective transformation of a world point with (X_W, Y_W, Z_W) coordinates into the image plane with (U, V) coordinates is shown. The mathematical formula is as follows:

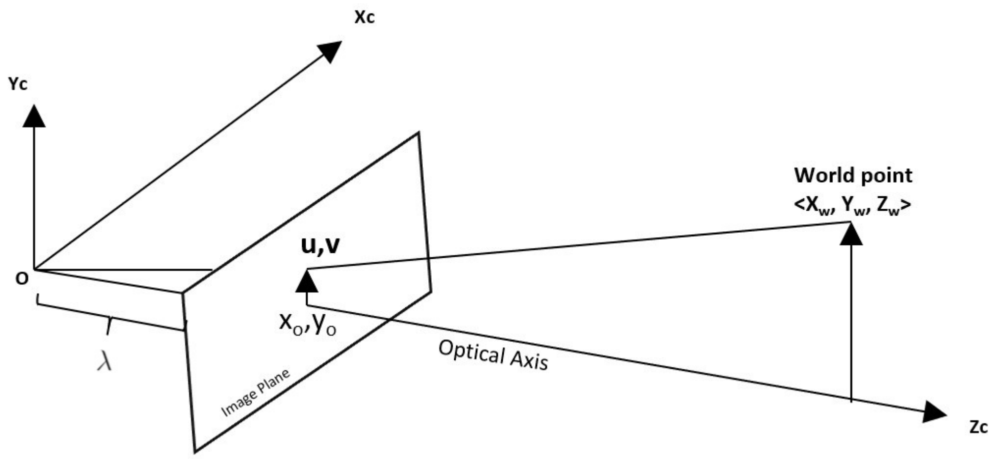


Figure 4.28: Pinhole camera model

Table 4.3: Notations used in the projective transformation

Notation	Meaning
X_W, Y_W, Z_W	World coordinates
X_c, Y_c, Z_c	Camera coordinate system
$R_{3\times3}$	3×3 rotation matrix converting world to camera coordinates
$T_{3\times1}$	3×1 translation matrix moving world to camera coordinates
x_0, y_0	The intersection point of optic axis and the image plane
a	Aspect ratio
s	Skew of the image plane
λ	Focal length of camera
u, v	Projection of X_W, Y_W, Z_W image plane

The projection from a world point (X_W, Y_W, Z_W) to a pixel point (u, v) is computed by using the following equation:

$$\begin{bmatrix} u \\ v \\ w \end{bmatrix} = \begin{bmatrix} \lambda & s & x_0 \\ 0 & a\lambda & y_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} R_{3\times 3} & t_{3\times 1} \\ 0_{1\times 3} & 1 \end{bmatrix} \begin{bmatrix} X_W \\ Y_W \\ Z_W \\ 1 \end{bmatrix} \tag{4.7}$$

This is a "projective transformation". The second matrix to right is the "extrinsic parameter matrix" that transforms (X_W, Y_W, Z_W) world coordinates into (X_C, Y_C, Z_C) camera coordinates by using a rotation matrix $R_{3\times 3}$ and a translation matrix $t_{3\times 1}$. Then, camera coordinates (X_C, Y_C, Z_C) are transformed to pixel coordinates by the leftmost matrix which is called "intrinsic parameters matrix". This matrix includes aspect ratio a, focal length λ, skew of the image plane s. In (x_0, y_0), the optical axis OZ_C goes through the plane of image. The following equation is the result of multiplication of intrinsic and extrinsic parameters matrices:

$$\begin{bmatrix} u \\ v \\ w \end{bmatrix} = \begin{bmatrix} m_{11} & m_{12} & m_{13} & m_{14} \\ m_{21} & m_{22} & m_{23} & m_{24} \\ m_{31} & m_{32} & m_{33} & m_{34} \end{bmatrix} \begin{bmatrix} X_W \\ Y_W \\ Z_W \\ 1 \end{bmatrix} \tag{4.8}$$

or in other words,

$$X^{'} = PX \tag{4.9}$$

where $X^{'} = (u, v, w)$ is the projected coordinates in the homogeneous coordinate system. P is the projective matrix and X is the point being projected to this coordinate system. The image plane can be non-homogeneous in reality. The non-homogeneous image pixels can be calculated by division of the first two elements in the homogeneous matrix to the third one.

In this research, two cameras are used, a visual and a thermal camera to make the fire detection robust. The thermal camera is a 24×32 array. Hence, the fusion requires a projective transformation between the planes of two images. In an image plane, points have two dimensions and the $Z_W = 0$. Therefore, the third column of matrix P is multiplied by zero and it can be removed. This transformation is called "homography" and has the following equation:

$$\begin{bmatrix} x^{'} \\ y^{'} \\ w \end{bmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \tag{4.10}$$

or in other words,

$$X^{'} = HX \tag{4.11}$$

where $X^{'}$ is the point that is projected in homogeneous coordinates. H is the "homography matrix". The projected image coordinates are computed as $u = x^{'}/w$ and $v = y^{'}/w$.

4.3.3.1 Estimating the homography matrix

To calculate the non-homogeneous image points, Eq. (4.10) can be expanded to the following equations:

$$u = \frac{x^{'}}{w} = \frac{h_{11}x + h_{12}y + h_{13}}{h_{31}x + h_{32}y + h_{33}} \tag{4.12}$$

$$v = \frac{x^{'}}{w} = \frac{h_{21}x + h_{22}y + h_{23}}{h_{31}x + h_{32}y + h_{33}} \tag{4.13}$$

The corresponding pixels (x, y) and (u, v) are found between two images either through manual selection of points or automated feature matching. Hence, two equations can be formed for each point. In order to calculate 8 unknown parameters in homography matrix, at least 4 correspondences are needed to estimate this matrix. By rearranging Eq. (4.12) and Eq. (4.13), it is formulated as following:

$$\begin{bmatrix} -x_1 & -y_1 & 1 & 0 & 0 & 0 & x_1u_1 & y_1u_1 & u_1 \\ 0 & 0 & 0 & -x_1 & -y_1 & 1 & x_1v_1 & y_1v_1 & v_1 \\ -x_2 & -y_2 & 1 & 0 & 0 & 0 & x_2u_2 & y_2u_2 & u_2 \\ 0 & 0 & 0 & -x_2 & -y_2 & 1 & x_2v_2 & y_2v_2 & v_2 \\ & & & & \vdots & & & & \end{bmatrix} \begin{bmatrix} h_{11} \\ h_{12} \\ h_{13} \\ h_{21} \\ h_{22} \\ h_{23} \\ h_{31} \\ h_{32} \\ h_{33} \end{bmatrix} = 0 \tag{4.14}$$

or in other words,

$$A\boldsymbol{h} = 0 \tag{4.15}$$

This equation can be solved either by calculating the Singular Value Decomposition (SVD) of A or by the eigenvalues of A^TA matrix. In the SVD method, the last column of V matrix is the estimated value of $\boldsymbol{h}$ vector. If the eigenvalue method is used, the $\boldsymbol{h}$ vector is the eigenvector of the lowest eigenvalues.

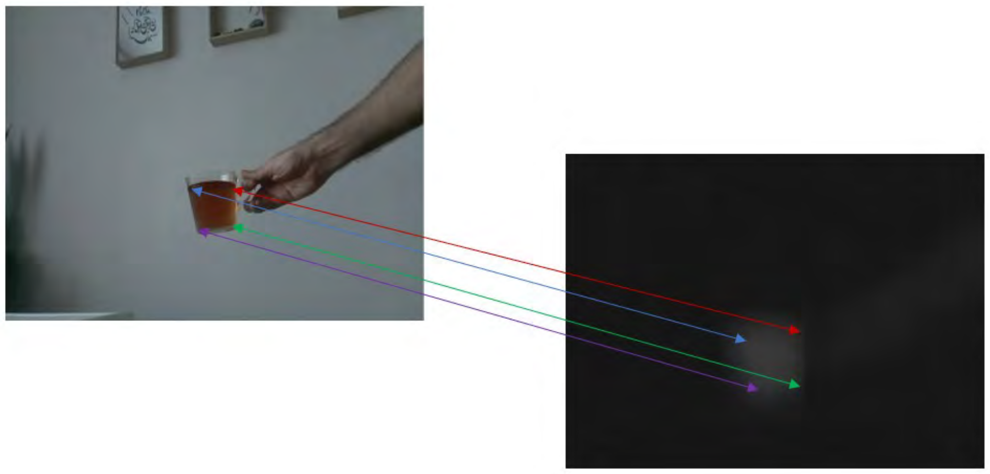
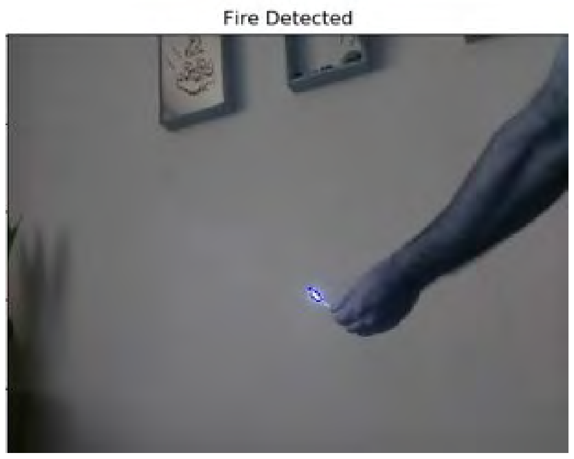
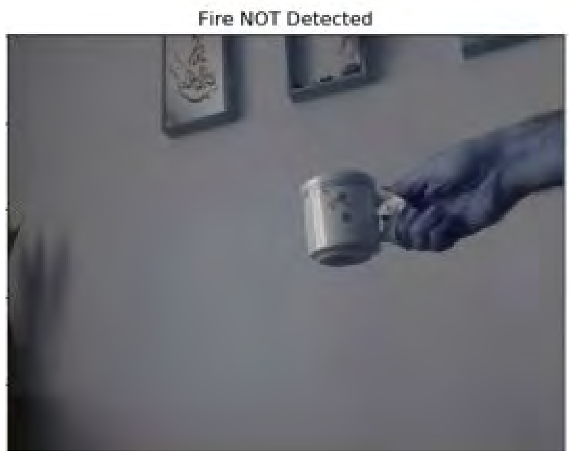


Figure 4.29: Aligning corresponding pixels

In this research, the SVD technique is used for estimating the homography matrix. Corresponding points between visual and thermal images are selected manually. The process is illustrated in Fig. 4.29. Finding the exact correspondent point is challenging because of the different nature between the visual and thermal images. To find the corresponding points, hot red tea in a cup was used because of its visibility in both images with a regular shape. It appears rectangular in both images and the four corners could be somewhat identified. 22 of such images are taken from different distances and angles. These images are used with 4 correspondence points from each pair of images to form the A matrix. Two thresholds were imposed onto the visual and thermal images, resulting in a binary image containing potential fire information. Using the computed homography matrix, the thresholded thermal image was aligned and superimposed onto the thresholded visual image. If the suspected fire region from both images overlapped, it could be concluded that there is a fire in the image.



(a) Fire detected in both images



(b) Fire not detected in both images

Figure 4.30: Fire detection using both camera images

4.4 Image-based thermo-visual servoing by 2DOF frame

The extensive development of visual servo algorithms in the last decade is remarkable. They are applied to unmanned aerial vehicle such as quadrotors, helicopters, airships and airplanes. Visual servoing algorithms make use of visual information to control a system. There are two major branches: position-based visual servoing (PBVS) and image-based visual servoing (IBVS). In PBVS, visual information is used to reconstruct the target's pose in a Cartesian space while in IBVS some image features are extracted and used to control the system. In this research, image-based thermo-visual servoing (IBTVS) is used to find the position of a fire and to move the cameras toward the fire. Image information are extracted from both thermal and visual cameras and then these data are fused to decide whether a fire has happened or not. If a fire is detected, the system calculates the relational position of fire in image coordinate system. This position is sent to the controller and it will rotate cameras toward the fire. This 2DOF frame is made to search and track fire or any hot object. In general, it can follow any object that has different temperature than the background. This system can be applied to many cases. For example, it can be used to track an animal in a dark environment or it can be used to dynamically measure the body temperature of a person.

To create an IBTVS feedback controller for the system that is illustrated in Fig. 3.9, the first step is to create the error function.

$$\begin{bmatrix} e_x \\ e_y \end{bmatrix} = \begin{bmatrix} x_d - x \\ y_d - y \end{bmatrix} \tag{4.16}$$

where e_x and e_y are the errors in x and y directions, and x_d and y_d are the desired positions in x and y directions.

The goal is to track the fire location, in other words, to put the fire in the center of the image of camera. i.e. $x_d = y_d = 0$. Therefore, Eq. (4.16) becomes as following:

$$\begin{bmatrix} e_x \\ e_y \end{bmatrix} = \begin{bmatrix} 0 - x \\ 0 - y \end{bmatrix} = \begin{bmatrix} -x \\ -y \end{bmatrix} \tag{4.17}$$

By imposing the control law of $\dot{e} = Ke$ to Eq. (3.12), a proportional (P) controller is designed as below:

$$\begin{bmatrix} \omega_x \\ \omega_y \end{bmatrix} = \begin{bmatrix} k_1 & 0 \\ 0 & k_2 \end{bmatrix} \times J_{image}^{-1} \begin{bmatrix} e_x \\ e_y \end{bmatrix} = K \times J_{image}^{-1} \begin{bmatrix} e_x \\ e_y \end{bmatrix} \tag{4.18}$$

K is the proportional gain matrix which tunes the convergence rate of $[x\ y]^T$ towards $[0\ 0]^T$. In experimental tests, this means that the UAV will be navigated towards the fire location. Fig. 4.31 illustrates the diagram of the IBTVS feedback controller.

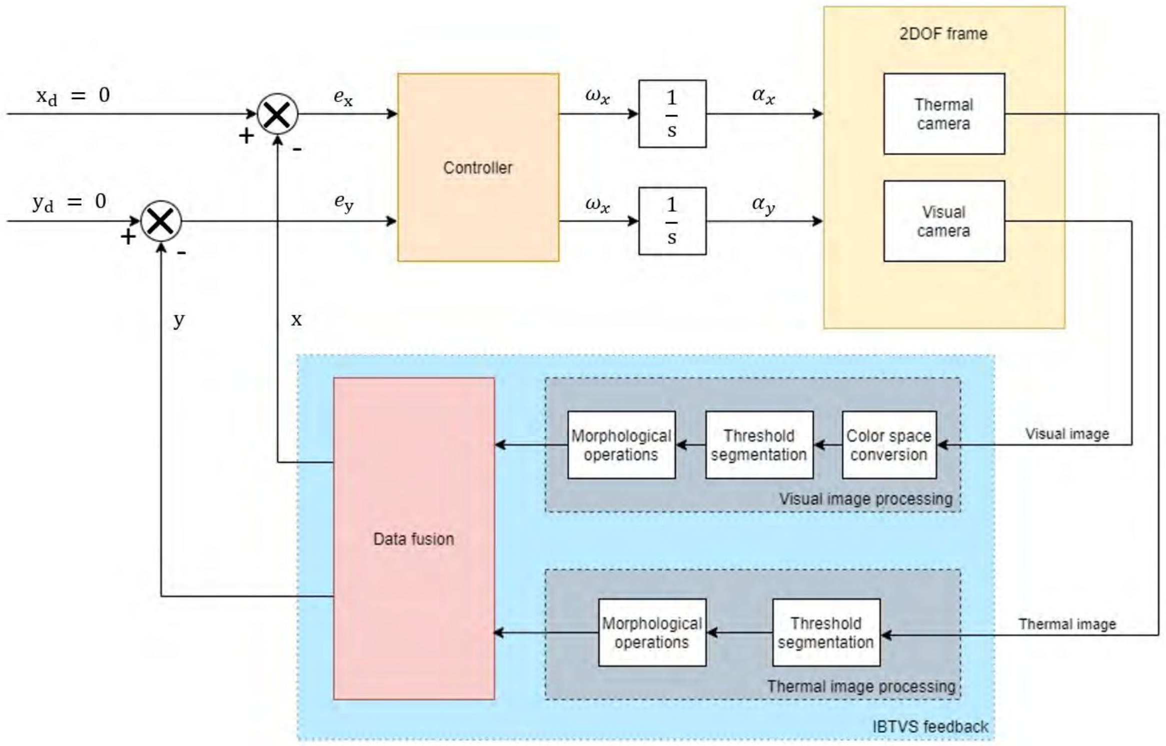


Figure 4.31: Diagram of the IBTVS feedback controller

4.5 Summary

In this chapter, multiple color spaces are introduced. Each of these color spaces can be used in the fire detection algorithm with visual camera. Next, the architecture of thermal camera is discussed including how to read/write data from/to camera and how to calculate the temperature image. Then, the fire detection algorithms that are used in visual and thermal cameras are shown and a method for fusing the images of visual and thermal cameras is implemented. Finally, the IBTVS system which can track and follow the fire is presented.

Chapter 5

Test Platforms

In order to test the system and the concept behind it, two test platforms are made and used. The first platform is a two-degree-of-freedom (2DOF) frame that can rotate in two different axes: yaw and pitch. The goal of using this frame is to check the searching and tracking ability of the system as well as its ability to locate the position of fire.

Originally, it was planned to mount this system on a UAV and test it there. This means, if there would be a real fire and the system would look for the fire and as soon as it finds the fire, it would signal the base and calculate the fire's location and fly over it. But due to the COVID-19 pandemic, the university was closed for a long time and this kind of testing was not possible. Therefore, it was decided to substitute this with a simulation. This is even better in terms of reducing the possibilities of real failures that would cost a lot for the team. Also, even more tests and scenarios are done in the simulation that could not be done easily in the real world. Co-simulation of a UAV means that a virtual UAV with the help of autopilot is flown. In the simulation environment, a location in the map is set as fire location and then a pre-planned mission is loaded into the autopilot of UAV. The goal of this experiment is to check if the developed co-simulation system can find the fire and locates it and flies over it. The combination of these two tests can assure that this system can be installed on a real UAV and then it can be used to search for a fire in a forest and locate the fire.

5.1 Testing of the Co-simulation system

The initial decision was to mount the system on a UAV to test it in practice. But due to the pandemic COVID-19 this was not possible. Therefore, the practical test is replaced by a software test. The previous test showed that by using two cameras (visual and thermal) it is possible to find out fire. The next step is to check if it is possible to communicate with the autopilot of a UAV and command it to move over the fire. This is the goal of current test. In order to run the test, some programs and software are needed to be installed. First, we take a look at DroneKit.

5.1.1 DroneKit API

Every UAV/drone has an autopilot. The autopilot is initially used to stabilize the UAV. Later, it is used to navigate the UAV; i.e. moving the UAV from a point to another. They are capable of operating missions nowadays. This means that some geographical points are uploaded in autopilot and they follow a path from one point to another. But this is not enough for our goal, which is searching for fire and locating it and moving the UAV above it. In order to do so, an intermediary system has to be there to communicate between Raspberry Pi and autopilot. Fortunately, there are some available Kits. One of them is called DroneKit. DroneKit helps us create powerful apps for UAVs. It allows developers to create apps that run on an onboard computer and communicate with the autopilot. Onboard apps can significantly enhance the autopilot, adding greater intelligence to vehicle behavior, and performing tasks that are computationally intensive or time-sensitive (for example, computer vision, path planning, or 3D modelling). Here are some of the main features of DroneKit.

DroneKit features:

- Connect to a vehicle (or multiple vehicles) from a script;

- Get and set vehicle state/telemetry and parameter information;
- Receive asynchronous notification of state changes;
- Guide a UAV to specified position (GUIDED mode);
- Send arbitrary custom messages to control UAV movement and other hardware (GUIDED mode);
- Create and manage waypoint missions (AUTO mode);
- Override RC channel settings.

In this research DroneKit-Python is used because of the available libraries for Python and its capability of developing applications promptly. DroneKit-Python is made available under the permissive open source Apache 2.0 License [65].

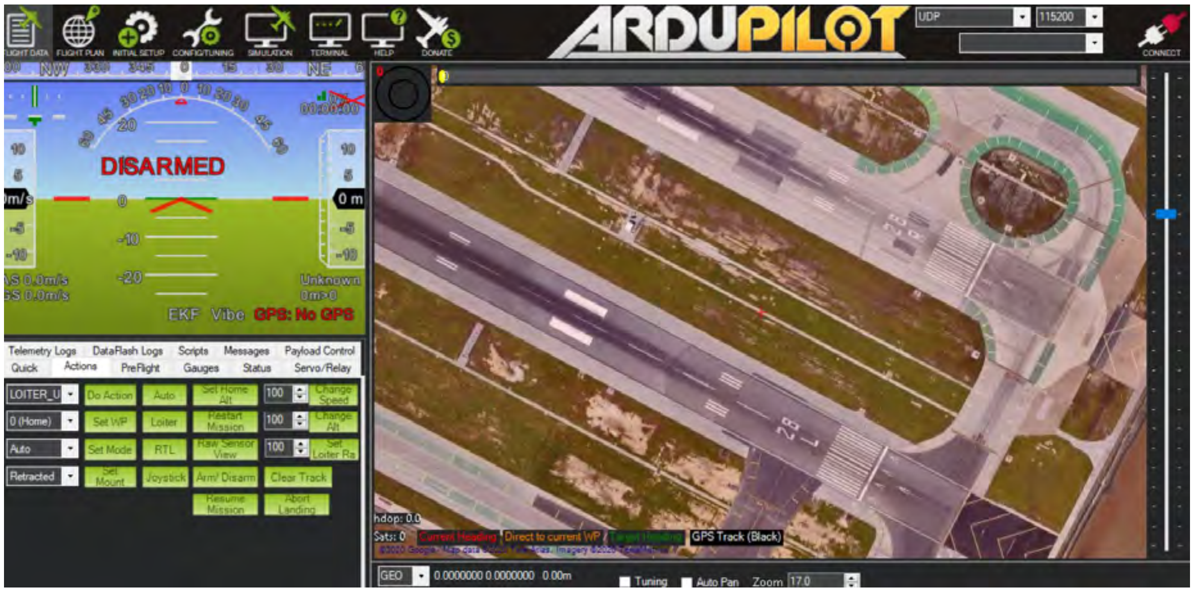
5.1.2 DroneKit SITL

DroneKit can be directly connected to autopilot. But, since we are not using any physical autopilot, there should be a program that simulates the autopilot actions. Fortunately, a program that is compatible with DroneKit is available. This program is called "DroneKit SITL". SITL stands for "software-in-the-loop". This basically simulates the combination of a UAV and its autopilot. It means that it allows us to create and test DroneKit-Python apps without a real vehicle (and from the comfort of our own developer desktop). SITL can run natively on Linux (x86 architecture only), Mac and Windows, or within a virtual machine. It can be installed on the same computer as DroneKit, or on another computer on the same network. In our case, a Linux OS on a virtual machine is installed and everything is installed on the Linux. After installing the program on Linux, in order to run it, this line of code should be run from the command prompt:

```
dronekit-sitl copter --home=37.6135,-122.357,0,180
```

5.1.3 Mission Planner

In order to create a mission and start the autopilot and also visualize the path of UAV, another application is needed. Mission Planner is a ground control station that can be used as a configuration utility or as a dynamic control supplement for an autonomous vehicle, which can setup, configure, and tune the vehicle for optimum performance.



FLIGHT DATA
FLIGHT PLAN
INITIAL SETUP
CONFIG/TUNING
SIMULATION
TERMINAL
HELP
DONATE
ARDUPILOT
UDP
115200
CONNECT
DISARMED
EKF Vibe GPS: No GPS
Telemetry Logs DataFlash Logs Scripts Messages Payload Control
Quick Actions PreFlight Gauges Status Servo/Relay
Do Action Auto Set Home Alt Change Speed
Set WP Loiter Restart Mission Change Alt
Set Mode RTL Raw Sensor View Set Loiter Rad
Set Mount Joystick Arm/Disarm Clear Track
Resume Mission Abort Landing
GPS Track (Black)
Tuning Auto Pan Zoom 17.0

Figure 5.1: Mission Planner interface

5.1.4 FlightGear

FlightGear is an open-source flight simulator. In this research, FlightGear is used to visualize the flight of the UAV. The difference between FightGear and Mission Planner is that mission planner cannot show the current attitude of the UAV. It can only show its position. Furthermore, its refresh rate is low. FlightGear, on the other hand, can show UAV's attitude in real-time. This is good to check UAV's condition and its attitude and its position with respect to the fire.

In order to communicate with FlightGear, a local User Datagram Protocol (UDP) connection is made with the help of "socket" library in Python. Six parameters are sent to FlightGear with a frequency of 50Hz. The following function is used to automatically send data to FlightGear.

```python
def send_to_FG():
    p = vehicle.attitude.pitch*180/3.14
    h = vehicle.attitude.yaw*180/3.14
    alt = 50 + vehicle.location.global_relative_frame.alt
    lat = vehicle.location.global_relative_frame.lat
    lon = vehicle.location.global_relative_frame.lon
    r = vehicle.attitude.roll*180/3.14
    my_message = "%.2f,%.2f,%i,%f,%f,%i\n" %(r,p,h,lat,lon,alt)
    byte_message = bytes(my_message,"utf-8")
    opened_socket.sendto(byte_message, ("192.168.2.12", 5006))
    threading.Timer(0.02,send_to_FG).start()
```

Figure 5.2: A view of FlightGear

5.1.5 How it works

In order to run the simulation, it is needed to write a script that simulates the action of cameras. First, a mission is uploaded to the SITL from Mission Planner application. Then, the script starts by taking off and following the mission. An imaginary fire location is put into the script. There are some functions in the script that simulate fire searching process. If at any location the fire point is within the field of view, the program will signal that a fire is detected and it calculates and sends the location of fire.

"CameraVeiw" function

The first step in simulating the camera is to find out the area that it is looking at. The location of UAV is available from its GPS, and it is assumed that the vehicle's roll and pitch is negligible. The FOV of thermal camera is $55° \times 35°$. In the cameraVeiw function, the area that the UAV is looking at, is defined by four points (P1 to P4 in Fig. 5.3) that are at the edge of the area. Next, in order to check if the fire location is within this area, it will check if the fire location is on which side of the lines of the area. If, it is on the right side of all them or on the left side of all of them, this means that the fire location is inside the area. In other words, it is detected. The function "leftOrRight" will check this.

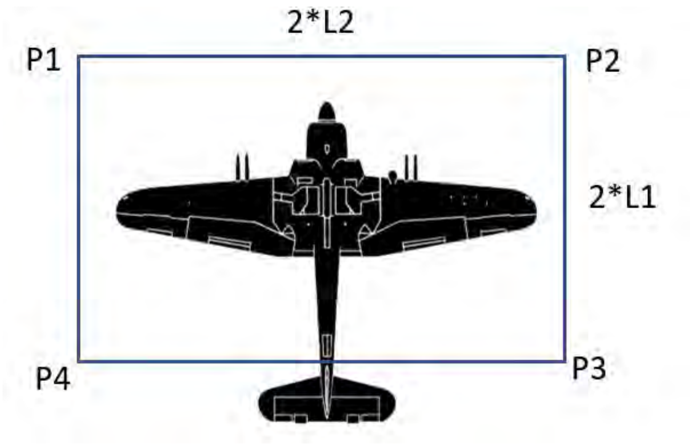


Figure 5.3: Top view with P1 to P4 points

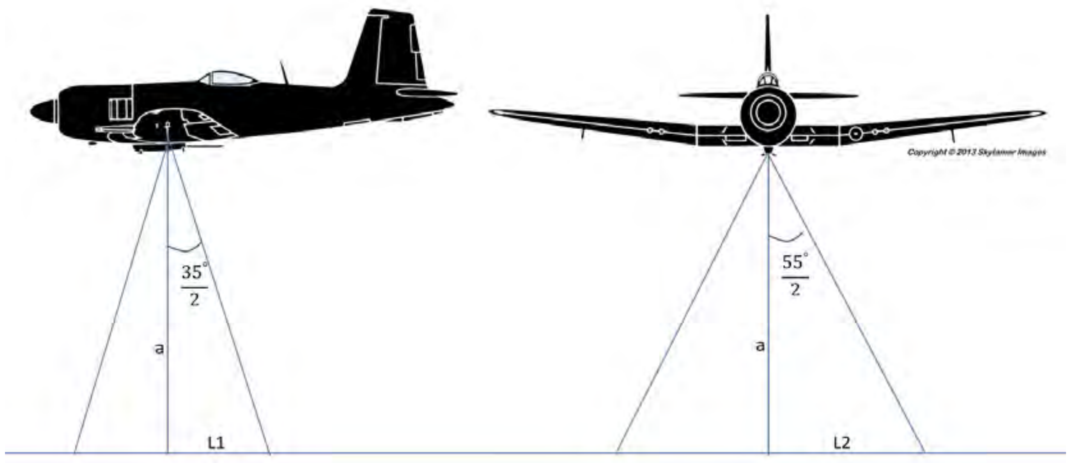


Figure 5.4: Field of view and points on earth

“fireAngle” and “findFireLoc” functions

After checking if the fire is inside the field of view, it is needed to find the location of fire. In order to do so, first, the α_x and α_y angles of fire should be calculated and then the location of fire can be easily found. If the real cameras were used, the function would find these angles based on the location of fire in the image. But here, it should be simulated from the location of fire. It is assumed that the horizontal distance between fire and camera is known as “$dist$”. Also the vertical distance is equal to the relative altitude of the UAV “a”. The GPS of UAV will provide a for the simulation.

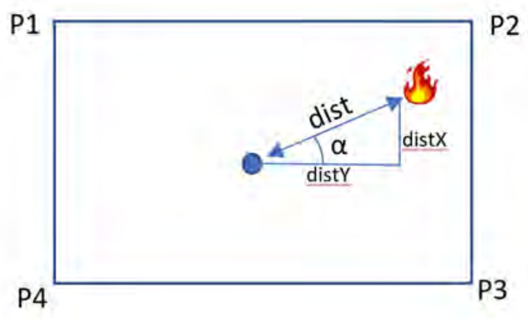


Figure 5.5: Fire and related distances

$$
\begin{aligned}
distY &= dist \times cos(\alpha) \\
distX &= dist \times sin(\alpha) \\
\alpha_y &= tan^{-1}(\frac{distY}{a}) \\
\alpha_x &= tan^{-1}(\frac{distX}{a})
\end{aligned}
\tag{5.1}
$$

After finding the α_x and α_y angles, the simulation of camera is finished and it is time to calculate the location of fire. This is done in "findFireLoc" function which does the reverse steps of "fireAngle" function . This function can also be used in real flight. That is, in the real flight, fire angles are calculated based on the real image of fire and then α_x and α_y angles are fed in this function to find out the exact location of fire. Fig. 5.6, shows the algorithm of co-simulation system and Fig. 5.7, demonstrates the flowchart of this system. To verify the performance of the co-simulation system, two scenarios are made. One is a mission without any fire and the other one is a mission in which it detects the fire. When a fire is detected, the UAV alarms the base and cancels the rest of the mission and flies over the fire. Fig. 5.8, shows the 3-D graph of both missions.

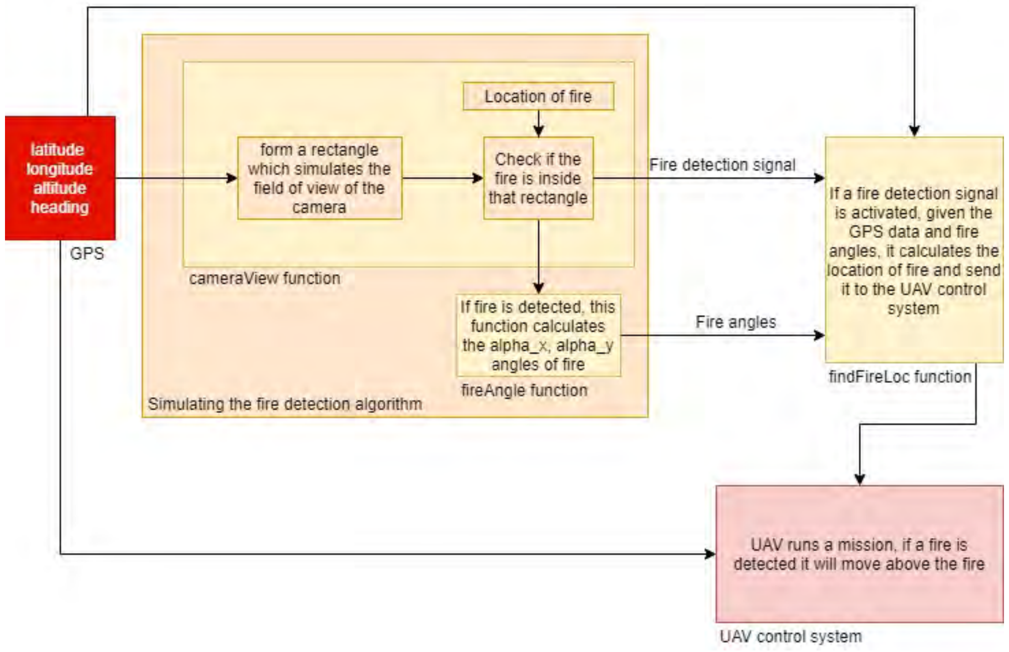

latitude
longitude
altitude
heading
GPS
form a rectangle which simulates the field of view of the camera
Location of fire
Check if the fire is inside that rectangle
cameraView function
If fire is detected, this function calculates the alpha_x, alpha_y angles of fire
fireAngle function
Simulating the fire detection algorithm
Fire detection signal
Fire angles
If a fire detection signal is activated, given the GPS data and fire angles, it calculates the location of fire and send it to the UAV control system
findFireLoc function
UAV runs a mission, if a fire is detected it will move above the fire
UAV control system

Figure 5.6: Algorithm used in the co-simulation system

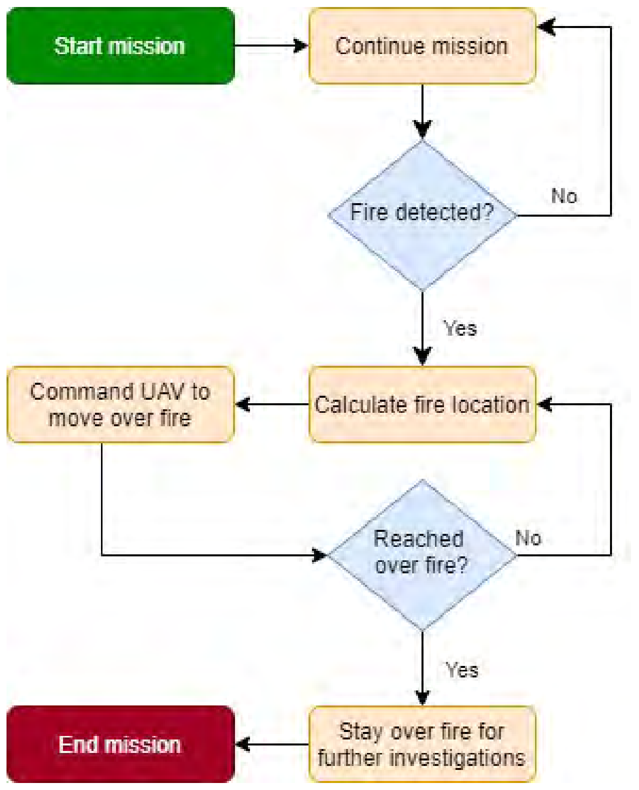


Figure 5.7: Flowchart of the co-simulation system

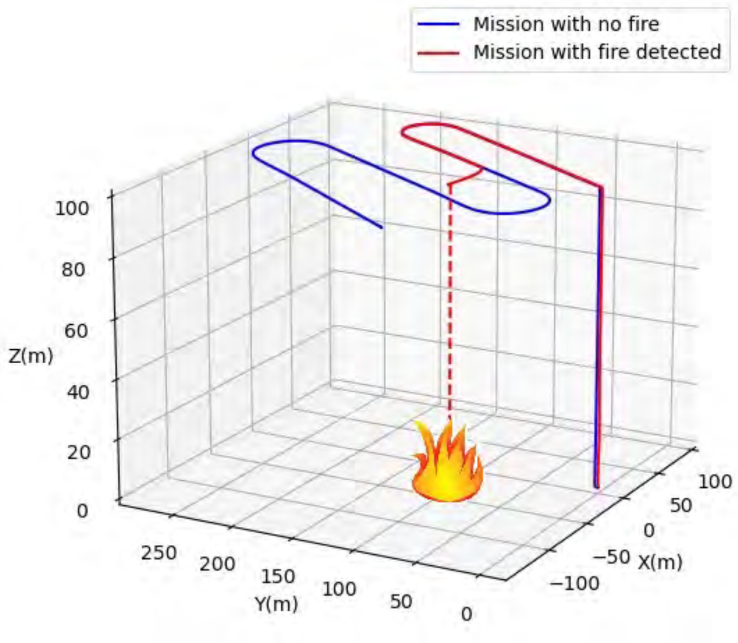


Figure 5.8: Fire detection system

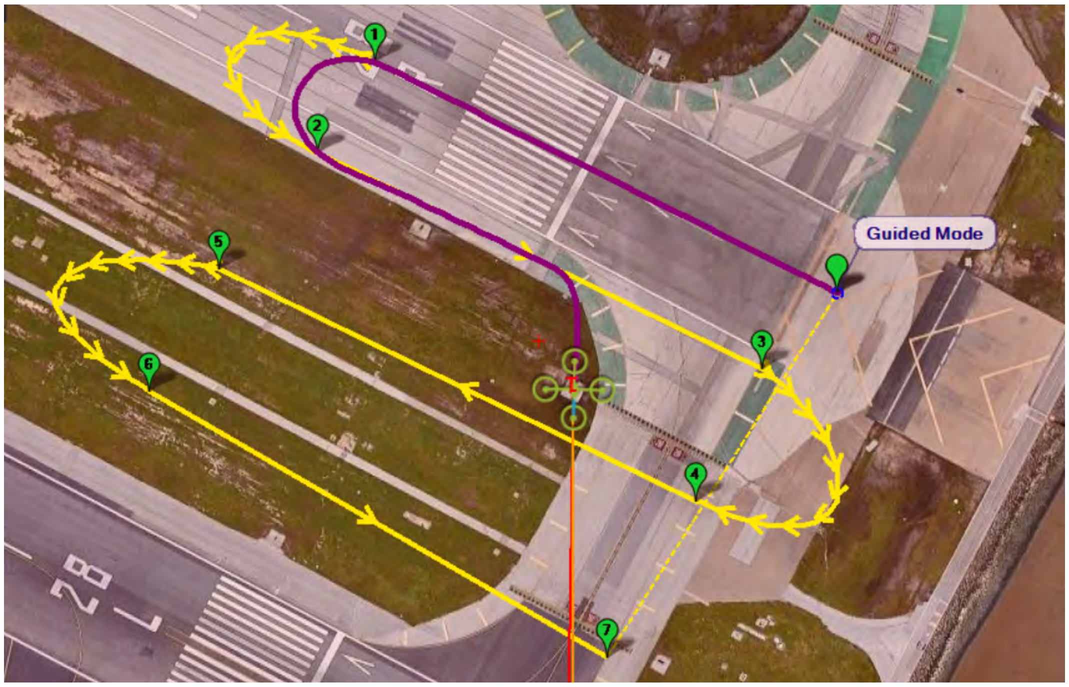


Figure 5.9: A view of Mission Planner in the fire detection operation

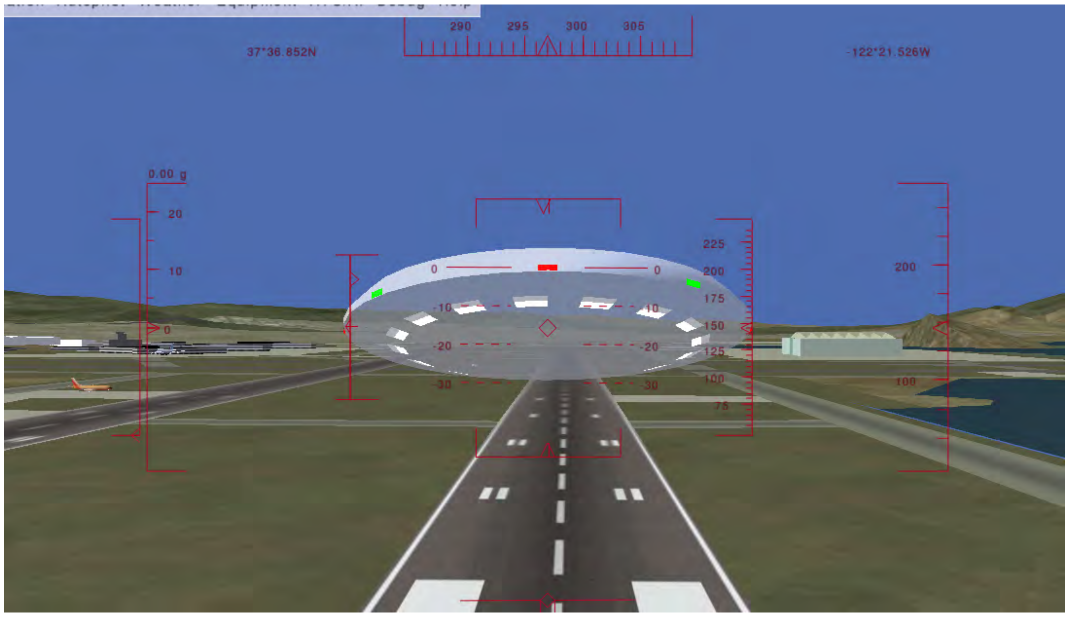


Figure 5.10: A view of FlightGear in the fire detection operation

5.2 Test results of 2DOF frame

Here, the results of a tracking test which is done with the 2DOF frame, is presented. In this test, a candle which is representing a fire, is used. Lab color space is used to detect the fire in the visual images and a temperature threshold is applied to the thermal images. The system is able to track the flame of candle and locate its location in terms of angles.

Fig. 5.11, illustrates the angles of camera and the fire. Fig. 5.12, shows the errors of fire in terms of pixels. The reason that Fig. 5.11 is represented in terms of angle (deg) and Fig. 5.12 is represented in terms of pixel is since the location of fire should be expressed in the global coordinate system. Because it should be reported to an external observer. But the error should be represented in the camera coordinate system because it is used to generate the control signal in that coordinate system.

Limitations of the system

There are some limitations to using this system in the real world. The thermal camera has a resolution of $32^\circ \times 24^\circ$. If this camera is mounted on a UAV and it is assumed that a fire has an area of at least 1 square-meter, and this fire should occupy at least 1 pixel of the image, then the maximum distance can be calculated as:

$$d \times \frac{\pi}{180} \times \frac{55}{32} = 1(m) \Longrightarrow d = 33.4(m) \tag{5.2}$$

Therefore, the maximum distance of the camera to the fire is 33.4 meters. This value can be increased either by using a camera with higher resolution or by assuming that the fire has bigger area.

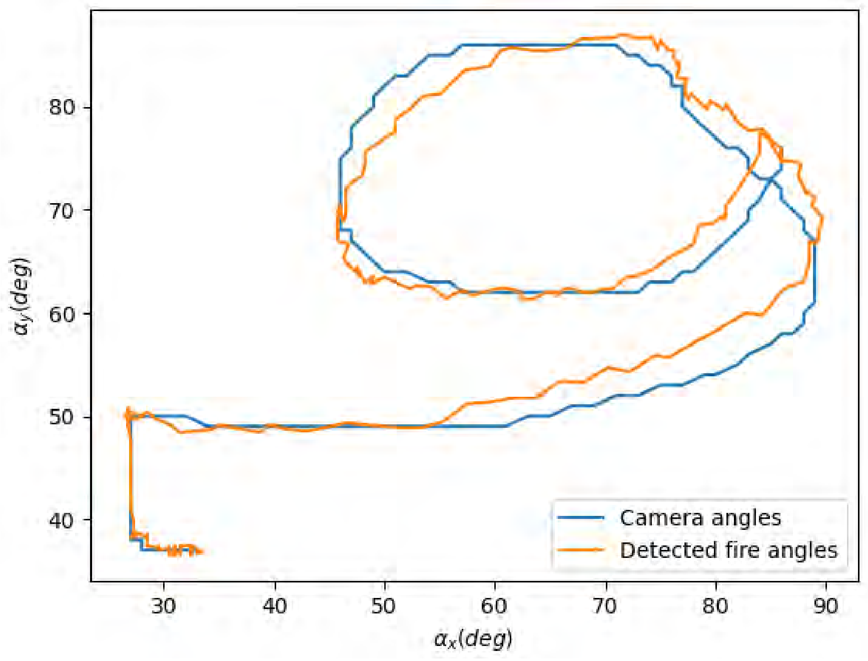


Figure 5.11: IBTVS tracking test

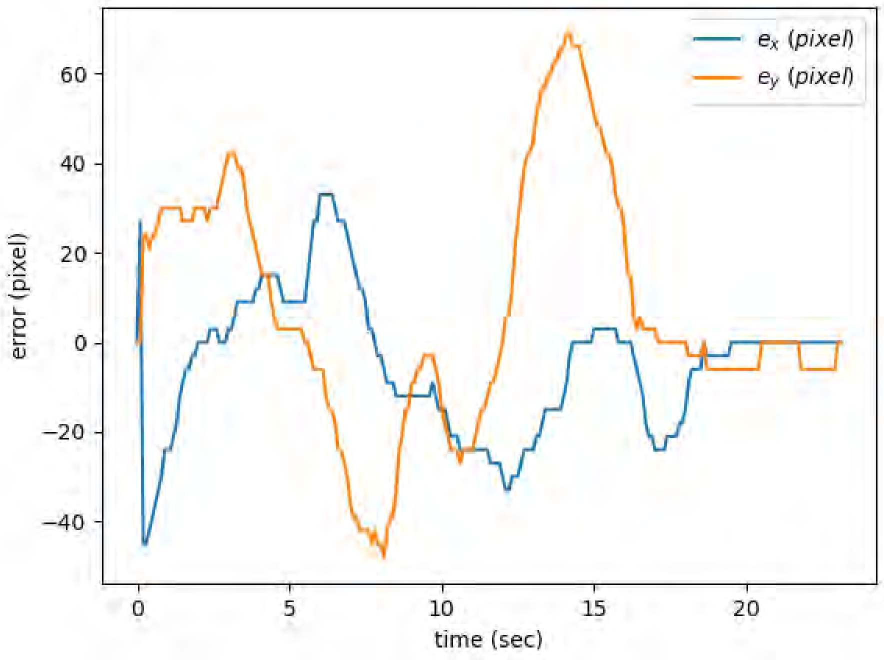


Figure 5.12: IBTVS tracking errors

5.3 Summary

In this chapter the co-simulation system is presented. The main goal of it is to simulate the behavior of fire detection system. All of the programs that are used in this simulation are described. Finally the test results of the simulation as well as 2DOF test are demonstrated. All of the videos related to this thesis are uploaded to this link.

Chapter 6

Conclusion and Future Works

In this chapter, the results of this research are discussed and then the suggested future works are presented. I hope that enthusiasts will follow this path to the point that this system is practically used in UAV to detect the forest fires.

6.1 Results

In this section, the results of this research is presented. At first, some methods for using visual camera were implemented. But, it shows that in many cases a single visual camera cannot correctly detect fire. It will give a false fire alarm. For example, if it sees an image of a fire or if it sees an object which has similar colors to a fire, then it was decided to add a thermal camera to this system. A single thermal camera is enough to detect forest fires in many cases. Also, since it has only one feature per pixel (i.e. temperature), the total data that needs to be evaluated is less than that in the visual camera. If a threshold for temperature is set, it can detect whether a fire is in the image or not. Finally, by combining both cameras, this system can have higher rate of success in detecting fire.

In order to use both cameras, a fusion method has to be proposed to make sure any pixel is the same in both images. Therefore, an alignment method is used to make sure both cameras are looking at the same pixel. The alignment method used in this research is able to align the images from both cameras. Once the images are aligned, a homography matrix is extracted to be used in further fire detection operations. After finding the fire, it is needed to locate it. In order to do so, two sets of parameters are required; the position of UAV including its latitude, longitude, altitude and heading and also the pitch and yaw angles of fires in the image. These angles can be found by knowing the field-of-view (FOV) of cameras and the location of fire in the image.

The next step in this research is planned originally to check the performance of the fire detection system on a UAV. Initially it was decided to mount the whole system in a UAV and fly it over real fire. But, due to the pandemic COVID-19 and university lockdown, it was not possible. As a result, a software-in-the-loop simulation is conducted with realistic three-dimensional (3D) visualization by using FlightGear and Mission Planner software environments. In this simulation, it is assumed that a fire occurs in one spot on the predefined path and the UAV should follow a predefined path to look for fire. Once it detects the fire, the UAV will signal the base existence of the fire. It also will calculate the location of fire and flies over it. To do so, these applications are needed to be run in two computers: DroneKit, DroneKit SITL, mavproxy on a Linux virtual machine, Mission Planner on a windows OS, and FlightGear on another windows OS. The results showed that the system is capable of finding the fire and estimating its location. The developed IBTVS can guide the UAV to fly over the detected fire for further investigations.

6.2 Future works

The following directions can be a future research path based on the works done in the current research in this thesis:

- Forest fires smokes are visible from far away distances while this is not always true for flames. This means, in order to have a system that can find forest fire in its early stages, it is necessary to look for smoke as well. Thus, it is worth investigating the combination of different fire features such as smoke and flame.

- An important parameter in consideration of forest fire is their propagation path and line of fire. The development of a method to do so can be a magnificent help to the firefighters who want to suppress the forest fires.
- Utilizing an IR camera as another source of image can be challenging if the fusion of three images is done. But, on the other hand, it can be a remarkable improvement in the fire detection process.
- In order to become operational, a system needs to pass many field tests. As indicated before, the main goal of this research was to mount the system on a UAV and test the system in more practical application scenarios but, due to the imposed conditions it did not happen. Hence, one recommended path can be conducting many field tests on a UAV. The challenging point might be the high frequency vibration of UAVs that can cause images to become blurry.

Appendix A

Hardware

A.1 Comparison of different Raspberry Pi models

There are more than ten different Raspberry Pi models. Here, a comparison table is presented for the four commonly used Raspberry Pi models.

Table A.1: Comparison between some of Raspberry Pi models

Model	**Raspberry Pi 4 B**	**Raspberry Pi 3 B+**	**Raspberry Pi Zero**	**Raspberry Pi A+**
Release date	2019 Jun 24	2018 Mar 14	2018 Jan 12	2014 Nov 10
Core Type	Cortex-A72 64-bit	Cortex-A53 64-bit	ARM1176JZF-S	ARM1176JZF-S
No. Of Cores	4	4	1	1
CPU Clock	1.5 GHz	1.4 GHz	1 GHz	700 MHz
RAM	1, 2, 4 GB DDR4	1 GB DDR2	512 MB	256 MB
USB	2x USB3+2x USB2	Yes 4x USB2.0	micro & OTG	Yes 1
HDMI port	2x micro HDMI	Yes	Yes mini	Yes
SPI	Yes	Yes	Yes	Yes
I^{2}C	Yes	Yes	Yes	Yes
GPIO	Yes	Yes 40-pins	Yes	Yes 40-pins
Camera	Yes	Yes	Yes	Yes
SD/MMC	Yes microSD	Yes microSD	Yes microSD	Yes microSD
Wi-Fi	2.4GHz and 5GHz	2.4GHz and 5GHz	2.4GHz	No
Bluetooth®	Yes 5.0	Yes 4.2, BLE	Yes 4.1	No
Height	3.37 in (85.6 mm)	3.37 in (85.6 mm)	1.18 in (30 mm)	2.55 in (65 mm)
Width	2.22 in (56.5 mm)	2.22 in (56.5 mm)	2.55 in (65 mm)	2.22 in (56.5 mm)
Depth	0.433 in (11 mm)	0.669 in (17 mm)	0.511 in (13 mm)	0.393 in (10 mm)
Weight	1.62 oz (46 g)	1.58 oz (45 g)	0.42328 oz (12 g)	0.81130 oz (23 g)
Power ratings	1.25 A @5V	1.13 A @5V	180 mA	200 mA

Appendix B

Codes

B.1 MLX90640 driver code

```
from  smbus2 import SMBus , i2c_msg
import time
import math
bus = SMBus(1)

def MLX90640_setFreq(freq):
    freq_bit = 0
    if freq in [64,32,16,8,4,2,1,0,5]:
        freq_bit = int(math.log2(freq)+1)
        MLX90640_I2CWrite(0x33,0x800D,(freq_bit<<7)|0x1801)

def I2C_read(addr,num):
    write = i2c_msg.write(0x33,[addr>>8,addr&0xFF])
    read = i2c_msg.read(0x33,num*2)
    bus.i2c_rdwr(write,read)
    data = list(read)
    i2c= []
    for value in range(len(data)):
        if value%2==0:
            b=data[value]<<8
        else:
            b=b|data[value]
            i2c=i2c+[b]
    return i2c

def MLX90640_I2CWrite(addr,reg,data):
    bus.write_i2c_block_data(addr,reg>>8,[reg&0xFF,data>>8,data&0xFF])
    #DEVICE_ADDRESS = OX33

def MLX90640_I2CRead(_deviceAddress,addr,num):
    write = i2c_msg.write(_deviceAddress,[addr>>8,addr&0xFF])
    read = i2c_msg.read(0x33,num*2)
    bus.i2c_rdwr(write,read)
    data = list(read)
    #print(data)
    i2c= []
    for value in range(len(data)):
        if value%2==0:
            b=data[value]<<8
        else:
            b=b|data[value]
            i2c=i2c+[b]
    return i2c

def MLX90640_DumpEE(slaveAddr):
     return MLX90640_I2CRead(slaveAddr, 0x2400, 832);

def MLX90640_GetFrameData(slaveAddr,oldStatus):
    dataReady=0
    while dataReady==0 :
        statusRegister = MLX90640_I2CRead(slaveAddr,0x8000,1)
        dataReady = (statusRegister[0]) & 0x0008
        dataReady = 1  # should remove this but increase freq.
        status    = (statusRegister[0]) & 0x0001
      # print("hi")
    #print("%7.3f" %(time.time()),end='\n')
    MLX90640_I2CWrite(slaveAddr,0x8000,0x30)

    frameData = MLX90640_I2CRead(slaveAddr,0x400,832)

    controlRegister1=MLX90640_I2CRead(slaveAddr,0x800D,1)
    frameData += [controlRegister1[0]];
    frameData += [status]
    return frameData

def ExtractVDDParameters(eeData):
    kVdd=(eeData[51] & 0xFF00) >> 8
    if kVdd>127:
        kVdd -= 256
    kVdd *= 32
    vdd25 = eeData[51] & 0x00FF
    vdd25 = ((vdd25-256)<<5)-8192
    return kVdd,vdd25

def ExtractPTATParameters(eeData):
    KvPTAT = (eeData[50] & 0xFC00) >> 10;
    if(KvPTAT > 31):
        KvPTAT = KvPTAT - 64;

    KvPTAT = KvPTAT/4096;

    KtPTAT = eeData[50] & 0x03FF;
    if(KtPTAT > 511):
        KtPTAT = KtPTAT - 1024;

    KtPTAT = KtPTAT/8;

    vPTAT25 = eeData[49];

    alphaPTAT = (eeData[16] & 0xF000) / pow(2, 14) + 8.0

    return KvPTAT,KtPTAT,vPTAT25,alphaPTAT

def ExtractGainParameters(eeData):
    gainEE = eeData[48];
    if(gainEE > 32767):
        gainEE = gainEE -65536;

    return gainEE;

def ExtractTgcParameters(eeData):

    tgc = eeData[60] & 0x00FF;
    if(tgc > 127):
        tgc = tgc - 256;

    tgc = tgc / 32.0;

    return tgc;

```

```
def ExtractResolutionParameters(eeData):
    resolutionEE = (eeData[56] & 0x3000) >> 12;

    return resolutionEE;

def ExtractKsTaParameters(eeData):
    KsTa = (eeData[60] & 0xFF00) >> 8;
    if(KsTa > 127):
        KsTa = KsTa -256;

    KsTa = KsTa / 8192.0;

    return KsTa;

def ExtractKsToParameters(eeData):
    step = ((eeData[63] & 0x3000) >> 12) * 10;
    ct=list(range(4))
    ksTo=ct
    ct[0] = -40;
    ct[1] = 0;
    ct[2] = (eeData[63] & 0x00F0) >> 4;
    ct[3] = (eeData[63] & 0x0F00) >> 8;

    ct[2] = ct[2]*step;
    ct[3] = ct[2] + ct[3]*step;

    KsToScale = (eeData[63] & 0x000F) + 8;
    KsToScale = 1 << KsToScale;

    ksTo[0] = eeData[61] & 0x00FF;
    ksTo[1] = (eeData[61] & 0xFF00) >> 8;
    ksTo[2] = eeData[62] & 0x00FF;
    ksTo[3] = (eeData[62] & 0xFF00) >> 8;

    for i in range(4):
        if(ksTo[i] > 127):
            ksTo[i] = ksTo[i] -256;
        ksTo[i] = ksTo[i] / KsToScale;

    return ct,ksTo

def ExtractAlphaParameters(eeData):
    accRow = [0] * 24
    accColumn = [0] * 32
    p = 0;
    accRemScale = eeData[32] & 0x000F;
    accColumnScale = (eeData[32] & 0x00F0) >> 4;
    accRowScale = (eeData[32] & 0x0F00) >> 8;
    alphaScale = ((eeData[32] & 0xF000) >> 12) + 30;
    alphaRef = eeData[33];

    for i in range(6):
        p = i * 4;
        accRow[p + 0] = (eeData[34 + i] & 0x000F);
        accRow[p + 1] = (eeData[34 + i] & 0x00F0) >> 4;
        accRow[p + 2] = (eeData[34 + i] & 0x0F00) >> 8;
        accRow[p + 3] = (eeData[34 + i] & 0xF000) >> 12;


    for i in range(24):
        if (accRow[i] > 7):
            accRow[i] = accRow[i] - 16;

    for i in range(8):
        p = i * 4;
        accColumn[p + 0] = (eeData[40 + i] & 0x000F);
        accColumn[p + 1] = (eeData[40 + i] & 0x00F0) >> 4;
        accColumn[p + 2] = (eeData[40 + i] & 0x0F00) >> 8;
        accColumn[p + 3] = (eeData[40 + i] & 0xF000) >> 12;

    for i in range(32):
        if (accColumn[i] > 7):
            accColumn[i] = accColumn[i] - 16;

    alpha = [0]*768
    for i in range(24):
        for j in range(32):
            p = 32 * i +j;
            alpha[p] = (eeData[64 + p] & 0x03F0) >> 4;
            if alpha[p] > 31:
                alpha[p] = alpha[p] - 64;

            alpha[p] = alpha[p]*(1 << accRemScale);
            alpha[p] = (alphaRef + (accRow[i] << accRowScale) + (accColumn[j]
   << accColumnScale) + alpha[p]);
            alpha[p] = alpha[p] / pow(2,alphaScale);
    return alpha

def ExtractOffsetParameters(eeData):
    occRow = [0]*24
    occColumn = [0]*32
    p = 0;
    occRemScale = (eeData[16] & 0x000F);
    occColumnScale = (eeData[16] & 0x00F0) >> 4;
    occRowScale = (eeData[16] & 0x0F00) >> 8;
    offsetRef = eeData[17];
    if (offsetRef > 32767):
        offsetRef = offsetRef - 65536;
    for i in range(6):
        p = i * 4;
        occRow[p + 0] = (eeData[18 + i] & 0x000F);
        occRow[p + 1] = (eeData[18 + i] & 0x00F0) >> 4;
        occRow[p + 2] = (eeData[18 + i] & 0x0F00) >> 8;
        occRow[p + 3] = (eeData[18 + i] & 0xF000) >> 12;

    for i in range(24):
        if (occRow[i] > 7):
            occRow[i] = occRow[i] - 16;

    for i in range(8):
        p = i * 4;
        occColumn[p + 0] = (eeData[24 + i] & 0x000F);
        occColumn[p + 1] = (eeData[24 + i] & 0x00F0) >> 4;
        occColumn[p + 2] = (eeData[24 + i] & 0x0F00) >> 8;
        occColumn[p + 3] = (eeData[24 + i] & 0xF000) >> 12;

    for i in range(32):
        if (occColumn[i] > 7):
            occColumn[i] = occColumn[i] - 16;

    offset = [0]*768
    for i in range(24):
        for j in range(32):
            p = 32 * i +j;
            offset[p] = (eeData[64 + p] & 0xFC00) >> 10;
            if (offset[p] > 31):
                offset[p] = offset[p] - 64;

            offset[p] = offset[p]*(1 << occRemScale);
            offset[p] = (offsetRef + (occRow[i] << occRowScale) + (occColumn[j
    ] << occColumnScale) + offset[p]);

    return offset

def ExtractKtaPixelParameters(eeData):
    p = 0;
    KtaRC = [0] * 4;
    KtaRoCo = (eeData[54] & 0xFF00) >> 8;
    if (KtaRoCo > 127):
        KtaRoCo = KtaRoCo - 256;

    KtaRC[0] = KtaRoCo;

    KtaReCo = (eeData[54] & 0x00FF);
    if (KtaReCo > 127):
        KtaReCo = KtaReCo - 256;

    KtaRC[2] = KtaReCo;

    KtaRoCe = (eeData[55] & 0xFF00) >> 8;
    if (KtaRoCe > 127):
        KtaRoCe = KtaRoCe - 256;

    KtaRC[1] = KtaRoCe;

```

```python
    KtaReCe = (eeData[55] & 0x00FF);
    if (KtaReCe > 127):
        KtaReCe = KtaReCe - 256;

    KtaRC[3] = KtaReCe;

    ktaScale1 = ((eeData[56] & 0x00F0) >> 4) + 8;
    ktaScale2 = (eeData[56] & 0x000F);

    kta = [0] * 768
    for i in range(24):
        for j in range(32):
            p = 32 * i +j;
            split = int(2*(p/32 - (p/64)*2) + p%2)
            kta[p] = (eeData[64 + p] & 0x000E) >> 1;
            if (kta[p] > 3):
                kta[p] = kta[p] - 8;

            kta[p] = kta[p] * (1 << ktaScale2);
            kta[p] = KtaRC[split] + kta[p];
            kta[p] = kta[p] / pow(2,ktaScale1);

    return kta

def ExtractKvPixelParameters(eeData):
    p = 0;
    KvT = [0] * 4
    KvRoCo = (eeData[52] & 0xF000) >> 12;
    if (KvRoCo > 7):
        KvRoCo = KvRoCo - 16;

    KvT[0] = KvRoCo;

    KvReCo = (eeData[52] & 0x0F00) >> 8;
    if (KvReCo > 7):
        KvReCo = KvReCo - 16;

    KvT[2] = KvReCo;

    KvRoCe = (eeData[52] & 0x00F0) >> 4;
    if (KvRoCe > 7):
        KvRoCe = KvRoCe - 16;

    KvT[1] = KvRoCe;

    KvReCe = (eeData[52] & 0x000F);
    if (KvReCe > 7):
        KvReCe = KvReCe - 16;

    KvT[3] = KvReCe;

    kvScale = (eeData[56] & 0x0F00) >> 8;

    kv = [0] * 768
    for i in range(24):
        for j in range(32):
            p = 32 * i +j;
            split = int(2*(p/32 - (p/64)*2) + p%2)
            kv[p] = KvT[split];
            kv[p] = kv[p] / pow(2,kvScale);
    return kv;

```

```python
def ExtractCPParameters(eeData):
    alphaSP = [0] * 2 ;
    offsetSP = [0] * 2;
    cpAlpha = [0] * 2
    cpOffset = [0] * 2
    alphaScale = ((eeData[32] & 0xF000) >> 12) + 27;

    offsetSP[0] = (eeData[58] & 0x03FF);
    if (offsetSP[0] > 511):
        offsetSP[0] = offsetSP[0] - 1024;


    offsetSP[1] = (eeData[58] & 0xFC00) >> 10;
    if (offsetSP[1] > 31):
        offsetSP[1] = offsetSP[1] - 64;

    offsetSP[1] = offsetSP[1] + offsetSP[0];

    alphaSP[0] = (eeData[57] & 0x03FF);
    if (alphaSP[0] > 511):
        alphaSP[0] = alphaSP[0] - 1024;

    alphaSP[0] = alphaSP[0] /  pow(2,alphaScale);

    alphaSP[1] = (eeData[57] & 0xFC00) >> 10;
    if (alphaSP[1] > 31):
        alphaSP[1] = alphaSP[1] - 64;

    alphaSP[1] = (1 + alphaSP[1]/128) * alphaSP[0];

    cpKta = (eeData[59] & 0x00FF);
    if (cpKta > 127):
        cpKta = cpKta - 256;

    ktaScale1 = ((eeData[56] & 0x00F0) >> 4) + 8;
    cpKta = cpKta / pow(2,ktaScale1);

    cpKv = (eeData[59] & 0xFF00) >> 8;
    if (cpKv > 127):
        cpKv = cpKv - 256;

    kvScale = (eeData[56] & 0x0F00) >> 8;
    cpKv = cpKv / pow(2,kvScale);

    cpAlpha[0] = alphaSP[0];
    cpAlpha[1] = alphaSP[1];
    cpOffset[0] = offsetSP[0];
    cpOffset[1] = offsetSP[1];

    return cpKta,cpKv,cpAlpha,cpOffset

def ExtractCILCParameters(eeData):
    ilChessC = [0] * 3
    calibrationModeEE = (eeData[10] & 0x0800) >> 4;
    calibrationModeEE = calibrationModeEE ^ 0x80;

    ilChessC[0] = (eeData[53] & 0x003F);
    if (ilChessC[0] > 31):
        ilChessC[0] = ilChessC[0] - 64;

    ilChessC[0] = ilChessC[0] / 16.0

    ilChessC[1] = (eeData[53] & 0x07C0) >> 6;
    if (ilChessC[1] > 15):
        ilChessC[1] = ilChessC[1] - 32;

    ilChessC[1] = ilChessC[1] / 2.0

    ilChessC[2] = (eeData[53] & 0xF800) >> 11;
    if (ilChessC[2] > 15):
        ilChessC[2] = ilChessC[2] - 32;

    ilChessC[2] = ilChessC[2] / 8.0

    return calibrationModeEE, ilChessC

def ExtractDeviatingPixels(eeData):
    pixCnt = 0;
    brokenPixCnt = 0;
    outlierPixCnt = 0;
    warn = 0;
    brokenPixels = [0] * 5
    outlierPixels = [0] * 5
    for pixCnt in range(5):
        brokenPixels[pixCnt] = 0xFFFF;
        outlierPixels[pixCnt] = 0xFFFF;



    pixCnt = 0;
    while (pixCnt < 768 & brokenPixCnt < 5 & outlierPixCnt < 5):
        if(eeData[pixCnt+64] == 0):
            brokenPixels[brokenPixCnt] = pixCnt;
            brokenPixCnt = brokenPixCnt + 1;

        elif((eeData[pixCnt+64] & 0x0001) != 0):
            outlierPixels[outlierPixCnt] = pixCnt;
            outlierPixCnt = outlierPixCnt + 1;
         pixCnt = pixCnt + 1;

    if(brokenPixCnt > 4):
        warn = -3;

    elif(outlierPixCnt > 4) :
        warn = -4;

    elif((brokenPixCnt + outlierPixCnt) > 4):
        warn = -5;

    else:
        for pixCnt in range(brokenPixCnt):
            for i in range(brokenPixCnt):
                warn = CheckAdjacentPixels(brokenPixels[pixCnt],brokenPixels[i
   ]);
                if(warn != 0):
                    return warn;

        for pixCnt in range(outlierPixCnt):
            for i in range(outlierPixCnt):
                warn = CheckAdjacentPixels(outlierPixels[pixCnt],outlierPixels
   [i]);
                if(warn != 0):
                    return warn;

        for pixCnt in range(brokenPixCnt):
            for i in range(outlierPixCnt):
                warn = CheckAdjacentPixels(brokenPixels[pixCnt],outlierPixels[i]);
                if(warn != 0):
                    return warn;

    return warn,brokenPixels,outlierPixels;

def CheckEEPROMValid(eeData):
     deviceSelect = eeData[10] & 0x0040;
     if(deviceSelect == 0):
         return 0;

     return -7;

def MLX90640_ExtractParameters(eeData):
    error = CheckEEPROMValid(eeData);

    if(error == 0):
        kVdd,vdd25 = ExtractVDDParameters(eeData);
        KvPTAT,KtPTAT,vPTAT25,alphaPTAT = ExtractPTATParameters(eeData );
        gainEE = ExtractGainParameters(eeData );
        tgc = ExtractTgcParameters(eeData );
        resolutionEE = ExtractResolutionParameters(eeData);
        KsTa = ExtractKsTaParameters(eeData );
        ct,ksTo = ExtractKsToParameters(eeData);
        alpha = ExtractAlphaParameters(eeData );
        offset = ExtractOffsetParameters(eeData );
        kta = ExtractKtaPixelParameters(eeData );
        kv = ExtractKvPixelParameters(eeData );
        cpKta,cpKv,cpAlpha,cpOffset = ExtractCPParameters(eeData );
        calibrationModeEE, ilChessC = ExtractCILCParameters(eeData );
        error,brokenPixels,outlierPixels = ExtractDeviatingPixels(eeData );
    mlx90640={}
    mlx90640['kVdd']=kVdd
    mlx90640['vdd25']=vdd25
    mlx90640['KvPTAT']=KvPTAT
    mlx90640['KtPTAT']=KtPTAT
    mlx90640['vPTAT25']=vPTAT25
    mlx90640['alphaPTAT']=alphaPTAT
    mlx90640['gainEE']=gainEE
    mlx90640['tgc']=tgc
    mlx90640['resolutionEE']=resolutionEE
    mlx90640['KsTa']=KsTa
    mlx90640['ct']=ct
    mlx90640['ksTo']=ksTo
    mlx90640['alpha']=alpha
    mlx90640['offset']=offset
    mlx90640['kta']=kta
    mlx90640['kv']=kv
    mlx90640['cpKta']=cpKta
    mlx90640['cpKv']=cpKv
    mlx90640['cpAlpha']=cpAlpha
    mlx90640['cpOffset']=cpOffset
    mlx90640['calibrationModeEE']=calibrationModeEE
    mlx90640['ilChessC']=ilChessC
    mlx90640['brokenPixels']=brokenPixels
    mlx90640['outlierPixels']=outlierPixels

    return error, mlx90640

def MLX90640_GetVdd(frameData, params):
    vdd = frameData[810];
    if(vdd > 32767):
        vdd = vdd - 65536;

    resolutionRAM = (frameData[832] & 0x0C00) >> 10;
    resolutionCorrection = pow(2, params['resolutionEE']) / pow(2,
   resolutionRAM);
    vdd = (resolutionCorrection * vdd - params['vdd25']) / params['kVdd'] +
   3.3;

    return vdd;

def MLX90640_GetTa(frameData,params):
    vdd = MLX90640_GetVdd(frameData, params);

    ptat = frameData[800];
    if(ptat > 32767):
        ptat = ptat - 65536;

    ptatArt = frameData[768];
    if(ptatArt > 32767):
        ptatArt = ptatArt - 65536;

    ptatArt = (ptat / (ptat * params['alphaPTAT'] + ptatArt)) * pow(2, 18);

    ta = (ptatArt / (1 + params['KvPTAT'] * (vdd - 3.3)) - params['vPTAT25']);
    ta = ta / params['KtPTAT'] + 25;

    return ta;

def MLX90640_CalculateTo(frameData, params, emissivity,  tr,result):
    irDataCP = [0] * 2;
    alphaCorrR = [0] * 4;
    #result = [0]*768
    subPage = frameData[833];
    vdd = MLX90640_GetVdd(frameData, params);
    ta = MLX90640_GetTa(frameData, params);
    ta4 = pow((ta + 273.15), 4);
    tr4 = pow((tr + 273.15), 4);
    taTr = tr4 - (tr4-ta4)/emissivity;

    alphaCorrR[0] = 1 / (1 + params['ksTo'][0] * 40);
    alphaCorrR[1] = 1 ;
    alphaCorrR[2] = (1 + params['ksTo'][2] * params['ct'][2]);
    alphaCorrR[3] = alphaCorrR[2] * (1 + params['ksTo'][3] * (params['ct'][3]
   - params['ct'][2]));

#------------------------- Gain calculation
   -----------------------------------

    gain = frameData[778];
    if(gain > 32767):
        gain = gain - 65536;


    gain = params['gainEE'] / gain;

#------------------------- To calculation
   -------------------------------------
    mode = (frameData[832] & 0x1000) >> 5;

    irDataCP[0] = frameData[776];
    irDataCP[1] = frameData[808];
    for i in [0,1]:
        if(irDataCP[i] > 32767):
            irDataCP[i] = irDataCP[i] - 65536;

        irDataCP[i] = irDataCP[i] * gain;

    irDataCP[0] = irDataCP[0] - params['cpOffset'][0] * (1 + params['cpKta'] *
     (ta - 25)) * (1 + params['cpKv'] * (vdd - 3.3));
    if( mode ==  params['calibrationModeEE']):
        irDataCP[1] = irDataCP[1] - params['cpOffset'][1] * (1 + params['cpKta
   '] * (ta - 25)) * (1 + params['cpKv'] * (vdd - 3.3));

    else:
        irDataCP[1] = irDataCP[1] - (params['cpOffset'][1] + params['ilChessC'
   ][0]) * (1 + params['cpKta'] * (ta - 25)) * (1 + params['cpKv'] * (vdd -
   3.3));


    for pixelNumber in range(768):
        ilPattern = int(pixelNumber / 32) - int(pixelNumber / 64) * 2;
        chessPattern = int(ilPattern) ^ int(pixelNumber - int(pixelNumber/2)
   *2);

        conversionPattern = (int((pixelNumber + 2) / 4) - int((pixelNumber +
   3) / 4) + int((pixelNumber + 1) / 4) - int(pixelNumber / 4)) * (1 - 2 *
   ilPattern);

        if(mode == 0):
            pattern = ilPattern;

        else :
            pattern = chessPattern;


        if(pattern == frameData[833]):

            irData = frameData[pixelNumber];

            if(irData > 32767):
                irData = irData - 65536;

            irData = irData * gain;

            irData = irData - params['offset'][pixelNumber]*(1 + params['kta'
    ][pixelNumber]*(ta - 25))*(1 + params['kv'][pixelNumber]*(vdd - 3.3));
            if(mode !=  params['calibrationModeEE']):
                irData = irData + params['ilChessC'][2] * (2 * ilPattern - 1)
    - params['ilChessC'][1] * conversionPattern;


            irData = irData / emissivity;

            irData = irData - params['tgc'] * irDataCP[subPage];

            alphaCompensated = (params['alpha'][pixelNumber] - params['tgc'] *
     params['cpAlpha'][subPage])*(1 + params['KsTa'] * (ta - 25));

            Sx = pow(alphaCompensated, 3) * (irData + alphaCompensated * taTr)
    ;
            Sx = math.sqrt(math.sqrt(abs(Sx))) * params['ksTo'][1];

            To = math.sqrt(math.sqrt(irData/(alphaCompensated * (1 - params['ksTo'][1] * 273.15) + Sx) + taTr)) - 273.15;

            if(To < params['ct'][1]):
                range1 = 0;

            elif(To < params['ct'][2]):
                range1 = 1;

            elif(To < params['ct'][3]):
                range1 = 2;

            else:
                range1 = 3;


            To = math.sqrt(math.sqrt(irData / (alphaCompensated * alphaCorrR[range1] * (1 + params['ksTo'][range1] * (To - params['ct'][range1]))) + taTr)) - 273.15;
            #print(To)
            result[pixelNumber] = To;
```

B.2 Thermal camera fire detection code

```
from MLX90640_I2C_Driver import *
from colorsys import *
import tkinter as tk
import time,threading
#import sched
from random import *
from math import *
```

```
import numpy as np
from PIL import Image, ImageTk
import cv2
import serial
from matplotlib import pyplot as plt
serIs = False   #Is serial connected?
#from skimage.transform import resize
if serIs == True:
    ser = serial.Serial('/dev/ttyS0',baudrate = 38400)
    if ser.is_open==False:
        ser.open()
window = tk.Tk()
canvas = tk.Canvas(window, bg="white", width= 800, height = 600)
canvas.pack()
mlx90640To = [0]*768
x=0
status=0
oldStatus=1
ii=1
eeData = MLX90640_DumpEE(0x33)
error,mlx90640 = MLX90640_ExtractParameters(eeData)
#s = sched.scheduler(time.time,time.sleep)
ang1 = 0
ang2 = 0
def sendSerial():
    if serIs == True:
        if abs(ang1) > 2 or abs(ang2) > 2:
            ser.write((str(int(ang1))+" "+str(int(ang2))+'\n').encode('utf-8')
    )
        else:
            ser.write(b'0\n')
        threading.Timer(0.1,sendSerial).start()

def create_spectrum():
    f=open("thermal_graph.txt")
    a=f.read()
    f.close()
    d=a.splitlines()
    l= []
    for x in range(len(d)):
        temp=d[len(d)-1-x].split()
        l+=[[int(temp[0]),int(temp[1]),int(temp[2])]]
    return l

def frame():
    color = np.zeros((24,32,3),dtype=np.uint8)
    return color

def interpolate_color(temp,min1,max1,l):
    temp = map(temp,min1,max1,0,len(l)-1)
    if type(temp)==int:
        f=l[temp]
        color=f'#{int(f[0]):02x}{int(f[1]):02x}{int(f[2]):02x}'
        return color
    elif type(temp)==list:
        color=[0]*len(temp)

        for i in range(len(temp)):
            f=l[temp[i]]
            color[i]=f'#{int(f[0]):02x}{int(f[1]):02x}{int(f[2]):02x}'

        return color
```

```

def interpolate_color2(temp,min1,max1,l):
    #if type(temp)==np.ndarray:
    color= np.zeros((24,32,3),dtype=np.uint8)
    temp = map(temp,min1,max1,0,len(l)-1)
    if type(temp)==int:
        f=l[temp]
        color=f
        return color
    else: # type(temp)==np.ndarray:

        for i in range(24):
            for j in range(32):
                f=l[temp[i*32+j]]
                color[i][j]=f
        #print(color)
        return color

def draw_spectrum():
    global l
    width=20
    height=120
    size=2
    min1=0
    max1=height
    color=[]
    for i in range(height):
        color+=[interpolate_color(i,min1,max1,l)]

    for i in range(height):
        aa=canvas.create_rectangle(340,i*size,340+width,size*i+size,fill=color
    [height-1-i],width = 0)
        del aa
    return color

def random_line(event):
    x1=randint(0,300)
    x2=randint(0,300)
    y1=randint(0,300)
    y2=randint(0,300)
    canvas.create_rectangle(x1,x2,y1,y2,fill=f'#{randint(0,0xffffff):06x}',
    width=20)

def map(x, in_min, in_max, out_min, out_max):
    if type(x)== int:
        return int((x-in_min)*(out_max - out_min) / (in_max - in_min ) +
    out_min)
    elif type(x)==list:
        y=[0]*len(x)
        coef= (out_max - out_min) / (in_max - in_min )
        for i in range(len(x)):
            y[i]=int((x[i]-in_min)* coef + out_min)
        return y

def thermal_color(x):
    global l
    j=interpolate_color2(x,int(min(x)),int(max(x)),l)
    return j

def delete_lines(event):
    canvas.delete('all')
    l=3,2,8,7
    canvas.create_oval(l)

def draw_thermal(To):
    global tt
    #print("%7.3f" %(time.time()-tt),end=' ')
    To=map(To,min(To),max(To),min(To),max(To))
    #print("%7.3f" %(time.time()-tt),end=' ')
    jj=thermal_color(To)
    #print("%7.3f" %(time.time()-tt),end=' ')
    jj_2 = cv2.resize(jj,(320,240))
    return jj_2

def findCenter(contour):
    i=0
    cX = 160
    cY = 120
    max = 0
    for c in contour:
        area = cv2.contourArea(c)
        if area < max:
            continue
        else:
            max = area
            M = cv2.moments(c)

            cX = int(M["m10"] / M["m00"])
            cY = int(M["m01"] / M["m00"])

    return cX,cY

```

```python
def find_angle(x,y):
        a=160/tan(55/360*3.1415)
        b=120/tan(35/360*3.1415)
        angle1 = -atan((160-x)/a)*180/3.1415
        angle2 = atan((120-y)/b)*180/3.1415
        #print("%5.1f %5.1f" %(angle1,angle2),end="\n")
        return angle1,angle2

def calc_To():
    global mlx90640To
    global mlx90640
    global x
    global  status
    global oldStatus

    mlx90640Frame = MLX90640_GetFrameData(0x33,oldStatus)
    status=mlx90640Frame[833]
    if status == oldStatus:
        mlx90640Frame[833]=1-status
    oldStatus=mlx90640Frame[833]
    vdd = MLX90640_GetVdd(mlx90640Frame,mlx90640)
    Ta = MLX90640_GetTa(mlx90640Frame,mlx90640)
    TA_SHIFT=8
    tr=Ta-TA_SHIFT
    emissivity=0.95
    MLX90640_CalculateTo(mlx90640Frame,mlx90640,emissivity,tr,mlx90640To)
    return draw_thermal(mlx90640To)

l=create_spectrum()
draw_spectrum()
cap = cv2.VideoCapture(0)
```

```python
cap.set(3,320)
cap.set(4,240)
cap.set(cv2.CAP_PROP_BUFFERSIZE,1)
tt=time.time()
now1 = tt
im = 1
sum = 0
cntr = 0
MLX90640_setFreq(16)
sendSerial()
while cap.isOpened():
    ret, frame = cap.read()
    frame = cv2.rotate(frame,cv2.ROTATE_180)
    frame = cv2.flip(frame,1)
    now2 = time.time()-tt
    #print("%7.3f" %(now1),end='\t')
    try:
        tem=calc_To()
    except:
        print("error")
        cntr+=1
        #exit(0)
    #now2 = time.time()-tt
    tem = cv2.cvtColor(tem, cv2.COLOR_RGB2BGR)
    tem = cv2.rotate(tem,cv2.ROTATE_180)
    sum = now2-now1
    now1 = now2
    if ret == True:

        frame = cv2.vconcat((tem,frame))
        cv2.imshow('Live', frame)
        kk1 = cv2.waitKey(1) & 0xFF
        if kk1 == ord('s'):
            cv2.imwrite('image'+str(im)+'.jpg',frame)
            cv2.imwrite('temp_map'+str(im)+'.jpg',color2)
            cv2.imwrite('thermal_pic'+str(im)+'.jpg',tem)
            im+=1
            print('saved')

        elif kk1 == ord('q'):
            break
    else:
        break
    color = np.zeros((24,32,3),dtype=np.uint8)
    for i in range(24):
            for j in range(32):

                if mlx90640To[i*32+j] > 128:
                    color[i][j] = [255,255,255]
                else:
                    color[i][j] = [int(mlx90640To[i*32+j])*2]*3
                color[i][j] = [int(mlx90640To[i*32+j])] #should be removed
    #ze = frame()#(mlx90640To[0])
    color2 = cv2.resize(color,(320,240))
    color2 = cv2.rotate(color2,cv2.ROTATE_180)
    color2 = cv2.cvtColor(color2,cv2.COLOR_BGR2GRAY)
    _ ,th = cv2.threshold(color2,50,255,cv2.THRESH_BINARY)
    th = cv2.dilate(th,None,iterations=5)
    contour,_ = cv2.findContours(th,cv2.RETR_TREE,cv2.CHAIN_APPROX_SIMPLE)
    cX,cY = findCenter(contour)
    #print(cX,cY)
    ang1,ang2 = find_angle(cX,cY)
    print(int(ang1),end = ' ')
    print(int(ang2))

    cv2.drawContours(color2,contour,-1,(100,0,0),5)
    cv2.circle(color2,(cX,cY),5,(255,0,0),-1)
    cv2.imshow('color',color2)
    #print(len(contour))
if serIs == True:
    ser.close()
cap.release()
cv2.destroyAllWindows()
```

B.3 Arduino code for running the servos of frame

```
#include <Adafruit_PWMServoDriver.h>
#include <SoftwareSerial.h>// import the serial library
SoftwareSerial mySerial(9,10); // RX, TX
Adafruit_PWMServoDriver pwm = Adafruit_PWMServoDriver();
long int meanX = 0, meanY = 0;
long t0;

void position(int commandX = 0, int commandY = 0) {
    int x, y;

    readAnlg(x, y);
    static int roll = map(x, 0, 1023, 550, 2400);
    static int pitch = map(y, 0, 1023, 550, 2120);
    if (commandX == 0 && commandY == 0) {

        if (abs(x - meanX) > 3)
            roll -= (x - meanX) / 40;
        if (abs(y - meanY) > 3)
            pitch += (y - meanY) / 40;
    } else {
        roll -= commandX;
        pitch += commandY;
    }
    roll = constrain(roll, 550, 2400);
    pitch = constrain(pitch, 550, 2120);
    setXY(roll, pitch);
}

void setup() {
    mySerial.begin(38400);
    Serial.begin(38400);
    pwm.begin();

    pwm.setPWMFreq(50);  // Analog servos run at ~60 Hz updates

    for (int i = 0; i < 100; i++) {
        meanX += analogRead(A0);
        meanY += analogRead(A1);
    }
    meanX /= 100;
    meanY /= 100;

    Serial.print(meanX);
    Serial.print(" ");
    Serial.println(meanY);
    t0 = millis();
}

```

```
void loop() {

    char g;
    int valX = 0, valY = 0;
    while (mySerial.available()) {
        g = mySerial.peek();
        if ((g >= '0' && g <= '9') || (g == '-')) {
            valX = mySerial.parseInt();
            Serial.print(valX);
            Serial.print(" ");
            valY = mySerial.parseInt();
            Serial.println(valY);

        }
        while (mySerial.available()) {
            g = mySerial.read();
        }
    }
    position(valX, valY);
    setFrequency(50);
}

boolean setFrequency(int freq) {
    static long lastT = t0;
    int a = 0;
    while ((millis() - lastT) < (1000 / freq)) {
        a++   ;
    }
    lastT = millis();

}
```

```

void setXY(int i, int j) {
    i = constrain(i, 550, 2400);
    int k = map(i, 550, 2400, 2400, 550);
    pwm.writeMicroseconds(0, i);
    pwm.writeMicroseconds(1, k);

    j = constrain(j, 550, 2120);
    int m = map(j, 550, 2120, 2120, 550);
    pwm.writeMicroseconds(2, j);
    pwm.writeMicroseconds(3, m);
}

void readAnlg(int& x, int& y) {

    x = analogRead(A0);
    y = analogRead(A1);
}
```

B.4 Software-in-the-loop main code in Python

```python
import time,threading,math
from dronekit import connect, VehicleMode, LocationGlobalRelative
import socket
opened_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
import geopy

# Set up option parsing to get connection string
import argparse
parser = argparse.ArgumentParser(description='Commands')
parser.add_argument('--connect')
```

```python
args = parser.parse_args()
import numpy as np
from geopy.distance import VincentyDistance
from geopy.distance import geodesic
import math

connection_string = args.connect
home = (37.6135,-122.357)
FOV = (55,35)
fireLoc = geopy.Point(37.61458,-122.35539)
print('Connecting to vehicle on: %s' % connection_string)
vehicle = connect(connection_string, wait_ready=True)

def fireAngle():
    h = math.degrees(vehicle.attitude.yaw)
    alt = vehicle.location.global_relative_frame.alt
    lat = vehicle.location.global_relative_frame.lat
    lon = vehicle.location.global_relative_frame.lon
    origin = geopy.Point(lat, lon)
    b = bearing(lat,lon,fireLoc.latitude,fireLoc.longitude)
    bNew = -b+90
    if(bNew < 0): bNew += 360
    diff = abs(h-bNew)

    if diff > 270: final = 360 - diff;
    elif diff > 180: final = diff - 180
    elif diff > 90: final = 180 - diff
    else: final = diff

    print("heading = %d, fireBearing = %2.2f, relative fire heading = %2.2f"
   %(h,bNew,final))
```

```
    dist =(geodesic((lat,lon),(fireLoc.latitude,fireLoc.longitude)).m)
    distY = dist * math.cos(math.radians(final))
    distX = dist * math.sin(math.radians(final))
    xAngle = math.atan2(alt,distX)*180/3.14
    yAngle = math.atan2(alt,distY)*180/3.14
    print("distance to fire=%3.2f, distX=%3.2f, distY=%3.2f" %(dist,distX,
   distY))
    diff2 = bNew - h
    x,y = findCameraSide(diff2)
    xAngle = math.atan2(distX,alt)*180/3.14*x
    yAngle = math.atan2(distY,alt)*180/3.14*y
    print("xAngle =%3.2f, yAngle=%3.2f" %(xAngle,yAngle))
    return xAngle,yAngle

def findFireLoc(xAngle,yAngle):
    h = math.degrees(vehicle.attitude.yaw)
    alt = vehicle.location.global_relative_frame.alt
    lat = vehicle.location.global_relative_frame.lat
    lon = vehicle.location.global_relative_frame.lon
    origin = geopy.Point(lat, lon)

    fireBearing = math.atan2(xAngle, yAngle) * 180 / 3.14 + h

    #fireBearing += 90;

    if fireBearing > 360: fireBearing -=360

    distX = math.tan(math.radians(xAngle))*alt
    distY = math.tan(math.radians(yAngle))*alt
    print("calculated distX and distY= %3.2f %3.2f" %(distX,distY))
    fireDist = math.sqrt(distX**2+distY**2)
    #fireBearing = math.atan2(distY, distX) * 180 / 3.14 + h
    if fireBearing < 0: fireBearing += 360
    elif fireBearing > 360: fireBearing -= 360
    fireLocFound = VincentyDistance(meters=fireDist).destination(origin,
   fireBearing)
    print("Calculated fire bearing = %2.2f, calculated fireDist = %2.2f" %(
   fireBearing,fireDist))
    print("FireLoc = %1.3f %1.3f" %(fireLocFound.latitude,fireLocFound.
   longitude))
    return

def cameraView():

    h = math.degrees(vehicle.attitude.yaw)
    alt = vehicle.location.global_relative_frame.alt
    fov_frwd = math.tan(math.radians(35/2)) * alt
    fov_side = math.tan(math.radians(55/2)) * alt
    lat = vehicle.location.global_relative_frame.lat
    lon = vehicle.location.global_relative_frame.lon
    origin = geopy.Point(lat, lon)
    pf = VincentyDistance(meters=fov_frwd).destination(origin, h)
    pb = VincentyDistance(meters=fov_frwd).destination(origin, h+180)
    p1 = VincentyDistance(meters=fov_side).destination(pf, h-90)
    p2 = VincentyDistance(meters=fov_side).destination(pf, h+90)
    p3 = VincentyDistance(meters=fov_side).destination(pb, h+90)
    p4 = VincentyDistance(meters=fov_side).destination(pb, h-90)
    p = [p1,p2,p3,p4,p1]
    side = []
    for i in range(4):
        side += [leftOrRight(p[i],p[i+1])]
        #print("%1.6f %1.6f" %(p[i].latitude,p[i].longitude))
    #print(side)
    if max(side) == min(side):
        print("fire detected")
    else: print("No fire detected")

def leftOrRight(lineP1,lineP2):
    #this function finds whether the point is on the left side of the
    #line or on the right side by calculating the bearing of p1 to p2
    # and p1 to point
    b1 = bearing(lineP1.latitude,lineP1.longitude, lineP2.latitude, lineP2.
   longitude)
    b2 = bearing(lineP1.latitude,lineP1.longitude, fireLoc.latitude, fireLoc.
   longitude)
    if b1 - b2 > 180:
        return "LEFT"
    elif b1 -b2 > 0:
        return "RIGHT"
    elif b1 - b2 < -180:
        return "RIGHT"
    else:
        return "LEFT"

def bearing(lat1,lon1,lat2,lon2):
    lat1 = math.radians(lat1)
    lon1 = math.radians(lon1)
    lat2 = math.radians(lat2)
    lon2 = math.radians(lon2)
    dLon = lon2 - lon1;
    y = math.sin(dLon) * math.cos(lat2);
    x = math.cos(lat1)*math.sin(lat2) - math.sin(lat1)*math.cos(lat2)*math.cos
   (dLon);
```

```python
    brng = np.rad2deg(math.atan2(x, y));
    return brng

def findCameraSide(diff2):
    if diff2 < 0:
        diff2 += 360
    if diff2 < 180:
        x = +1
    else:
        x = -1
    if diff2 > 90 and diff2 < 270:
        y = -1
    else:
        y = 1
    return x,y

def get_distance_metres(aLocation1, aLocation2):

    dlat = aLocation2.lat - aLocation1.lat
    dlong = aLocation2.lon - aLocation1.lon
    return math.sqrt((dlat*dlat) + (dlong*dlong)) * 1.113195e5
def oneHz():
    cameraView()
    threading.Timer(1,oneHz).start()

def send_to_FG():

    #print("%.3f" %time.time())
    p = vehicle.attitude.pitch*180/3.14
    h = vehicle.attitude.yaw*180/3.14
    alt = 50 + vehicle.location.global_relative_frame.alt
    lat = vehicle.location.global_relative_frame.lat
    lon = vehicle.location.global_relative_frame.lon
   # print("%.8f %.8f" %(lat,lon))
    r = vehicle.attitude.roll*180/3.14
  my_message = "%.2f,%.2f,%i,%f,%f,%i\n" %(r,p,h,lat,lon,alt)
    byte_message = bytes(my_message,"utf-8")
    #print(my_message+" %2.4f" %time.time())
    opened_socket.sendto(byte_message, ("192.168.2.12", 5006))
    threading.Timer(0.02,send_to_FG).start()

def arm_and_takeoff(aTargetAltitude):
    """
    Arms vehicle and fly to aTargetAltitude.
    """

    print("Basic pre-arm checks")
    # Don't try to arm until autopilot is ready
    while not vehicle.is_armable:
        print(" Waiting for vehicle to initialise...")
        time.sleep(1)

    print("Arming motors")
    # Copter should arm in GUIDED mode
    vehicle.mode = VehicleMode("GUIDED")
    vehicle.armed = True

    # Confirm vehicle armed before attempting to take offs
    while not vehicle.armed:
        print(" Waiting for arming...")
        time.sleep(1)

    print("Taking off!")
    vehicle.simple_takeoff(aTargetAltitude)  # Take off to target altitude

    # Wait until the vehicle reaches a safe height before processing the goto
    #  (otherwise the command after Vehicle.simple_takeoff will execute
    #   immediately).
    while True:
        #
        print(" Altitude: ", vehicle.location.global_relative_frame.alt)
        # Break and return from function just below target altitude.
        if vehicle.location.global_relative_frame.alt >= aTargetAltitude *
   0.95:
            print("Reached target altitude")
            break
        time.sleep(1)

send_to_FG()
oneHz()
arm_and_takeoff(20)
```

Bibliography

[1] J. Martinez-de Dios, B. Arrue, A. Ollero, L. Merino, and F. Gómez-Rodríguez, “Computer vision techniques for forest fire perception,” *Image and Vision Computing*, vol. 26, no. 4, pp. 550–562, 2008.

[2] W. Lee, S. Kim, Y. T. Lee, H. W. Lee, and M. Choi, “Deep neural networks for wild fire detection with unmanned aerial vehicle,” in *2017 IEEE International Conference on Consumer Electronics (ICCE)*, pp. 252–253, 2017.

[3] S. Rudz, K. Chetehouna, A. Hafiane, H. Laurent, and O. Séro-Guillaume, “Investigation of a novel image segmentation method dedicated to forest fire applications,” *Measurement Science and Technology*, vol. 24, no. 7, p. 075403, 2013.

[4] T. Ingalsbee and U. Raja, “The rising costs of wildfire suppression and the case for ecological fire use,” in *The Ecological Importance of Mixed-Severity Fires* (D. A. DellaSala and C. T. Hanson, eds.), pp. 348–371, Elsevier, 2015.

[5] K. Skala and A. DUBRAVIĆ, “Integrated system for forest fire early detection and management,” *Periodicum Biologorum*, vol. 110, no. 2, 2008.

[6] M. A. Cochrane, “Fire science for rainforests,” *Nature*, vol. 421, no. 6926, pp. 913–919, 2003.

[7] V. G. Ambrosia and T. Zajkowski, *Selection of Appropriate Class UAS/Sensors to Support Fire Monitoring: Experiences in the United States*, pp. 2723–2754. Dordrecht: Springer Netherlands, 2015.

[8] C. I. F. F. Centre, "Annual area burned in Canada," 2020. https://ciffc.net/en/ext/hectares-by-year.

[9] R. S. Allison, J. M. Johnston, G. Craig, and S. Jennings, "Airborne optical and thermal remote sensing for wildfire detection and monitoring," *Sensors (Basel, Switzerland)*, vol. 16, no. 8, 2016.

[10] E. den Breejen, M. Breuers, F. Cremer, R. Kemp, M. Roos, K. Schutte, and J. De Vries, "Autonomous forest fire detection," *International Conference on Forest Fire Research*, vol. 2, 2001.

[11] A. E. Çetin, K. Dimitropoulos, B. Gouverneur, N. Grammalidis, O. Günay, Y. H. Habiboğlu, B. U. Töreyin, and S. Verstockt, "Video fire detection – Review," *Digital Signal Processing*, vol. 23, no. 6, pp. 1827–1843, 2013.

[12] H. Olsson, M. Egberth, J. Engberg, J. E. Fransson, T. G. Pahlén, *et al.*, "Current and emerging operational uses of remote sensing in swedish forestry," in *Proceedings of the Seventh Annual Forest Inventory and Analysis Symposium; 2005; Portland, ME.*, vol. 77, 2007.

[13] R. W. Beard, T. W. McLain, D. B. Nelson, D. Kingston, and D. Johanson, "Decentralized cooperative aerial surveillance using fixed-wing miniature UAVs," *Proceedings of the IEEE*, vol. 94, no. 7, pp. 1306–1324, 2006.

[14] C. R. Butler, M. B. O'Connor, and J. M. Lincoln, "Aviation-related wildland firefighter fatalities — United States, 2000–2013," *MMWR Morbidity and Mortality Wkly. Rep.*, vol. 64, pp. 793–796, 2015.

[15] F. A. Hossain, Y. M. Zhang, and C. Yuan, "A survey on forest fire monitoring using unmanned aerial vehicles," in *2019 3rd International Symposium on Autonomous Systems (ISAS)*, pp. 484–489, 2019.

[16] C. Yuan, Z. Liu, and Y. M. Zhang, "UAV-based forest fire detection and tracking using image processing techniques," in *2015 International Conference on Unmanned Aircraft Systems (ICUAS)*, pp. 639–643, 2015.

[17] C. Yuan, Y. M. Zhang, and Z. Liu, "A survey on technologies for automatic forest fire monitoring, detection, and fighting using unmanned aerial vehicles and remote sensing techniques," *Canadian Journal of Forest Research*, vol. 45, no. 7, pp. 783–792, 2015.

[18] F. Sharifi, A. Chamseddine, H. Mahboubi, Y. M. Zhang, and A. G. Aghdam, "A distributed deployment strategy for a network of cooperative autonomous vehicles," *IEEE Transactions on Control Systems Technology*, vol. 23, no. 2, pp. 737–745, 2015.

[19] M. Mirzaei, F. Sharifi, B. W. Gordon, C. A. Rabbath, and Y. M. Zhang, "Cooperative multi-vehicle search and coverage problem in uncertain environments," in *2011 50th IEEE Conference on Decision and Control and European Control Conference*, pp. 4140–4145, 2011.

[20] E. Pastor, C. Barrado, P. Royo, E. Santamaria, J. Lopez, and E. Salami, "Architecture for a helicopter-based unmanned aerial systems wildfire surveillance system," *Geocarto International*, vol. 26, no. 2, pp. 113–131, 2011.

[21] G. P. J. IV, L. G. Pearlstine, and H. F. Percival, "An assessment of small unmanned aerial vehicles for wildlife research," *Wildlife Society Bulletin*, vol. 34, no. 3, pp. 750–758, 2006.

[22] B. R. Christensen, "Use of uav or remotely piloted aircraft and forward-looking infrared in forest, rural and wildland fire management: evaluation using simple economic analysis," *New Zealand Journal of Forestry Science*, vol. 45, no. 1, p. 16, 2015.

[23] A. Rango, A. Laliberte, C. Steele, J. E. Herrick, B. Bestelmeyer, T. Schmugge, A. Roanhorse, and V. Jenkins, "Using unmanned aerial vehicles for rangelands: Current applications and future potentials," *Environmental Practice*, vol. 8, no. 3, pp. 159–168, 2006.

[24] T. Celik, H. Ozkaramanlt, and H. Demirel, "Fire pixel classification using fuzzy logic and statistical color model," in *2007 IEEE International Conference on Acoustics, Speech and Signal Processing-ICASSP'07*, vol. 1, pp. 1205–1208, IEEE, 2007.

[25] V. Ambrosia, S. Wegener, D. Sullivan, S. Buechel, S. Dunagan, J. Brass, J. Stoneburner, and S. Schoenung, "Demonstrating UAV-acquired real-time thermal data over fires," *Photogrammetric Engineering Remote Sensing*, vol. 69, pp. 391–402, 2003.

[26] V. G. Ambrosia, S. Wegener, T. Zajkowski, D. Sullivan, S. Buechel, F. Enomoto, B. Lobitz, S. Johan, J. Brass, and E. Hinkley, "The Ikhana unmanned airborne system (UAS) western states fire imaging missions: from concept to reality (2006–2010)," *Geocarto International*, vol. 26, no. 2, pp. 85–101, 2011.

[27] J. R. Martínez-de Dios, L. Merino, and A. Ollero, "Fire detection using autonomous aerial vehicles with infrared and visual cameras," *IFAC Proceedings Volumes*, vol. 38, no. 1, pp. 660–665, 2005.

[28] M. V. Persie, A. Oostdijk, J. Fix, M. van Sijll, and L. Edgardh, "Real-time UAV based geospatial video integrated into the fire brigades crisis management GIS system," *ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, vol. XXXVIII-1/C22, 2012.

[29] M. Tranchitella, S. Fujikawa, T. L. C. Ng, D. Yoel, D. Tatum, P. Roy, C. Mazel, S. Herwitz, and E. Hinkley, "Using tactical unmanned aerial systems to monitor and map wildfires," *AIAA Infotech@Aerospace Conference and Exhibit*, 2007.

[30] F. Esposito, G. Rufino, A. Moccia, P. Donnarumma, M. Esposito, and V. Magliulo, “An integrated electro-optical payload system for forest fires monitoring from airborne platform,” in *2007 IEEE Aerospace Conference*, pp. 1–13, IEEE, 2007.

[31] G. P. Jones, *The feasibility of using small unmanned aerial vehicles for wildlife research*. PhD thesis, University of Florida, USA, 2003.

[32] A. Restas, “Wildfire management supported by UAV based air reconnaissance: experiments and results at the szendro fire department, Hungary,” in *First International Workshop on Fire Management*, 2006.

[33] R. Charvat, R. Ozburn, S. Bushong, K. Cohen, and M. Kumar, “Sierra team flight of Zephyr UAS at West Virginia wild land fire burn,” *Infotech@Aerospace*, 2012.

[34] T. H. Chen, P. H. Wu, and Y. C. Chiou, “An early fire-detection method based on image processing,” in *2004 International Conference on Image Processing*, vol. 3, pp. 1707–1710, 2004.

[35] B. U. Töreyin, Y. Dedeoğlu, and A. E. Çetin, “Flame detection in video using hidden Markov models,” in *IEEE International Conference on Image Processing 2005*, vol. 2, pp. II–1230, 2005.

[36] B. U. Töreyin, Y. Dedeoğlu, U. Güdükbay, and A. E. Çetin, “Computer vision based method for real-time fire and flame detection,” *Pattern Recognition Letters*, vol. 27, no. 1, pp. 49–58, 2006.

[37] B. U. Töreyin, Y. Dedeoğlu, and A. E. Çetin, “Contour based smoke detection in video using Wavelets,” in *2006 14th European Signal Processing Conference*, pp. 1–5, IEEE, 2006.

[38] T. Çelik and H. Demirel, “Fire detection in video sequences using a generic color model,” *Fire Safety Journal*, vol. 44, no. 2, pp. 147–158, 2009.

[39] W. Phillips, M. Shah, and N. Da Vitoria Lobo, "Flame recognition in video," *Pattern Recognition Letters*, vol. 23, no. 1-3, pp. 319–327, 2002.

[40] C. Yuan, K. A. Ghamry, Z. Liu, and Y. M. Zhang, "Unmanned aerial vehicle based forest fire monitoring and detection using image processing technique," in *2016 IEEE Chinese Guidance, Navigation and Control Conference (CGNCC)*, pp. 1870–1875, IEEE, 2016.

[41] T. Chen, Y. Yin, S. Huang, and Y. Ye, "The smoke detection for early fire-alarming system base on video processing," in *2006 International Conference on Intelligent Information Hiding and Multimedia*, pp. 427–430, 2006.

[42] C. Yu, Y. Zhang, J. Fang, and J. Wang, "Texture analysis of smoke for real-time fire detection," in *2009 Second International Workshop on Computer Science and Engineering*, vol. 2, pp. 511–515, 2009.

[43] C. Yuan, Z. Liu, and Y. M. Zhang, "Vision-based forest fire detection in aerial images for firefighting using UAVs," in *2016 International Conference on Unmanned Aircraft Systems (ICUAS)*, pp. 1200–1205, 2016.

[44] C. C. Ho and T. H. Kuo, "Real-time video-based fire smoke detection system," in *2009 IEEE/ASME International Conference on Advanced Intelligent Mechatronics*, pp. 1845–1850, 2009.

[45] C. Yu, J. Fang, J. Wang, and Y. Zhang, "Video fire smoke detection using motion and color features," *Fire Technology*, vol. 46, pp. 651–663, 2010.

[46] S. Ham, B. Ko, and J. Nam, "Fire-flame detection based on fuzzy finite automation," in *2010 20th International Conference on Pattern Recognition*, pp. 3919–3922, IEEE, 2010.

[47] J. R. Martinez-de Dios, J. André, J. Gonçalves, B. Arrue, A. Ollero, and D. Viegas, "Laboratory fire spread analysis using visual and infrared images," *International Journal of Wildland Fire*, vol. 15, pp. 179–186, 2006.

[48] A. Ollero, B. Arrue, J. Martinez, and J. Murillo, "Techniques for reducing false alarms in infrared forest-fire automatic detection systems," *Control Engineering Practice*, vol. 7, no. 1, pp. 123–131, 1999.

[49] B. C. Arrue, A. Ollero, and J. R. Matinez de Dios, "An intelligent system for false alarm reduction in infrared forest-fire detection," *IEEE Intelligent Systems and Their Applications*, vol. 15, no. 3, pp. 64–73, 2000.

[50] I. Bosch, A. Serrano, and L. Vergara, "Multisensor network system for wildfire detection using infrared image processing," *The Scientific World Journal*, vol. 2013, 2013.

[51] J. J. Huseynov, S. Baliga, A. Widmer, and Z. Boger, "Infrared flame detection system using multiple neural networks," in *2007 International Joint Conference on Neural Networks*, pp. 608–612, 2007.

[52] A. E. Ononye, A. Vodacek, and E. Saber, "Automated extraction of fire line parameters from multispectral infrared images," *Remote Sensing of Environment*, vol. 108, no. 2, pp. 179–188, 2007.

[53] L. Merino, F. Caballero, J. R. Martinez-de Dios, and A. Ollero, "Cooperative fire detection using unmanned aerial vehicles," in *Proceedings of the 2005 IEEE International Conference on Robotics and Automation*, pp. 1884–1889, 2005.

[54] J. Martinez-de Dios and A. Ollero, "A new training-based approach for robust thresholding," in *Proceedings World Automation Congress*, vol. 18, pp. 121–126, IEEE, 2004.

[55] I. Bosch, A. Serrano, and L. Vergara, "Multisensor network system for wildfire detection using infrared image processing," *The Scientific World Journal*, vol. 2013, p. 402196, 2013.

[56] E. Pastor, A. Àgueda, J. Andrade Cetto, M. Muñoz Messineo, Y. Perez, and E. Planas, "Computing the rate of spread of linear flame fronts by thermal image processing," *Fire Safety Journal*, vol. 41, pp. 569–579, 2006.

[57] A. Ononye, A. Vodacek, and E. Saber, "Automated extraction of fire line parameters from multispectral infrared images," *Remote Sensing of Environment*, vol. 108, pp. 179–188, 2007.

[58] J. J. Huseynov, S. Baliga, A. Widmer, and Z. Boger, "Infrared flame detection system using multiple neural networks," in *2007 International Joint Conference on Neural Networks*, pp. 608–612, IEEE, 2007.

[59] C. Yuan, Z. Liu, and Y. M. Zhang, "Fire detection using infrared images for UAV-based forest fire surveillance," in *2017 International Conference on Unmanned Aircraft Systems (ICUAS)*, pp. 567–572, 2017.

[60] E. Wardihani, M. Ramdhani, A. Suharjono, T. Setyawan, S. S. Hidayat, Helmy, S. Widodo, E. Triyono, and F. Saifullah, "Real-time forest fire monitoring system using unmanned aerial vehicle," *Journal of Engineering Science and Technology*, vol. 13, pp. 1587–1594, 2018.

[61] Melexis inspired engineering, *MLX90640 32x24 IR array datasheet*, 2019. Available at https://www.melexis.com/-/media/files/documents/datasheets/mlx90640-datasheet-melexis.pdf.

[62] "Adafruit pwm servo driver library." https://github.com/adafruit/Adafruit-PWM-Servo-Driver-Library.

[63] S. Liu, "Image-based visual servoing using improved image moments in 6-dof robot systems," Master's thesis, Concordia University, Canada, 2008.

[64] R. Gonzalez and R. Woods, *Digital Image Processing*. Pearson/Prentice Hall, 2006.

[65] "Dronekit-python library for communicating with drones via mavlink." https://github.com/dronekit/dronekit-python.