University Institute of Lisbon

Department of Information Science and Technology

Autonomous Environmental Protection Drone

Carlos Miguel Domingues Saraiva

A Dissertation presented in partial fulfillment of the Requirements for the Degree of

Master in Telecommunications and Computer Engineering

Supervisor

Prof. Dr. Pedro Joaquim Amaro Sebastião, Assistant Professor

ISCTE-IUL

Co-Supervisor

Eng. António Sérgio Lima Raimundo, PhD Student

ISCTE-IUL

October 2019

Resumo

Durante o verão, os incêndios florestais constituem a principal razão do desflorestamento e dos danos causados às casas e aos bens das diferentes comunidades de todo o mundo. A utilização de veículos aéreos não tripulados (VANTs), em inglês denominados por Unmanned Aerial Vehicles (UAVs) ou Drones, aumentou nos últimos anos, tornando-os uma excelente solução para tarefas difíceis como a conservação da vida selvagem e prevenção de incêndios florestais. Um sistema de deteção de incêndio florestal pode ser uma resposta para essas tarefas. Com a utilização de uma câmara visual e uma Rede Neuronal Convolucional (RNC) para processamento de imagem com um UAV pode resultar num eficiente sistema de deteção de incêndio. No entanto, para que seja possível ter um sistema completamente autónomo, sem intervenção humana, para observação e deteção de incêndios durante 24 horas, numa dada área geográfica, requer uma plataforma e procedimentos de recarga automática. Esta dissertação reúne o uso de tecnologias como RNCs, posicionamento cinemático em tempo real (RTK) e transferência de energia sem fios (WPT) com um computador e software de bordo, resultando num sistema totalmente automatizado para tornar a vigilância florestal mais eficiente e, ao fazê-lo, realocando recursos humanos para outros locais, onde estes são mais necessários.

Palavras-chave: Veículos Aéreos Não Tripulados, Rede Neuronal Convolucional, Ambiente, Fogo Florestal, Deteção de fogo, Posicionamento Cinemático em Tempo Real.

Abstract

During the summer, forest fires are the main reason for deforestation and the damage caused to homes and property in different communities around the world. The use of *Unmanned Aerial Vehicles* (UAVs, and also known as drones) applications has increased in recent years, making them an excellent solution for difficult tasks such as wildlife conservation and forest fire prevention. A forest fire detection system can be an answer to these tasks. Using a visual camera and a Convolutional Neural Network (CNN) for image processing with an UAV can result in an efficient fire detection system. However, in order to be able to have a fully autonomous system, without human intervention, for 24-hour fire observation and detection in a given geographical area, it requires a platform and automatic recharging procedures. This dissertation combines the use of technologies such as CNNs, Real Time Kinematics (RTK) and Wireless Power Transfer (WPT) with an on-board computer and software, resulting in a fully automated system to make forest surveillance more efficient and, in doing so, reallocating human resources to other locations where they are most needed.

Keywords: Unmanned Aerial Vehicles, Convolutional Neural Network, Environment, Autonomous, Forest Fire, Fire Detection, Real Time Kinematics.

Contents

List of Figures

List of Tables

Abbreviations

1G	1-**G**eneration
2G	2-**G**eneration
3D	3-**D**imensions
3G	3-**G**eneration
4G	4-**G**eneration
AC	**A**lternating **C**urrent
ANAC	**A**gência **N**acional de **A**viação **C**ivil
ADC	**A**nalog-to-**D**igital **C**onverters
AI	**A**rtificial **I**ntelligence
ANN	**A**rtificial **N**eural **N**etwork
API	**A**pplication **P**rogramming **I**nterface
ARP	**A**ddress **R**esolution **P**rotocol
CAN	**C**ontroller **A**rea **N**etwork
CNN	**C**onvolutional **N**eural **N**etwork
CPU	**C**entral **P**rocessing **U**nit
CSI	**C**amera **S**erial **I**nterface
DC	**D**irect **C**urrent
DGNSS	**D**iferential **GNSS**
DGPS	**D**ifferential **GPS**
EKF	**E**xtended **K**alman **F**ilter
ESC	**E**lectronic **S**peed **C**ontroller
FPS	**F**rames **P**er **S**econd
FPU	**F**loating-**P**oint **U**nit
FTP	**F**ile **T**ransfer **P**rotocol

GCS	**G**round **C**ontrol **S**tation
GLONASS	**GLO**bal **NA**vigation **S**atellite **S**ystem
GNSS	**G**lobal **N**avigation **S**atellite **S**ystem
GPIO	**G**eneral **P**urpose **I**nput / **O**utput
GPS	**G**lobal **P**ositioning **S**ystem
GPU	**G**raphics **P**rocessing **U**nit
GSD	**G**round **S**ampling **D**istance
HALE	**H**igh **A**ltitude **L**ong **E**ndurance
HDMI	**H**igh-**D**efinition **M**ultimedia **I**nterface
HTTP	**H**yper **T**ext **T**ransfer **P**rotocol.
I/O	**I**nput / **O**utput
I2C	**I**nter-integrated **C**ircuit
IC	**I**ntegrated **C**ircuit
ID	**ID**entification
IMU	**I**nertia **M**easurement **U**nit
IP	**I**nternet **P**rotocol
IT	**I**nstituto de **T**elecomunicações
IUL	**I**nstituto **U**niversitário de **L**isboa
LED	**L**ight **E**mitting **D**iode
LGPL	**L**esser **G**eneral **P**ublic **L**icense
LTE	**L**ong **T**erm **E**volution
MALE	**M**edium **A**ltitude **L**ong **E**ndurance
MAV	**M**icro **UAV**
MCM	**M**ulti-**C**hip **M**odule
MEMS	**M**icro-**E**lectro-**M**echanical **S**ystems
MLL	**M**achine **L**earning **I**nterface
MUAV	**M**ini **UAV**
NAV	**N**ano **UAV**
NaN	**N**ot a **N**umber
OS	**O**perating **S**ystem
PC	**P**ersonal **C**omputer

QZSS	**Q**uasi-**Z**enith **S**atellite **S**ystem
RAM	**R**andom **A**ccess **M**emory
RC	**R**adio **C**ontrolled
RGB	**R**ed **G**reen **B**lue
RNC	**R**ede **N**euronal **C**onvolucional
ROS	**R**obot **O**perating **S**ystem
RTCM	**R**adio **T**echnical **C**ommission for **M**aritime **S**ervices
RTK	**R**eal **T**ime **K**inematic
RTL	**R**eturn **T**o **L**aunch
SBAS	**S**atellite-**B**ased **A**ugmentation **S**ystems
SD	**S**ecure **D**igital
SITL	**S**oftware **I**n **T**he **L**oop
TUAV	**T**actical **UAV**
UART	**U**niversal **A**synchronous **R**eceiver-**T**ransmitter
UAVCAN	**UAV CAN**
UAV	**U**nmanned **A**erial **V**ehicles
UDP	**U**ser **D**atagram **P**rotocol
URL	**U**niform **R**esource **L**ocator
USB	**U**niversal **S**erial **B**us
WPT	**W**ireless **P**ower **T**ransfer

Chapter 1

Introduction

1.1 Motivation and Context

For a long time forests have always been a big part of our ecosystem. However, humanity hasn't always been very good at cleaning, protecting or ensuring its prosperity. Forest fires, also known as wild fires, are uncontrolled fires that occur in wild areas and can cause significant damage to natural and human resources. These fires have the power to eradicate forests, burn infrastructures and may even result in high human death toll near urban areas [1]. In 2017, 10 percent of the Portuguese forest got burned down due to a number of reasons which included natural disasters, criminal human activity and the lack of resources to fight these disasters. Moreover, apart from Portugal there are other countries like The United States of America, Indonesia or Canada that suffer from the same problem causing the deforestation rate to increase rapidly.

A number of systems for automatic forest fire detection have been developed based on different sensors [2]. Some of the first automatic forest fire detection systems were based on infrared cameras capable of detecting the radiation emitted by a fire [3]. Considering the technological era that the world is currently facing, there should be other ways to automate the prevention of these types of environmental catastrophes that do not rely solely on human resources.

Following up to the presented problem, this project aims to create a system based on UAV (*Unmanned Aerial Vehicle*) technology. This system would collect real time data from a camera installed on an UAV to detect forest fires, alerting the authorities on the moment of detection. This could be a big advantage when used in remote places and isolated areas where human access is difficult and sometimes almost impossible. The UAV would be programmed to fly over designated coordinates from time to time while collecting data, and then autonomously return to its base station where it would automatically land and recharge on a solar based powered station. Once fully recharged it would again patrol the area to ensure the absence of fires or report if any is detected.

It is expected that the development of such system would have a great positive impact, not only because it could allow cost and time reduction in terms of human resources, but also for having a much more efficient forest control system.

1.2 Goals

The purpose of this project is to create a fully automated system to make forest surveillance more efficient and by doing it so, reallocating human resources to where they are most needed. The UAV for the study is being provided by Instituto de Telecomunicações (IT-IUL) located in ISCTE-IUL, Lisbon, Portugal establishing the start point for this research. The first and most important goal is the development of the forest fire detection system, to deliver an early and accurate fire detection alarm. The system will use an image processing artificial intelligence algorithm running on the UAV's companion computer to detect fire pixels and then send a notification.

Secondly, to have a fully automated procedure, the UAV has to be able to land autonomously at the recharging platform with high accuracy. This is a high-risk procedure considering that sometimes winds can go up to 25 km/h making it extremely hard to land. A Real Time Kinematic (RTK) GNSS system will be used to help with the take off and landing procedures providing the UAV with highly accurate manoeuvres.

Lastly, there is the development of the wireless transfer solar powered station where the UAV will dock and perform battery recharge. This station will use two separated coils, one attached to the UAV and another at the recharging station. Since the charging process is wireless, the need for both coils to be aligned is high, therefore the necessity for a high precision landing. Apart from the coils, the station will also have a battery that will be solar powered. In case of low solar radiation, there is the possibility of this recharging station to be placed in the old forest protection outposts that have a generator.

1.3 Research Questions and Methods

Before starting to develop this project, some questions need to be made regarding the system. These questions target the problems that the project aims to solve and the purpose for developing it. The questions are:

- How will the system improve the governmental fire surveillance system?
- Will the system meet the requirements needed for an accurate early forest fire detection?
- Will the system be efficient for daily use and will the low budget materials outweigh the performance needed?

In order to achieve the goals previously specified in section 1.2, some research methods need to be applied. These methods will use quantitative variables to recover data, which will then be used to test the system and afterwards evaluate it and improve it if necessary. This data will also dictate the efficiency of the system, answering the last research question about its performance and implementation.

1.4 Main Contributions

This dissertation's work has contributed for several entities as well as for the scientific community. The dissertation's application area promotes UAV applications for institutions such as:

- Wildlife conservation institutes;
- Firefighters;
- Civil Protection Services.

To the scientific community this research's contributions reveals the value this technology can have for high precision UAV applications, automated recharging platforms and image processing using CNNs. One of the result of this contributions in this community was the submitted journal paper for IEEE Access:

C. Saraiva, A. Raimundo, P. Sebastião, *"Autonomous Fire Detection System supported By UAVs and Convolutional Neural Networks"*, IEEE Access.

1.5 Dissertation Structure

This section contains the dissertation's structure. In total, it has 7 chapters and each chapter contains the following topics:

- Chapter 1 – Introduction: Presents the introductory aspects related to the subject of the dissertation;
- Chapter 2 – State of the Art: Contains all the methodology used and the research done before this dissertation's work;
- Chapter 3 – UAV Flight Automation: Details how the automatic procedures of the UAV were planned and preformed;

- Chapter 4 – Fire Detection Algorithm and Real-Time Detection: Describes the development of the image processing algorithm and the tools used for that purpose and how the algorithm was used to detect early fire;
- Chapter 5 – Base Station and Landing Procedure: Explains how the high precision landing was achieved and how the recharging station was designed and how it could be implemented;
- Chapter 6 – Results: Shows this dissertation's results from different test simulations, real life's and algorithm's performance evaluation;
- Chapter 7 – Conclusions and Future Work: Describes the conclusions obtained during the dissertation and how it can be improved.

In addition to the previous chapters, the chapter "Appendices" was added, and it contains all the supplementary information that can be used to complement the dissertation's comprehension.

Chapter 2

State of the Art

2.1 Artificial Intelligence

Artificial Intelligence (AI) is the continuous study of intelligence behaviour displayed by machines. AI has two main branches: engineering and scientific. The engineering branch is focused on building and developing intelligent machines, while the scientific one is focused on developing the concept and the vocabulary to help us understand intelligent behaviour. There are 3 type of AIs: analytical, human-inspired, and humanized artificial intelligence. With the evolution of computers and algorithmic improvements, new methods of studying algorithms emerged, evolving into today's concepts such as machine learning and deep learning [4][5].

2.1.1 Machine Learning

Machine Learning is the scientific study of algorithms and statistical models that computer systems use to perform a specific task without being programmed to do so. By recreating the learning procedure that humans do, using mathematical models based in "training data", computers can make predictions or decisions. When training algorithms, computers use the training data to know which specific patterns to look for in data later used as input, the training procedure allows them to identify and recognize specific objects or patterns. The more data is used to train the computer the more accurate the detection will be. Machine learning is used nowadays for a lot of different tasks such as email filtering, credit card fraud detection and computer vision tasks [6][7].

2.1.2 Deep Learning

Nowadays, Deep Learning is the most used technique for implementing machine learning. Deep learning evolved to solve the biggest problem in machine learning, which was the ability to extract high-level features from raw data. As referred earlier in Chapter 2.1.1, this concept is about the scientific study of algorithms, however deep learning is a lot more specific. Deep learning applies algorithms inspired by the structure and function of the brain called artificial neural networks. These structures are composed of multiple processing layers to learn representations of data with several levels of abstraction. The learning process in these networks can be supervised, semi-supervised or unsupervised [8][9][10][11]. Deep learning has brought breakthroughs in the areas of processing images, video, speech and audio, where other machine learning techniques have only shown some light on sequential data such as text and speech.

2.1.3 Artificial Neural Networks

Artificial Neural Networks (ANN) also know as connectionist systems are inspired by artificial representation of humans' nervous system. ANNs are networks connected by artificial neurons which are connected to each other, working just like synapses in the human brain. These synapses have weights represented by real numbers. They are able to receive, process and transmit signals between them. An ANN has a number of layers and each layer is composed by a different number of artificial neurons. The artificial neurons turn on or off as the input is passed along the net. The output of each layer is simultaneously the subsequent layer's input, with the initial layer receiving the input data as a starting point [12]. The layers structure is shown in figure 2.1.

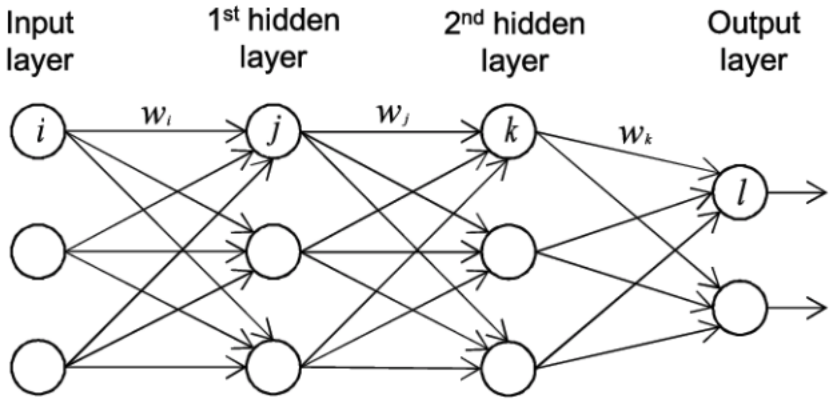


FIGURE 2.1: Artificial neural network (Source: [13]).

The neurons, also known as nodes, combine the input from the data with a set of weights, that either increase or decrease the input. The product of the input and weights are summed $\sum_{i=1}^{n} x_i w_i$ and the result is passed through a node's so called activation function $f(\sum_{i=1}^{n} x_i w_i)$, as shown in figure 2.2. The function calculates if the signal should progress further through the network and the activation value. If the signal passes through it becomes activated and affects the final outcome [14].

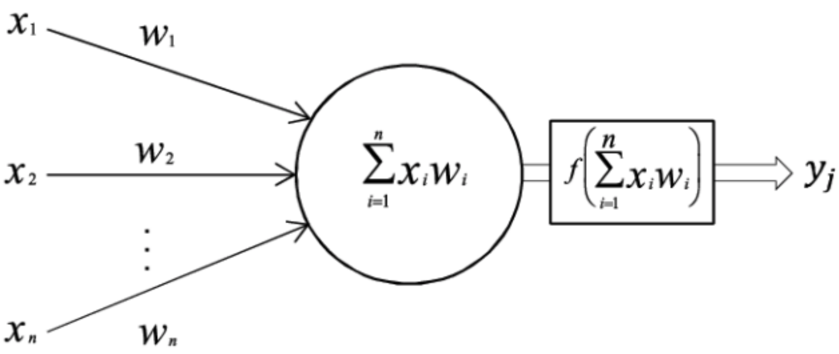


FIGURE 2.2: Artificial neuron.

There are many types of ANNs: autoencoder, probabilistic, time delay, and convolutional (CNN). For this project the selected ANN will be the CNN because not only is it faster to train but also displays more accurate results for image processing [15].

2.1.4 Convolutional Neural Networks

Convolutional Neural Networks (CNN or ConvNets), are specific networks built to detect and understand patterns. Understanding these patterns is what makes CNNs so useful for image analysis. What differentiates convNets from a standard multi-layer network is the CNN's hidden convolutional layers, these layers are composed of four different types:

- *Image processing layer* - it is an optional pre-processing layer of predefined filters that remain fixed during training. Besides the raw input image provided to the network, other information such as edges and gradients are also provided [16].
- *Convolutional Layer* - is the layer responsible for capturing the Low-Level features such as edges, color, gradient orientation, etc. The architecture also adapts to the High-Level features if more layers are added, improving the network's understanding of the images in the dataset.
- *Pooling layer* - this layer is in charge of reducing the computational power required to process the data using dimensionality reduction. This layer is also used while training the model for extracting dominant features such as rotational and positional invariant. There are two types of Pooling: Max-Pooling, which returns the maximum value from the portion of the image covered by the Kernel, and Average Pooling which returns the average of all the values. Max Pooling performs as a noise suppressant, discarding the noisy activations and also removing the noise throughout the dimensionality reduction. Average Pooling only provides dimensionality reduction as a noise suppressing mechanism. Max Pooling usually provides better results than Average Pooling.
- *Fully-Connected layer* - this layer learns the non-linear combinations of the high-level features, represented by the output of the convolutional layer.

After passing through all the layers the final output is flatten and fed to a regular Neural Network for classification purposes. Afterwards the image is flatten into a column vector. The flattened output is fed to a feed-forward neural network and back-propagation is applied to every iteration of training. Over a series of epochs (steps), the model is able to distinguish between dominating and certain low-level features in images and label them using a classification technique [17]. The full architecture of a CNN is illustrated in figure 2.3.

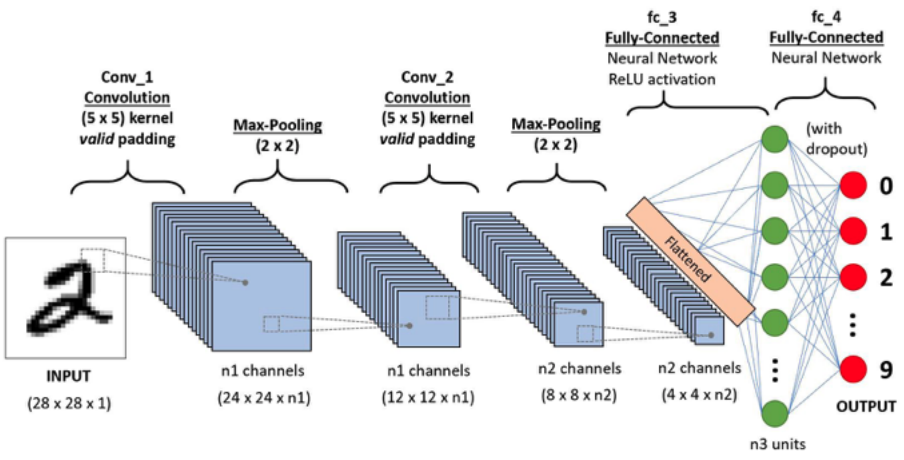


Figure 2.3: Convolutional neural network (Source: [17]).

2.2 Unmanned Aerial Vehicles

The wide applications of Unmanned Aerial Vehicles (UAVs) along the years have made this new technology part of our daily lives. A lot of companies have emerged due to this technology, from surveillance to radioactive analysis, the applications are infinite. Currently, the UAV performance is so high that it can make flights as long as 8 hours with maximum stability and reliability. In order to communicate with these UAVs, companies have created various open source software developed by programmers and UAV enthusiasts.

2.2.1 UAV Types

UAV types are distinguished by the altitude in which they operate and the different flight performance they have. Each one has its own operational process, functionalities and purposes:

HALE (High Altitude Long Endurance) – These vehicles can fly over 15 km of altitude, they are known for their flight autonomy up to 24 hours and their trans-global control range. They are controlled from fixed ground control stations by governmental Air Forces and their main application is to execute long-range military surveillance missions, reconnaissance and, if armed, aerial strikes in war zones;

MALE (Medium Altitude Long Endurance) – Similar to HALE, these are controlled from fixed ground stations, with a 500 km range. However, these vehicles operate at short ranges and fly at low altitudes (5 km – 15 km).

TUAV (Medium range or Tactical UAV) – Controlled by simpler systems, TUAVs are smaller than HALE and MALE, and they also operate at shorter ranges (100 km – 300 km). Usually used by the military from naval or air force ground bases. Their applications are more diverse being also used for power-line inspection, ship-to-shore surveillance, crop-spraying and traffic monitoring.

MUAV or Mini UAV– Even smaller than TUAVs, these light-weighted UAVs (20 kg of maximum weight) can be hand-launched and have even shorter ranges (30 km). They are used for a wide range of civilian purposes.

MAVs or Micro UAVs – These vehicles are smaller and lighter than any of the above, and can perform actions that no other UAV can. They can be used for indoor flights because of their slow flight with precision direction. These UAVs are commonly used for civilian purposes, because they are affordable, easily controllable and can perform simple actions. These UAVs flight altitude and weight is limited by local legislation.

NAV (Nano Air Vehicles)- As the name suggests they are vehicles of the size of a seed and they are most commonly used in swarm applications. They can be used in lots of other applications but are limited by the size of the sensors that they can use, like a camera.

In this dissertation, the focus will be on the applications of the most commonly used UAVs, the Micro Air Vehicles (MAVs) [18].

2.2.2 Different UAV Architectures

UAVs have different applications, therefore their structure can also change accordingly to their objective. Nowadays the most common MAVs are separated in two types: fixed-wing and multi-rotor, each being explained below [18].

Fixed-Wing: Similar to airplanes, this UAV can take advantage of the two wings' aerodynamics, allowing it to reach higher flying speeds and glide in the air. This systems make it possible to save energy because of the low power consumption needed to maintain altitude, increasing the flight's duration. Also, fixed-wing systems can be used to install solar panels allowing in-flight charging. This makes it possible to achieve a higher mission duration making them an optimal choice for applications such as surveillance missions. Also, due to its structure, take-off and landing is done horizontally and can't hover certain positions.

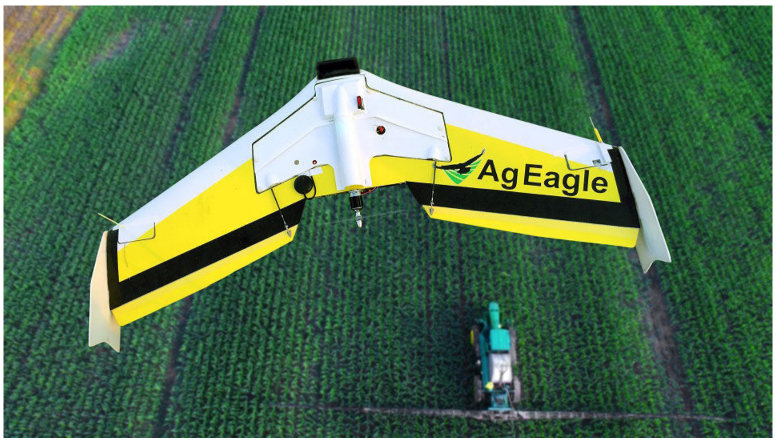


FIGURE 2.4: Fixed-Wing MAV.

Multi-rotor: These UAVs are designed similarly to a helicopter capable of hovering and standing still in mid-air. Due to a different structure and the increased amount of propellers, their energy consumption is increased and the flight duration drastically reduced. Even though the flight speed is low, it implicates more flight control. There are also a variety of different multi-rotors such as *Quadcopters*, *Hexacopters* and *Octacopters*. They lift off and land vertically which makes them ideal for automated procedures.

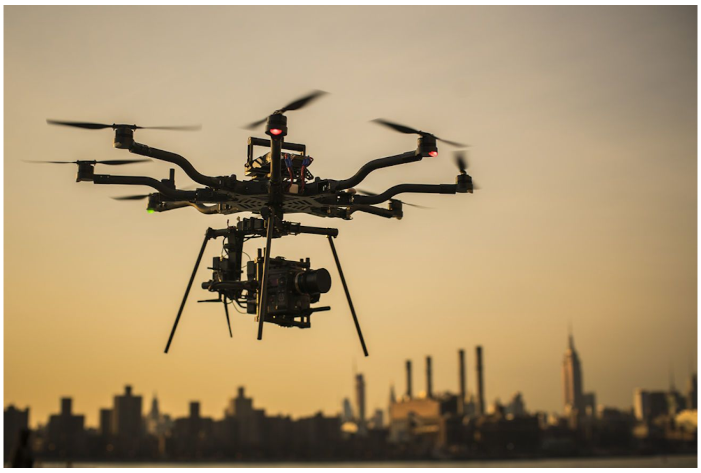


FIGURE 2.5: Multi-Rotor MAV.

2.2.3 UAV's Main Modules

An UAV requires different components in order to fly, since the used UAV is a multi-rotor it requires specific modules, such as:

- Frame - is the UAV's chassis which supports all payload (weight)[19];
- Rotor - are the UAV's motors, and they are placed one each frame's arm;
- Propellers - these are placed above the rotors and they are used to create a downwards air flow allowing enough strength to lift the UAV [19];
- Electronic Speed Controllers (ESC) - these components are the bridge between rotors and the flight controller, and they control the rotor's speed and rotation direction [19];

- Power supply - is the UAV's battery source, and usually they are lithium polymer batteries;
- Flight controller - is the UAV's centralized control unit for all the procedures that the UAV executes [19];
- GNSS sensor - is the UAV's main navigation sensor, usually the sensor used is the GPS and it allows to perform autonomous tasks, such as waypoint-guided missions;
- Companion computer - is the intermediary companion hardware between user's command and the flight controller [19].

Figure 2.6 shows the different models used on this dissertation's UAV, the ESCs are not visible however they are under each arm.

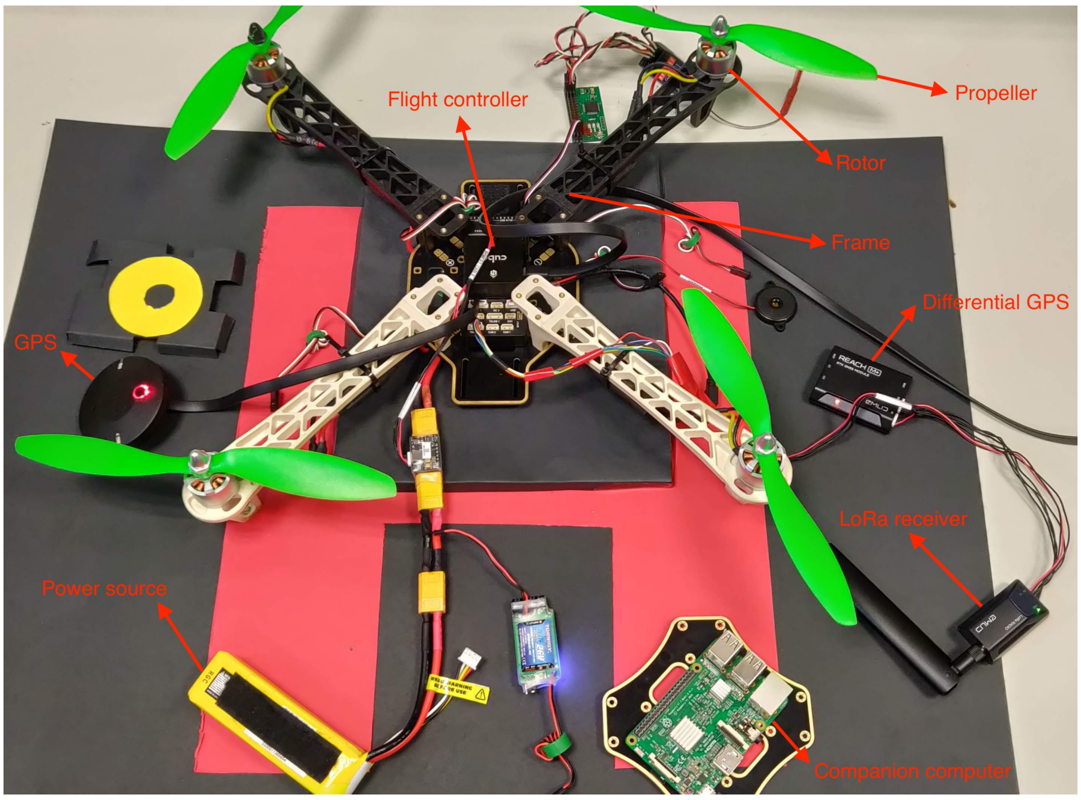


FIGURE 2.6: UAV modules.

2.3 Companion Computer and Flight Controller

An UAV's onboard flight controller, besides being the centralized control unit, also allows reading the sensors' data, sending and receiving commands from and to the Ground Control Stations (GCS). It is the UAV's heart and it controls all of the onboard electrical components like the motors and ESCs.

In order to be able to process all the information the flight controller receives, it also needs a onboard computer. This computer, also known as companion computer, is a low-powered computer that allows the UAV to take automated procedures with customized scripts, extend communications by using different protocols (such as 3G/4G) and even process algorithms based on the input received by the sensor's data. Both of these components are connected internally via serial connection, and they exchange data between them, thus allowing to create an automated system.

2.3.1 *Hex Cube Pixhawk 2.1*

The *Hex Cube Pixhawk 2.1* controller is based on the Pixhawk-project FMUv3 open hardware design,as shown in figure 2.7. It was built to reduce the wiring, improve reliability, and ease the assembly. Cube includes vibration isolation on two of the Inertial measurement units (IMUs), which is composed by 3-axis gyroscope and 3-axis accelerometer), with a third fixed IMU as a reference or for backup making it a perfect fit to operate in hard conditions and where GPS signal is low or non existent. Its main features are:

- 32bit STM32F427 Cortex-M4F core with FPU;
- 256 KB RAM;
- 32 bit STM32F103 fail-safe co-processor;
- Connectivity options for additional peripherals (UART, I2C, CAN);

- Redundant power supply inputs and automatic fail-over;
- MicroSD card for high-rate logging over extended periods of time.

Figure 2.7: Pixhawk Cube 2.1 flight controller.

2.3.2 *Raspberry Pi*

Raspberry is a low cost single board computer, of about the size of a credit card, as shown in figure 2.8. *Raspberry Pi* can run a wide range of Linux based Operating Systems (OS) like *Raspbian* or *Ubuntu* that come with a user friendly graphical interface and are installed on a Secure Digital (SD) card. The wide variety of OSs allows this mini-computer to have lots of applications such as web server, a personal cloud or even a media hub [20]. The version in use for this research, will be the *Raspberry Pi 3B* which has a Quad Core 1.2GHz 64bit CPU, 1 GB of RAM, Wi-fi, Bluetooth and other useful inputs, such as USB ports, HDMI port, RJ45 port, audio port and GPIO (General Purpose Input / Output) connectors. These GPIO connectors are used to control sensors like reading buttons, and switches and can control LEDs, relays or even motors.

In order to have it up and running, one should either buy an SD memory card with an operating system pre-installed or download one's own OS from raspberrypi.org and install it on the SD card. After, one should add the required additional packages, such as compilers and libraries [21].

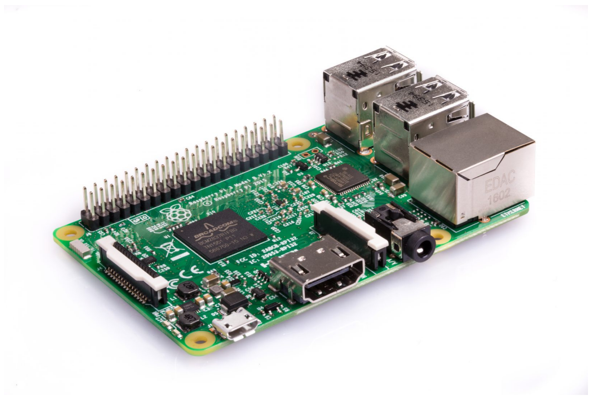


Figure 2.8: Raspberry Pi board.

The small size, light weight, low cost and power consumption, and the communication capabilities are the biggest advantages of using the *Raspberry Pi* when compared to other solutions. All these advantages make this mini-computer the optimal choice to use in this dissertation.

2.3.3 Jetson TX2

Jetson TX2 just like *Raspberry* is a single board computer, however is not a low cost solution for a companion computer. It is a high performance computer build by the NVIDIA company to be a power-efficient embedded AI. It has a dual-core NVIDIA Denver2 + quad-core ARM Cortex-A57 CPU, 8GB 128-bit RAM and integrated 256-core Pascal GPU. Jetson TX2 runs on Linux OS and provides a high compute performance with less than 7.5 W of power [22][23].

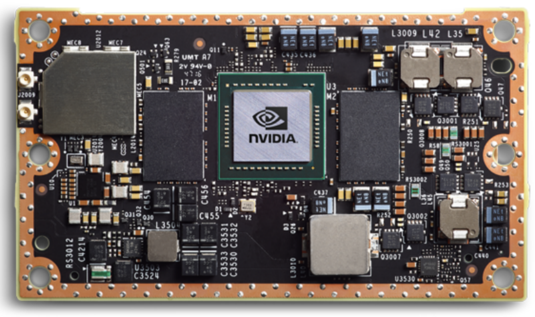


Figure 2.9: Jetson module TX2.

This module is the best high budget option for deploying computer vision and deep learning [23].

2.4 Communication Protocols

In UAV's context, one of the the hardest choices is the communication protocol. There are lots of protocols from which we can choose in order to fulfill the requirements for our system. However, these protocols have different characteristics such as the speed to transfer data, power consumption or even security protocols that they apply.

Table 2.1: Major wireless communication protocols characteristics

Feature	Wi-Fi	Bluetooth	ZigBee	LoRaWAN	4G
Based Data Rate	11Mbps	1Mbps	250kbps	11kbps	100Mbps
Frequency	2,45GHz	2,45GHz	2,45GHz	868MHz	2600MHz
Range	1-100m	10m	10-100m	20km	50-150km*
Security	WPA/WPA2	128 bit	128 bit	128 bit	128 bit

*From a single cell tower [24][25].

From table 2.1 it can be concluded that in order to be able to video stream live images, the only available options would be Wi-Fi, 3G and 4G. However, due to the systems objective to control the UAV remotely from a long range (20-70km) ground control station (GCS), the only viable solution will be the 4G protocol. Therefore, 4G will be the only telecommunication protocol referred in this section.

Since the UAV's communication protocol for media transfer with GCS is already defined, the only protocol left to define is the one used to send and receive commands from the ground station. This protocol is known as MAVLink and will be explained below.

2.4.1 4G

Mobile communications have evolved from the analog mobile radio system the 1st Generation (1G), to the first digital mobile system the 2nd Generation (2G) and finally evolving the digital system to handle broadband data known as 3rd Generation (3G). The latest stable full developed network technology that is used daily by millions of people is the 4G or LTE (Long-Term Evolution). This technology has improved the 3G network with better support for mobile broadband [26].

LTE networks bring fast and higher data rates or bandwidth, and offer packetized data communications. Broadband is the basis of enabling multimedia communications, including video that requires huge amounts of data. Due to the increasing demand of this technology world wide, the charges dropped significantly making it accessible to almost everyone. By being a mobile communication, 4G networks cover almost all the territory making this telecommunication protocol perfect due to its availability. Because its service is based on a ALL-IP network, this makes it ideal for home-networking, live telemetry and sensor-network service [27].

2.4.2 MAVLink Message Protocol

MAVLink (Micro Air Vehicle Link) is a header-only message protocol most commonly used to send data and commands between vehicles and ground stations GCSs and also in the inter-communication of the subsystem of the vehicle. MAVLink was first released in early 2009 by Lorenz Meier under LGPL license. There are two types of MAVLink protocols, MAVLink1 and MAVLink2. MAVLink2 extends all the features of MAVLink1 but allows new fields to be added to existing MAVLink1 messages and supports new types of MAVLink messages [28].

MAVLink messages are identified by the ID field on the packet. This ID is unique for each vehicle and ground station. The payload contains the data from the message. All messages are encrypted with sensor related content, however messages are not guaranteed to be delivered which means the ground stations must often check the state of the UAV to determine if the command was executed. Messages can contain a variety of commands like take off, raise or decrease altitude (throttle increase or decrease, respectively). The messages' size can go from 8 to 263 bytes as shown in figure 2.10 [29][30][31].

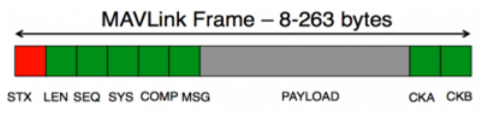


FIGURE 2.10: MAVLink message format (Source: [31]).

2.5 Flight Operating System

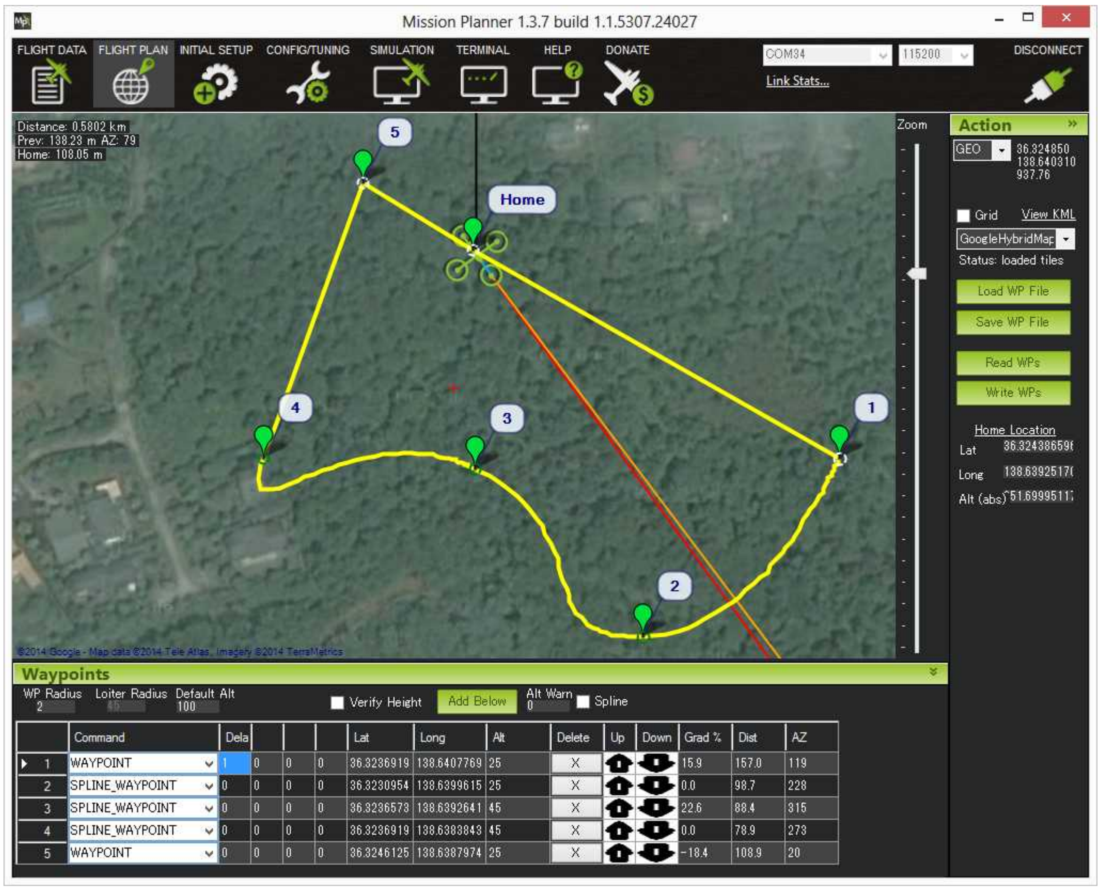
There are lot of already developed flight operating systems, and after a long research the most significant was chosen to be studied.

ArduPilot

ArduPilot is an open source project software suite that consists of a navigation software and aims to enable the creation and use of trusted, autonomous, unmanned vehicle systems. Although *Ardupilot* does not manufacture any type of hardware. Their developed firmware works on many different boards to control all types of unmanned vehicles. This tool for UAV enthusiasts and professionals consists of 3 major features: the hardware compatibility allowing to run *Ardupilot* in different flight controllers, the firmware itself that can control any vehicle required and the software that is the interface to the controller, also known as ground control station. The software can run on many different PCs or mobile devices.

Using inputs from sensors, the flight controller is able to send outputs to electronic components such as ESCs, servos, gimbals, etc. The firmware is the code running on the controller, by matching it to the mission and vehicle type (like a Copter, Plane, Rover, Sub, or Antenna Tracker) it is then ready to execute flight operations.

Coupled with ground control software, vehicles running *Ardupilot* can have advanced functionality including real-time communication with operators. A GCS allows users to set-up, configure, test, and tune the vehicle. Advanced packages allow autonomous mission planning, operation, and post-mission analysis. Software such as "Mission Planner" is one of these GCSs and it is fully compatible with any of *Ardupilot's* firmware. Figure 2.11 shows that it offers point-and-click interaction with the hardware, custom scripting, mission planning and simulation [32][33].



Distance: 0.5802 km
Prev: 138.23 m AZ: 79
Home: 108.05 m

36.324850
138.640310
937.76

Status: loaded tiles

Lat 36.32438659
Long 138.6392517
Alt (abs) 51.6999511

WP Radius 2 Loiter Radius 45 Default Alt 100 Alt Warn 0

	Command	Dela				Lat	Long	Alt	Delete	Up	Down	Grad %	Dist	AZ
1	WAYPOINT	1	0	0	0	36.3236919	138.6407769	25	X			15.9	157.0	119
2	SPLINE_WAYPOINT	0	0	0	0	36.3230954	138.6399615	25	X			0.0	98.7	228
3	SPLINE_WAYPOINT	0	0	0	0	36.3236573	138.6392641	45	X			22.6	88.4	315
4	SPLINE_WAYPOINT	0	0	0	0	36.3236919	138.6383843	45	X			0.0	78.9	273
5	WAYPOINT	0	0	0	0	36.3246125	138.6387974	25	X			-18.4	108.9	20

FIGURE 2.11: Mission Planner user interface.

2.6 Sensors

The sensor system is composed by two major types: internal and external sensors. The internal sensors are all the necessary sensors to the critical system operationality, like the UAV's IMU. The other type of sensors are the external sensors, which are necessary to the rest of the features such as image processing and precision landing.

2.6.1 Internal

The flight controller contains most of the internal sensors. In this case the flight controller in use will be the *Hex Cube Pixhawk 2.1 Flight Controller* which contains the major sensors for flight operations. The sensors are:

- ST Micro L3GD20H 16 bit gyroscope
- ST Micro LSM303D 14 bit accelerometer / magnetometer
- MPU-9250 Nine-Axis (Gyro + Accelerometer + Compass) MEMS Motion-Tracking Device
- MEAS MS5611 barometer

ST Micro L3GD20H is a 16 bit low-power three-axis angular rate sensor. This sensor is a device used for measuring or maintaining the orientation and angular velocity of the UAV. With help from software this sensor helps the UAV maintain its stability while in autopilot as well as manual mode, insuring that the propellers compensate each other and the UAV's angle stays horizontal [34][35][36].

ST Micro LSM303D is a 14 bit accelerometer / magnetometer. This sensor contains 3 magnetic field channels and 3 acceleration channels. The magnetometer or smart digital magnetometer is a 3-axis smart digital magnetometer that detects the strength and direction of a magnetic field and communicates the x, y and z components. In UAV context it is the UAV's own compass. The accelerometer is used to measure the acceleration of an UAV. It works by sensing the acceleration of gravity using the same technology which is also behind gyroscopes [37][38][39][40][41].

MPU-9250 is a multi-chip module (MCM) features three 16-bit analog-to-digital converters (ADCs) for digitizing the gyroscope outputs, three 16-bit ADCs for digitizing the accelerometer outputs, and three 16-bit ADCs for digitizing the magnetometer outputs. MPU-9250 is also designed to interface with multiple non-inertial digital sensors, such as pressure sensors, on its auxiliary I^2C port. It is used to precisely and accurately track the user both in fast and slow motions. This is the main accelerometer and gyroscope. All the others offer higher accuracy [42].

MEAS MS5611 is a barometer, which measures the air pressure. This sensor is optimized for altimeters and variometers with an altitude resolution of 10 cm. It provides pressure and temperature values. In UAV context it provides essential information of the UAV's altitude [43][44].

2.6.2 External

The external sensors are the ones that are not necessary to make the UAV fly, but instead are all the others that are needed to execute the designed features (being the other features the fire detection algorithm and the autonomous landing procedure). The only extra sensor necessary for these operations is the visual camera and the GNSS sensor.

2.6.2.1 Camera

There are lots of different types of cameras for lots of different applications. In UAV context, the visual cameras can be applied to photography, filming, to do city surveillance, agriculture or even war zone strikes. For environmental purposes, more precisely for forest surveillance, a variety of camera types can be used. The best types to be considered for fire detection algorithms are multispectral and thermographic or infrared cameras.

TABLE 2.2: Camera types.

Specifications	Thermal (FLIR Duo Pro R)	Multispectral (RedEdge-M)
Dimensions (mm)	87 x 82 x 69	94 x 63 x 46
Weight (g)	325 - 375	173
Spectral Bands (nm)	7500 - 13500	20 - 400
Power (W)	10	8
Field of View (°)	56	47.2

Multispectral camera – This type of camera captures image data within specific wavelength ranges across the electromagnetic spectrum. These wavelengths being so specific, they can be separated from each other by using filters or other type of instruments that are sensitive to particular wavelength. Even light from the beyond the visible range, like infrared and ultra-violet, can be detected and analyzed. With such a detailed analysis a lot of information can be extracted from the images, information that could not be detected by our eye sight like red, green and blue levels. Although it seems as a viable option for fire detection, in case of smoke surrounding the UAV, this analysis will fail [45][46].

Thermal camera – This type of cameras reads the Infrared spectrum to detect heat coming from all surfaces and objects. By capturing the heat waves coming from the objects, it creates images and videos. From the analysis of those images, a fire can easily be spotted due to the high temperatures it emits, making this type of camera suitable for forest surveillance [47].

From both types of cameras presented above we can conclude that the most reliable and efficient camera to be used in this case is the thermal camera. However due to the nature of the project, there is no reliable way of gathering enough data to build a dataset for the analysed cameras. The camera to use in this project will be a simple one, compatible with the Raspberry Pi, the Raspberry Pi Camera Module v2.

The Raspberry Pi Camera Module v2 (figure 2.12) features an ultra-high quality 8 megapixel Sony IMX219 image sensor, and a fixed focus camera lens. The module is capable of 3280 x 2464 pixel static images, and also supports 1920 x 1080, 1280 x 720 and 640 x 480 video resolutions. It can be attached to the Raspberry Pi using a 15 Pin Ribbon Cable, to the dedicated 15-pin MIPI Camera Serial Interface (CSI), which was designed especially for interfacing with cameras. The CSI bus is capable of extremely high data rates, and it exclusively carries pixel data to the BCM2835 processor. The board itself is tiny, at around 25mm x 24mm x 9mm, and weighs just over 3g, making it perfect for UAV applications where size and weight are important [48].

FIGURE 2.12: Raspberry Pi camera module v2.

2.6.2.2 GNSS sensor

For UAV navigation, a GNSS sensor was used. It's considered to be an internal sensor, but in the *Hex Cube Pixhawk 2.1* Flight Controller is considered to be an external sensor. It is one of the most important sensors in order to keep track of the UAV and plan the desired mission.

The Global Positioning System (GPS) is the most globally known GNSS constellation. Also know as GPS, is a radio-navigation based system owned and developed by the United States government. It provides geolocation data, such as latitude, longitude, altitude, and time information to the receiving GNSS sensor anywhere on Earth. Obstacles such as mountains and buildings cause significant GPS signals losses. In UAV's there is a lot of concern regarding this matter because once the GPS signal is lost, UAVs may actually crash because they lose the ability to react. However, some measures will be presented later in order to prevent such disasters [49].

The model used in this project will be the HERE2 GNSS GPS (figure 2.13). With a built-in STM32F302 microprocessor. It runs ChibiOS (a real-time operating system), which enables user-defined features. HERE2 supports standard serial port + I2C transmission data and also supports CAN bus on the way. CAN is the most commonly used communication protocol in the automotive industry, and has many advantages such as strong real-time data communication, high reliability, flexible application, and redundant structure. CAN allows for a distributed Autopilot system by placing components at their optimal position on an airframe without worrying about signal degradation. HERE2 utilizes the UAVCAN bus protocol specifically developed for UAV applications. UAVCAN has high data throughput, low latency and is suitable for application scenarios with high real-time requirements. A complete IMU including an accelerometer, compass and gyroscope can meet the diverse needs of the user for navigation. Combining barometer and GPS data, users could potentially run a separate Extended Kalman Filter (EKF) navigation system inside HERE2 to implement navigation tasks that are completely independent of flight control [50][51].

2.7 UAV Automatic Procedures

To design a fully automated procedure, the UAV has to be able to follow a flow of operations that complete the system's operability. From the mission planning to the in-flight operations and the automatic landing, they all complete each other making all the UAV operations fully automated.

FIGURE 2.13: HERE2 GNSS GPS module.

2.7.1 Mission Planning

Due to *Ardupilot*, creating mission scripts became relatively easy. Using python programming language and state of the art compilers, simple commands like take-off, land or go to waypoint (intermediate point on a route) became accessible with a simple line of code. Mission planning will consist on creating a python script to automate the UAV to takeoff, go to the waypoint, run the fire prediction algorithm and, if nothing is detected, return to base. All these procedures will be explained below.

2.7.2 Decision Making

The behaviour/procedures of the UAV need to be set in by order to make mission execution flawless and complete. The takeoff requirements are all the necessary diagnostics to make sure the mission is completed with success. The flight procedures are the operating decisions the UAV needs to execute the mission while on the air.

The take-off requirements shown in figure 2.14 make sure that the system and all the sensors are working properly and if there is enough battery to proceed with the mission. This is the first task: making sure the battery levels are full for take off, otherwise the UAV will stay at the base recharging. After running the battery levels and performing the full diagnostic, the UAV will be ready for take-off.

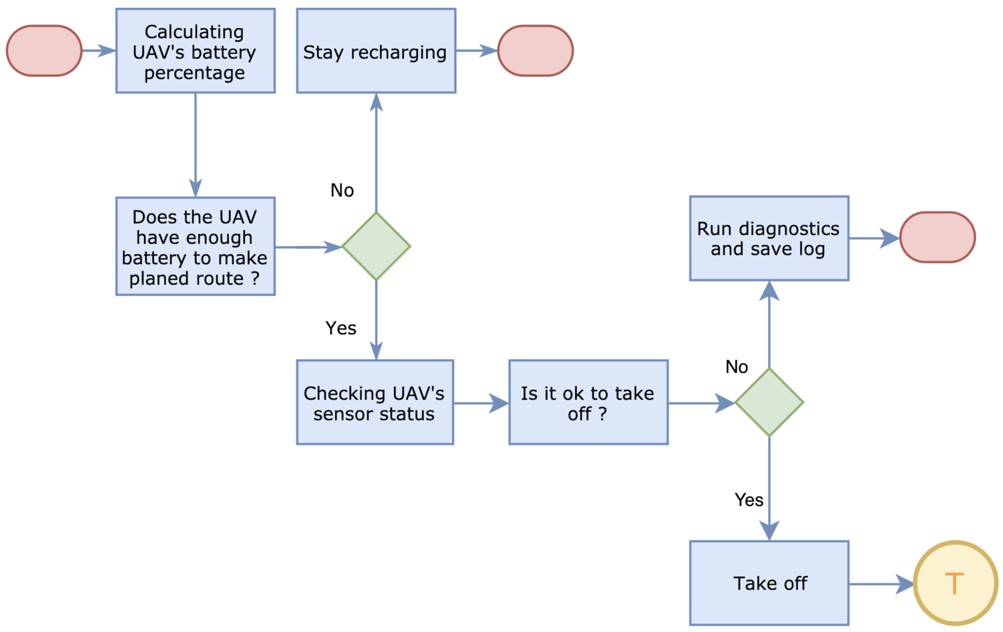

Calculating UAV's battery percentage
Stay recharging
No
Does the UAV have enough battery to make planed route ?
Yes
Run diagnostics and save log
No
Checking UAV's sensor status
Is it ok to take off ?
Yes
Take off
T

FIGURE 2.14: UAV's procedure for take-off diagram.

The flight procedures shown in figure 2.15 begin after take-off, where the UAV will fly to the designated coordinate and take the sensors' data to run the fire detection algorithm. Afterwards, in case of a fire detected, it will notify the user and return to base, otherwise it will check if it is the last coordinate and if not, it will continue to carry out the rest of the mission. When the last coordinate is reached and no fire has been detected, the UAV will return to its base to recharge and wait to execute its next mission.

2.8 Fire Detection Algorithm

To detect a fire in the forest, the surveillance range needs to bee quite high. To cover such space, the information collected from each position needs to be wide. The only way such a system would work would be using an UAV at a high altitude to gather a large amount of information at once. The most effective way to gather that much data would be using a sensor that could cover a hide area, taking us to

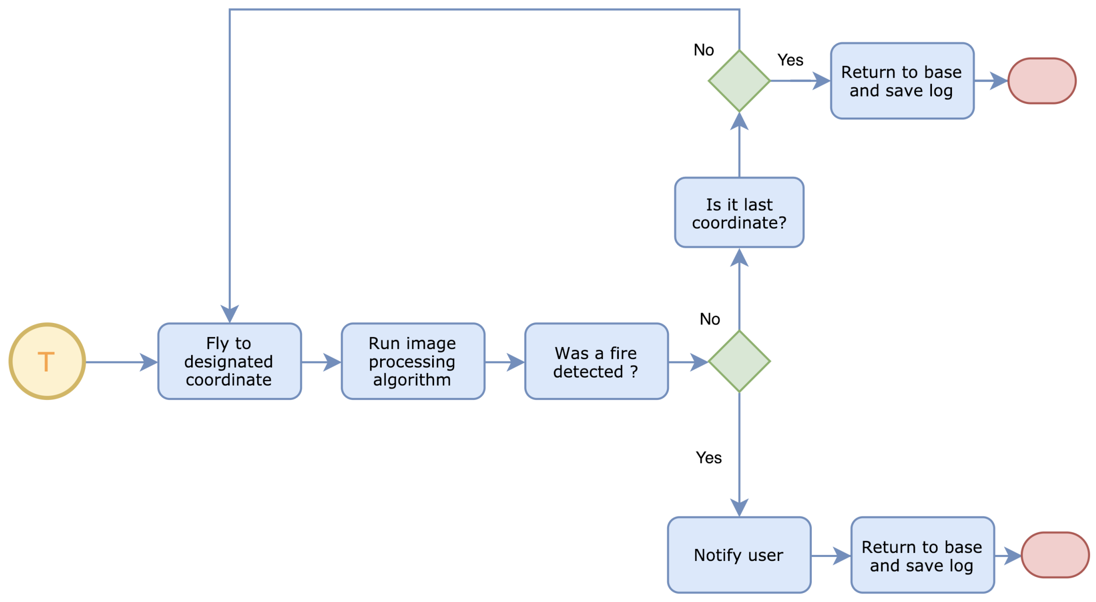

No
Yes
Return to base and save log
Is it last coordinate?
No
T
Fly to designated coordinate
Run image processing algorithm
Was a fire detected ?
Yes
Notify user
Return to base and save log

FIGURE 2.15: UAV's flight procedure diagram.

the visual camera. By analyzing image frames for a few seconds, the amount of information needed will be enough to determine if a fire exists or not. The process of analyzing images is called image processing.

Digital Image Processing

Digital image processing is about processing images to a digital format with a digital computer. It involves digital image analysis, which is being able to retrieve information from the pictures, reading it and gather it. In many cases, digital image processing tends to replicate human vision functions.

There are 3 levels of digital image processing: low-level, intermediate-level and high-level. Low-level image processing algorithms are simple algorithms that receive an image as input and then output the same image with the desired changes. Intermediate-level algorithms are algorithms that also have as input images but as output may have a variety of image related features such as object contours. High-level algorithms are algorithms that use symbolic representation for both input and output. common applications are: object detection and classification such as the ones that will be used in this project [52].

2.9 Related Work

Nowadays the approach to forest fire detection is based on manned aerial vehicle, ground-based equipment, or satellite surveillance [53]. However, this solution is expensive and makes use of a lot of human resources, threatening many human lives. Therefore, the use of small UAVs for fire detection is an obvious and reliable option [54]. UAVs can be extremely useful, providing wide area coverage even with cloudy weather, working day and night providing 24 hour coverage and in case of incidents, they are easy to recover and relatively inexpensive. In case of battery-powered UAVs, they can also be more environmentally friendly and can use energy sources like solar panels [55], [56], and most importantly, they can execute missions autonomously without the need for human pilots or operators.

Vision based solutions have big advantages and provide real-time information. They have a wide detection range and have become the key component in the UAVs forest fire detection system applications [57]. During the last years, vision based solutions for forest fire detection have used a lot of different segmentation methods. Vision based fire detection techniques typically make use of three dominant features: color motion and geometry [58]. The color is the more relevant visual feature of fire and it is why most methods take advantage of the discriminative properties in color space to obtain fire regions in the images [59], [60]. Some researchers used color and motion features based on red-green-blue (RGB) model to extract real fire and smoke in video sequences [61]. Others proposed real-time algorithm that combines motion and color clues with fire flicker analysis on wavelength domain to detect fire in video sequences [62]. In [63] the fire detection system is a combination of a generic color model based on RGB color space, motion information, and Markov process enhanced fire flicker analysis. In [64], a different approach is used making use of a rule based generic color model for flame pixel classification. Experimental results show that the detection performance is significantly improved. To detect both fire and smoke areas together, a solution based on video image segmentation is presented in [65] [66].

In order to be able to recharge the UAV autonomously, the concept of recharging the UAV during the mission has been about increasing flight time and providing the ability for longer missions [67],[68]. At the University of Southern California, a ground robot that can autonomously recharge on a stationary “charging dock” was developed [69]. Another similar application is the development of a battery swapping station for small coaxial helicopters [70]. For ground robots there are already some successful applications, however for UAV a mainstream application has not yet been successfully deployed. Researchers at the Massachusetts Institute of Technology [67],[68] and ETH Zurich [71] have been working towards persistence missions with autonomous recharging stations for UAVs, nonetheless all these works use a contact based charging station. The negative side of these applications is that they require a high precision landing without margin for error and its mechanical design and implementation increases the manufacturing cost and the design complexity [72].

To solve this problem, a technology called Wireless Power Transfer (WPT) is proposed. WPT is a famous technique being used in many low-power charging applications for smart-phones and other electronic devices [73],[74]. Nowadays, the strongly coupled magnetic resonant induction for WPT allows efficient and high power transmission to distances up to 2 m [73]. Magnetic resonant induction-based WPT has little interference and disturbance with its environment, and it is omnidirectional characteristic motivated for the application to UAVs [74],[75]. A more recent approach made by [76] reveals a affordable solution to increase flight time for outdoor UAVs using a wireless charging station without any human intervention.

Chapter 3

UAV Flight Automation

3.1 Pre-Flight Procedures

As referred in section 2.4.2, MAVLink can be used not only to send messages between the vehicle and the ground station but also in the inter-communication between vehicle subsystems. Using this principle, MAVLink commands can be used to make an UAV land or take-off by itself without having to use a Radio Controlled (RC) transmitter. A large variety of methods can be created, using different programming languages that will send multiple commands and wait for an acknowledge response in return. By creating different methods and combining them, coding libraries can be composed and published, being accessible for everyone to use, in order to facilitate the mission's automation. For this project, two python libraries known as Dronekit and Pymavlink will be used.

3.1.1 Vehicle Connection

To establish the program's connection to the UAV, Ardupilot already has a default User Datagram Protocol (UDP) port defined (14550). This port is always listening on a standby status waiting for commands. When creating the script for the UAV's automation, the connection command needs to be structured with the correspondent arguments. The *argparse* library is a helpful tool to use for this purpose as it helps adding the connection arguments, such as the default port.

```
parser.add_argument('--connect', default='127.0.0.1:14550')
```

Afterwards, the connection method is used to send the connection arguments, the communication speed and another parameter "wait-ready", that defines if the program waits for for an acknowledge or proceeds without it.

```
vehicle = connect(args.connect, baud=57600, wait_ready=True)
```

3.1.2 Pre-Flight System Analysis

The pre-flight system analysis is the UAV's self analysis that determines if the UAV is ready to run automated missions or not. It will log all the system's parameters before flight and check the status of some of them such as:

- Battery status
- Vehicle EKF
- Vehicle armable state
- Vehicle system status state

First of all, the battery level is checked since it must be at 100% in order for the UAV to start the mission, otherwise the procedure will abort and the program will disconnect from the UAV correctly. Then it will check the EKF algorithm, which is used to estimate vehicle position, velocity and angular orientation based on rate gyroscopes, accelerometer and other sensors. The program will wait until all the sensors used by the EKF are ready and check every 2 seconds if they are not. After confirming if all the previous parameters are correct, the last check before arming the UAV is to verify the vehicle's state and the system's status. If the system's status returned variable equals "STANDBY" that means the UAV is grounded on standby, and it can be launched at anytime. However the system status verification can return other states such as:

- UNINIT - System not initialized and the state is unknown.
- BOOT - System is booting up;
- CALIBRATING - System is calibrating and is not flight-ready;
- ACTIVE - System is active and might be already airborne. Motors are engaged;
- CRITICAL - System is in a non-normal flight mode but it can still navigate;
- EMERGENCY - System is in a non-normal flight mode and it has lost control over parts or over the whole airframe. It is in mayday and going down;
- POWEROFF - System just initialized its power-down sequence and will shut down [77].

The full pre-flight system analysis depicted in figure 3.1 take a small amount of time to fully execute. However, they are crucial and extremely important for mission success.

After logging all the parameters and making sure the vehicle is ready to fly, the take-off sequence begins.

3.1.3 Take-off

When dealing with UAVs and other types of air vehicles, the motors must be *armed* in order to initialize take-off. In the present study scenario, arming motors means the UAV is ready to fly and the motors can then spin when one applies *throttle*. What is often called throttle (in an aviation context) is also called a thrust lever and it is used to control the thrust output of the aircraft's engines [78][79].

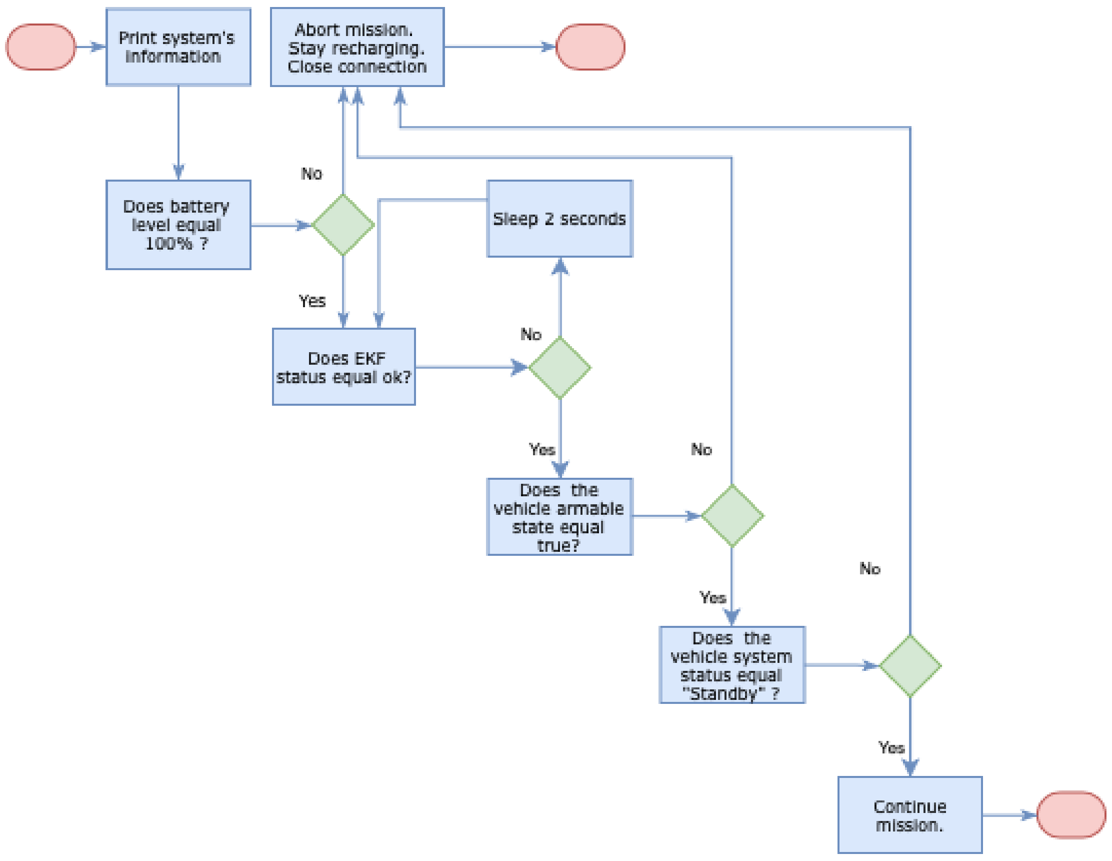

Print system's information
Abort mission. Stay recharging. Close connection
Does battery level equal 100% ?
No
Yes
Sleep 2 seconds
Does EKF status equal ok?
No
Yes
Does the vehicle armable state equal true?
No
Yes
Does the vehicle system status equal "Standby" ?
No
Yes
Continue mission.

FIGURE 3.1: Pre-flight system analysis diagram.

In Ardupilot, there are different flight modes that can be used according to the type of mission to be executed. These flight modes can be different on the various Ardupilot firmwares (p.ex, ArduCopter flight modes are different from ArduRover) In this case, and since ArduCopter firmware is used, "GUIDED" flight mode will be used because it is the adequate mode to navigate to single points autonomously. The mode selection is required, since in order to arm the vehicle a flight mode must be defined.

After defining the flight mode, the command to arm the vehicle is sent and the program waits for the acknowledge in order to use the take-off command. The take-off requires the altitude input and in case this input is different from a number, the program will print the following message "Altitude was NaN or Infinity. Please provide a real number" and end the program. When the right input is correctly given, the UAV will takeoff and print with 1 second intervals the current altitude, until it reaches 95% of the desired altitude. At this point, the

"Reached target altitude" message will be presented and the UAV will wait for following instructions [77].

3.2 Mission Planning

To plan an autonomous mission for UAVs, several variables need to be taken into account. In order to define those variables a logical flow of events was established:

- Define Flight Duration;
- Define Total Coverage;
- Define a Route;
- Add Route to Script.

3.2.1 Defining Flight Duration

Using the average case scenario, a multi-rotor's flight time will be about 10 minutes. With a flight speed of about 5 m/s (18 km/h) the maximum mission distance the UAV can achieve will be $5 \cdot 60 \cdot 10 = 3000$ m, per flight. However, the UAV has to do pre-flight procedures, has to take-off, has to land and after reaching a waypoint, it also has to wait for 10 seconds for the image processing algorithm to run, thus leaving approximately 4:54 minutes for continuous flight time (this values where taken in consideration according to simulation experiments). The need for the UAV to hover above each waypoint for 10 seconds is to increase the image processing algorithm's accuracy, and also to allow saving battery - the image processing algorithm consumes a lot of the companion's computer resources increasing battery drain.

3.2.2 Defining Total Coverage

The UAV's total coverage depends on three factors: flight duration, which is already defined, camera specifications and mission's altitude. According to the Portuguese legislation, an UAV can fly up to 120 m high without having to ask for special permission from the Portuguese Civil Aviation Authority (ANAC) [80]. Flying at an altitude of 120 m and using the Raspberry Pi camera module v2 as the main camera, it is possible to calculate the amount of ground coverage per picture. In these calculations, the knowledge of Photogrammetry is needed, which is the science of taking measurements from photographs. The first step is to calculate the Ground Sampling Distance (GSD), the distance between two consecutive pixel centers measured on the ground [81][82][48]. The GSD value is given by:

$$GSD = \frac{Sw \times H}{FR \times imW} \tag{3.1}$$

where Sw is the sensor width of the camera (in millimeters), FR the focal length of the camera (in millimeters), H the flight's height (in meters) and imW the image width (in pixels). The camera specifications such as focal length and sensor's width can be found in appendix C. Considering the respective values the following equation is obtained:

$$GSD(cm/px) = \frac{6.35 \times 120 \times 100}{3.04 \times 1920} \tag{3.2}$$

This way, the GSD will be equal to 13.06 (centimeter/pixel). The second step is to calculate the ground distance that can be seen from the picture using:

$$Dw(m) = \frac{GSD \times imW}{100} \tag{3.3}$$

$$DH(m) = \frac{GSD \times imH}{100} \tag{3.4}$$

where Dw is the footprint width, which is the distance covered on the ground by one image in width direction, DH the footprint height, which is the distance covered on the ground by one image in height direction and imH is the image height in pixels. The relation between the previous parameters can be seen in figure 3.2.

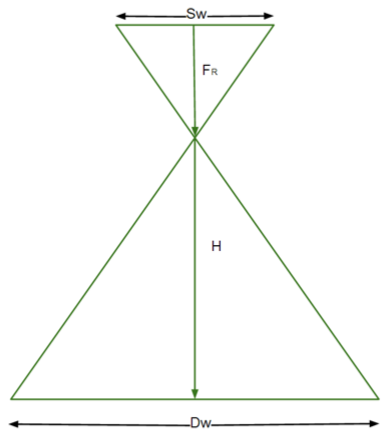


FIGURE 3.2: Representation of the camera calculations.

The real distance in meters of each picture is: $Dw = 251$ m and $DH = 141$ m. This means one picture will cover a total area of 35391 m^2 $\approx$ 3.54 hectares. With the picture size calculated, the easiest approach to define the total area coverage is to define the number of pictures to take according to the maximum distance the UAV can travel. In section 3.2.1, it has concluded that 4:54 minutes was the maximum continuous flight time of the UAV, if the UAV's velocity is multiplied with the flight time, that leaves exactly $5 \cdot ((60 \cdot 4) + 0.54) = 1470$ m maximum distance per flight. By simulating a combination of different number of pictures, arranged in different ways, with different routes, the result of 8 pictures was achieved. To scan this area, the UAV has to fly over $141 \cdot 6 + 251 \cdot 2 = 1348$ m, therefore passing the maximum distance per flight requirement. The pictures disposition can be seen in figure 3.3.

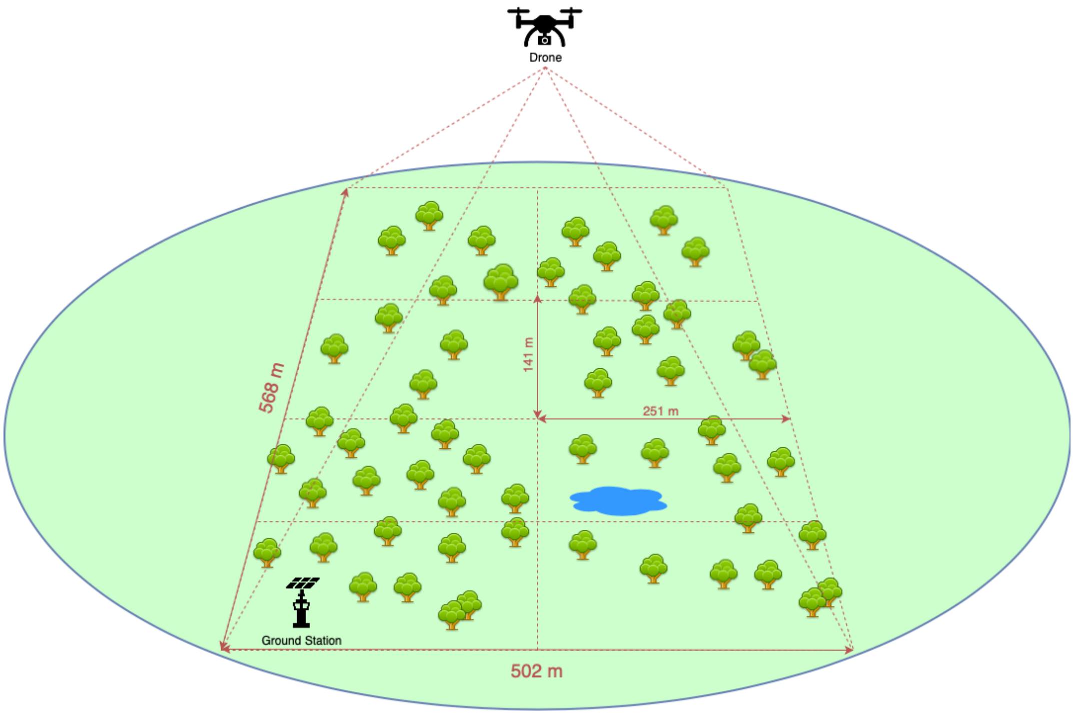


FIGURE 3.3: Mission planning diagram.

3.2.3 Defining a Route

In order to plan the route for the UAV to autonomously follow, a software simulator was used (explained in subsection 6.1). Different types of routes were taken into consideration and the resulting route was also the most efficient and battery saving option. With the altitude and the number of smaller areas (pictures) defined, the mission can now be planned using the middle of each smaller area as a waypoint. By using 8 different waypoints and the ground station as the home waypoint, the final route can now be calculated for the UAV to follow.

When dealing with UAVs flying in an autonomous mode, special attention to the UAV's orientation is required. Since the camera does not move, the perspective of the picture depends on which side the UAV is facing. Whenever the UAV is flying and has to change direction to go to the next waypoint, firstly it rotates facing forward to the waypoint, and only then it starts moving in the desired direction. For example, taking figure 3.4 and imagining the UAV needs to go from waypoint 4 to 5. When it reaches the fifth waypoint, the front of the UAV will

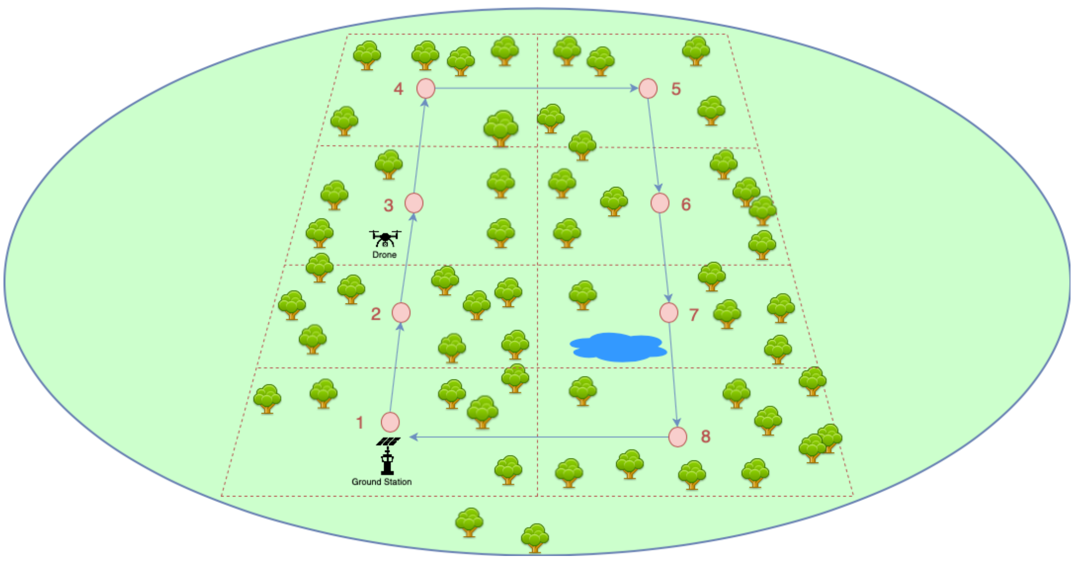


FIGURE 3.4: Initial waypoints position diagram.

be faced towards the direction of the followed route instead of facing forward to the route it has to follow next. Thus resulting in the image of the fifth waypoint having to be rotated in a 90° angle, which will reduce the covered area. In order to avoid this situation, two more waypoints were added: one before waypoint 5 facing north and another before waypoint 1 facing south because when the UAV travels from waypoint 8 to 1 the same situation will occur. This extra waypoint before waypoint 1 will be the home waypoint from where the UAV will take-off and land. This solution will solve the previous problem, maintaining the image aspect ratio with a 180° angle rotation from the previous images. The extra waypoints will only differ in a five meter distance from the desired waypoint. The final route waypoints are illustrated in figure 3.5.

3.2.4 Adding Route to Script

After planning the route, each waypoint must now be defined in the program and sent individually to the flight controller as a MAVLink command. To begin, a function is used to get the global position of the ten final waypoints positioned to form the rectangular surveillance area. Using the size of the rectangle as the

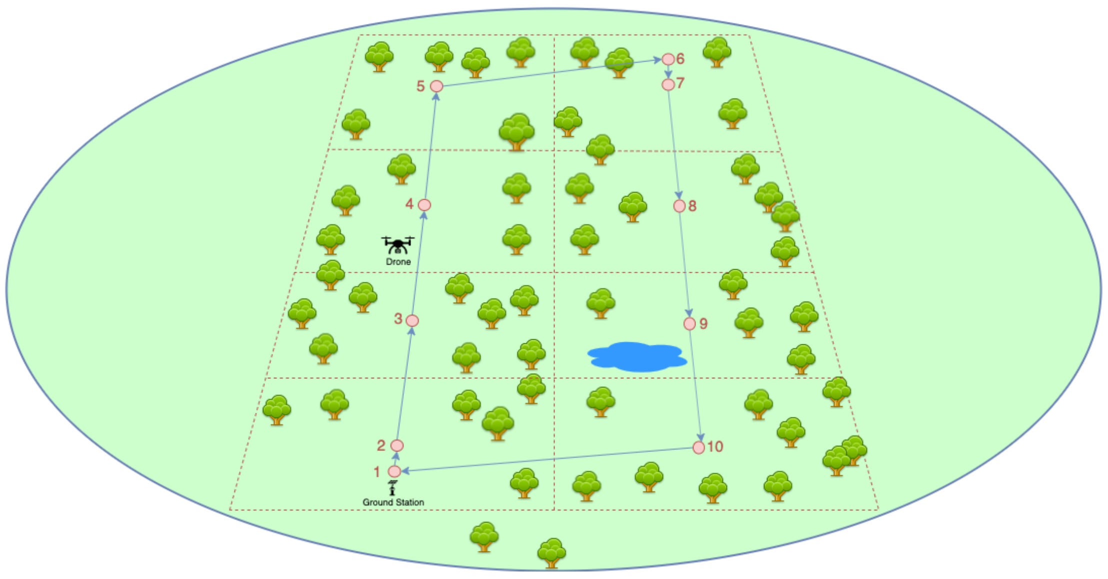


FIGURE 3.5: Final waypoints position diagram.

only input, a route can be created. The 10 waypoints will have the following coordinates:

TABLE 3.1: Waypoints coordinates

Point ID	x (m)	y (m)
1	0	0
2	5	0
3	$5 + a$	0
4	$5 + 2 \cdot a$	0
5	$5 + 3 \cdot a$	0
6	$10 + 3 \cdot a$	0
7	$5 + 3 \cdot a$	b
8	$5 + 2 \cdot a$	b
9	$5 + a$	b
10	5	b

where x is the distance from the ground station to north, y the distance from the ground station to east, a the height of the rectangle, b the width of the rectangle.

If for different purposes the same route has to be adjusted to different rectangle sizes the only parameters that would need changing would be the picture's height, and width. The waypoint's altitude will be the same as the designated altitude command on takeoff.

Once the waypoint's coordinates are calculated, they can now be added to the commands that are to be sent to the UAV. A dummy waypoint is also added to notify the user when the last destination has been reached.

As a safety measure, all commands are cleared so as to empty the command queue before the take-off command is sent. This ensures that, if any of the previous commands were given, they will be ignored. In case the UAV does not receive the previous take-off command, a fail safe is implemented to make sure the UAV proceeds with the take-off anyway. The *go to* command will be similar to the code right below in all of the ten waypoints.

```
cmds.add(Command( 0, 0, 0,mavutil.mavlink.MAV_FRAME_GLOBAL_RELATIVE_ALT
,mavutil.mavlink.MAV_CMD_NAV_WAYPOINT, 0, 0, 0, delay, 0, 0, point1.lat
, point1.lon, altitude))
```

After adding all the commands with the waypoints to the queue, the mission is uploaded to the UAV. In order to keep track of the mission's status along the waypoints, two functions are created: one to give the ground distance, in meters, between two locations and another to get the global position between the current location and the next waypoint. By using these two functions together the program can print every 1 second the distance to the next waypoint, thus keeping track of the current mission status. At the end of the mission, in order to make the UAV return to the base, the previously referred dummy waypoint is used. By creating a simple "if" statement, the next waypoint is checked and if it corresponds to the last one, the mission is terminated. When the mission ends, a command is sent to the UAV to switch to the Return To Launch (RTL) mode that makes the UAV return to the home position and stay hovering. If no altitude is set in the RTL command the UAV will reach the DroneKit's library default altitude (15 m), or maintain the current altitude if the current altitude is higher than the default. After getting to the home position, the UAV will start to descend until it reaches the altitude and location where it was armed [83].

Following the designated route, the mission will have a total distance of $141 \cdot 6 + 251 \cdot 2 + 5 \cdot 2 = 1358$ m, and the total flight time will depended on the type of

UAV and flight speed. However having by default the previous mentioned values the mission duration will be 9:07 minutes.

3.3 Flight Altitude vs Processing Time

To demonstrate the efficiency of the chosen altitude, a study was conducted to show how many hectares can be processed in 1 hour, using different altitudes. In order to perform the study, some of the calculations made in the previous section 3.2.2 were applied, changing the ground captured area and the altitude. To obtain more accurate results, simulations were made each 10 m of altitude. Repeating the previous calculations for 10, 20, 30, 40, 50, 60, 70, 80, 90, 100 and 110 m, the following values were calculated for the distance covered on the ground by one image's height and width.

Table 3.2: Relation between UAV's altitude and picture real dimensions.

A (m)	Dw (m)	DH (m)
10	21	12
20	42	23
30	62	35
40	84	47
50	104	59
60	125	70
70	146	82
80	167	94
90	188	106
100	209	117
110	230	129

where A is the UAV's altitude (m). When the distance covered on the ground by one image changes, the total ground area covered per flight and the total flight time will also change. So to obtain the number of hectares processed per hour, the following formula was used:

$$N = \frac{Total\ area\ covered\ (hectares)}{Total\ flight\ time\ (hours)} \tag{3.5}$$

where N is number of hectares processed per hour. Using the previous values of altitude, the resulting chart demonstrates the number of hectares processed per hour for each altitude.

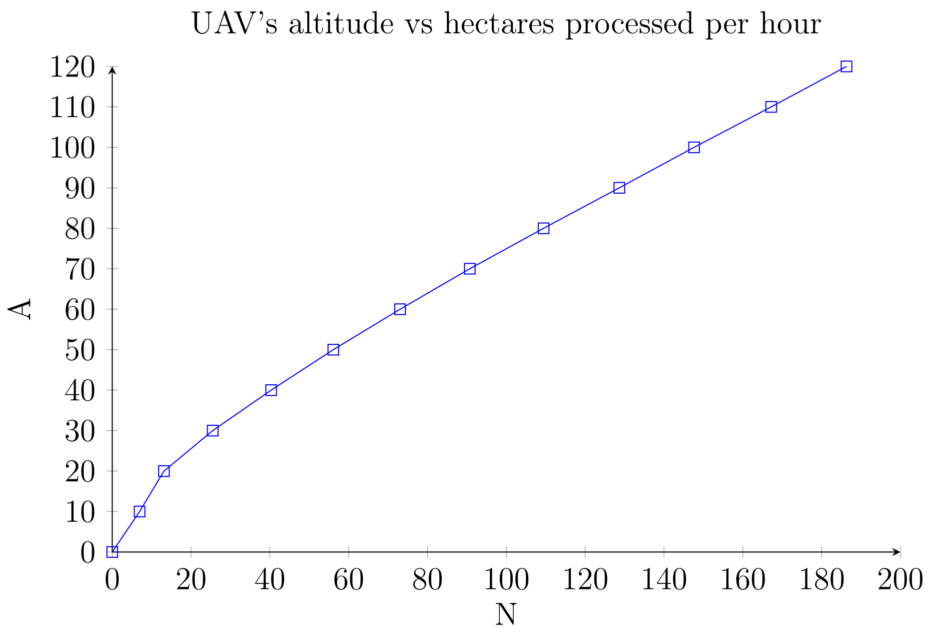


FIGURE 3.6: UAV's altitude vs hectares processed per hour

In conclusion, the advantage of using the maximum altitude allowed (120 m) it is noticeable and clearly the best available option.

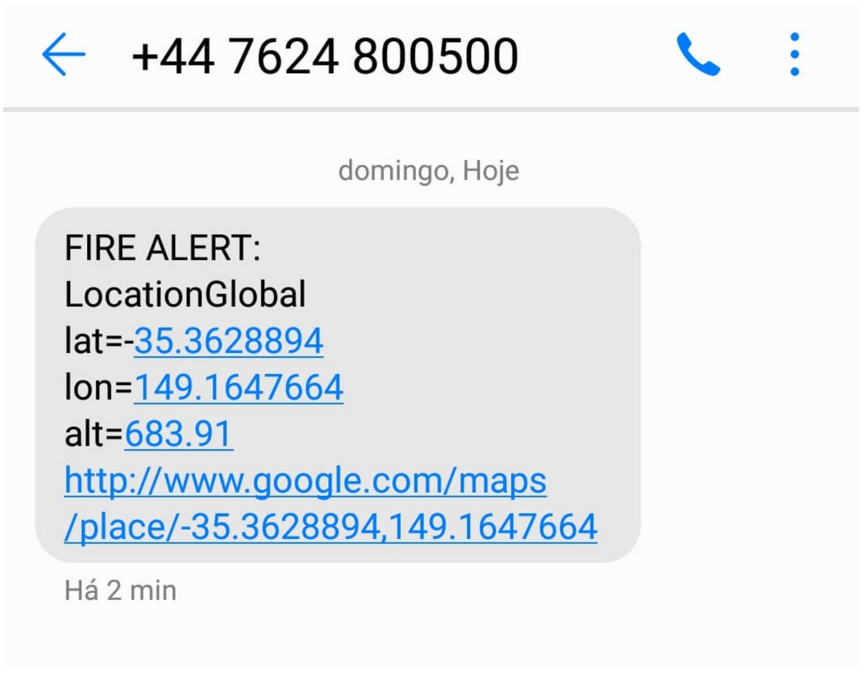
3.4 Decision Making and User Notification

In this system, User Notification is a fundamental aspect and should be taken into serious consideration. Nowadays, notifications can be sent in the form of emails, text messages, phone calls or others. For someone who is responsible for the surveillance of forests and other natural preserved areas getting cell phone signal can sometimes be challenging since cell towers may be located along great distances. With such distances between the towers and the user's phone and all the obstacles along the way, any chances of notification systems using internet are unreliable, therefore the only two options would be phone calls and text messages. However, if the signal strength is low the phone call will be hard to understand, thus making text messages the most reliable notification system to use. Even though it appears to be the best communication system, one should bear in mind that phone calls and text messages are paid services that can only be used by paying a specific fee to a national telecommunications service provider. Picking up on the chosen solution the service used will be TextMagic which is a business text-messaging service for sending notifications, alerts, reminders, confirmations and SMS marketing campaigns [84]. By using TextMagic, creating text messages will be a simple task, since this solution already integrates a python library known as TextMagic library. To create a text message the steps are: authenticate the user (with security token), specify the destination phone number and fill the message content, as shown below.

```
message = client.messages.create(phones="91*******",
text="FIRE ALERT:" + "\n" + str(vehicle.location.global_frame))
```

Due to the variety of functionalities inside DroneKit python library, getting the global location of the UAV is about finding the right method. In this case, the method "vehicle.location.global_frame". By using this method, an object is obtained containing the current position (with latitude, longitude and altitude values) of the UAV. Finally, the object is deconstructed and written in a way which is easy to read and copy the content. If the user has access to the internet he can also take advantage of the Google Maps URL which is also obtained from the deconstruction of the global position object. An example of the fire alert notification message can be seen in figure 3.7.

The only step remaining in the notification system, is when to notify the user. This final step is achieved by adding a trigger to the image processing algorithm which is activated every time a fire is detected.



domingo, Hoje

FIRE ALERT:
LocationGlobal
lat=-35.3628894
lon=149.1647664
alt=683.91
http://www.google.com/maps/place/-35.3628894,149.1647664

Há 2 min

FIGURE 3.7: Fire alert notification message.

Chapter 4

Fire Detection Algorithm and Real-Time Detection

4.1 Image Recognition and Object Detection

Nowadays image recognition and object detection are considered basic concepts, with self driving cars and image processing technologies. However, there is a great difference between the two of them: in image recognition, the image is received as an input and the output is the class label to which the image belongs (ex: car, bike, dog, etc...). Object detection also receives an image as input, but the output is the combining result of the class label to which the image belongs and image localization (the location of the object). Object detection not only identifies the object class but also locates the object, surrounding it with a bounding box [85]. In order to be able to detect these objects and classify them, machine learning algorithms, such as convolutional neural networks can be used even though it requires training. For this purpose, tools such as *Tensorflow* was used.

4.1.1 TensorFlow

TensorFlow is an open-source interface for experimenting and implementing machine learning algorithms. It can be executed in a large variety of heterogeneous systems with little or no change at all. This system can be used from mobile devices such as phones or tablets to large-scale distributed systems with hundreds of machines and thousands of computational devices such as CPU and GPU cards. TensorFlow is known for its flexibility because it can be used to express a wide variety of algorithms including training algorithms for deep neural network models. This system is owned by Google and it is the result of the Google Brain project that started in 2011 to evolve the knowledge of very-large-scale deep neural networks, not only to use in other Google products but also for research [86].

TensorFlow predecessor was DistBelief, the first-generation scalable distributed training and inference system. DistBelief was used for unsupervised learning, language representation, image classification and object detection, video classification, speech recognition and many others. When compared to DistBelief, TensorFlow's programming module is more flexible, its performance is better, it supports training and uses a wider range of models in a great variety of heterogeneous hardware platforms.

TensorFlow uses a structure known as data flow graphs to represent computation. A data flow graph has two basic units: a node that represents a mathematical operation and an edge representing a multi-dimensional array, known as a tensor. This high-level abstraction reveals how the data flows between operations and can be easily represented in figures, such as in figure 4.1 where the *b, w, x* represent the tensors, *c* represents the output and the other elements represent the nodes. Users typically create a computational graph using one of the supported languages: Python or C++.

To interact with TensorFlow, the applications created by the users create a *Session.* In order to create a computation graph, the Session interface supports an

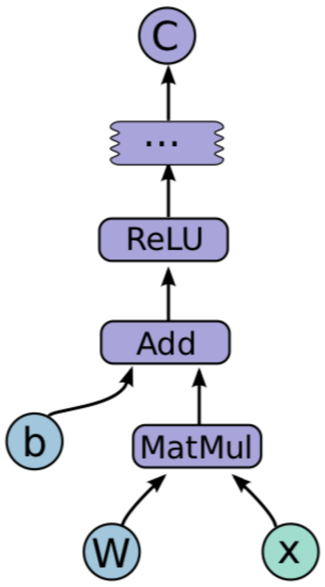


FIGURE 4.1: TensorFlow graph (Source: [87]).

extended method to augment the current graph managed by the session with additional nodes and edges. The other extended operation supported by the session is *Run*. This operation takes a set of output names that need to be computed, as well as an optional set of tensors to be fed into the graph in place of certain outputs of nodes. After setting up a Session with a graph once, the Run call can be used to execute a graph or subgraphs as many times as the users wants. Because TensorFlow graphs can run multiple times, the only way to save the tensors across executions is to use *Variables*. These variables are a special kind of operation that returns a handle to a persistence mutable tensor that survives across executions of the graph.

When the applications created by the users, the *client*, communicates with TensorFlow, the *master*, it creates one or more *worker processes* that will be responsible for the arbitrary access to one or more computational devices and for executing graph nodes on those devices as instructed by the master. There are two ways to implement TensorFlow: the local implementation or the distributed implementation. In the local implementation all the components such as the client, master and workers run on the same operating system process. The distributed implementation is almost the same as the local with the difference of supporting an environment where master, client and workers operate in different machines. The difference between both implementations can be seen in figure 4.2.

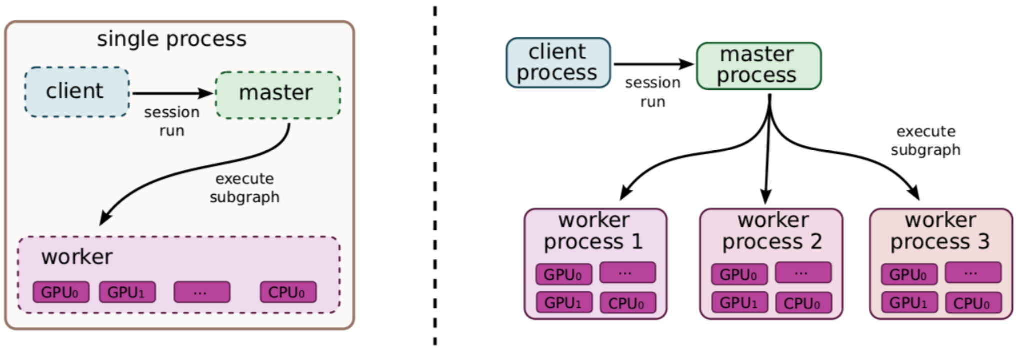


FIGURE 4.2: TensorFlow local vs distributed implementation (Source: [87]).

As referred earlier in section 2.1.2 deep neural networks have achieved breakthrough performance on computer vision tasks [44]. These tasks are one of the main applications for TensorFlow at Google. Training a network to achieve high accuracy values requires a large amount of computation, and TensorFlow has the purpose to scale out this computation across a cluster of GPU enabled servers. In this project, TensorFlow will be used to train a CNN to detect a fire using image processing techniques.

TensorFlow also comes with a powerful tool to help users understand the structure of their computation graphs and the overall behavior of machine learning models, TensorBoard. The graph analysis for deep neural networks can be quite complex, due to the size and topology of this graphs, normal visualization techniques often result in cluttered and overwhelming diagrams. In order to help users see the underlying organization of the graphs, the algorithms in TensorBoard break nodes into high-level blocks so as to point up groups with identical structures. This technique results in reduced clutter and focuses attention on the core sections of the computation graph. TensorBoard is user-friendly and offers real-time updates on the state of the various aspects of the model [87], has can be seen in figure 4.3.

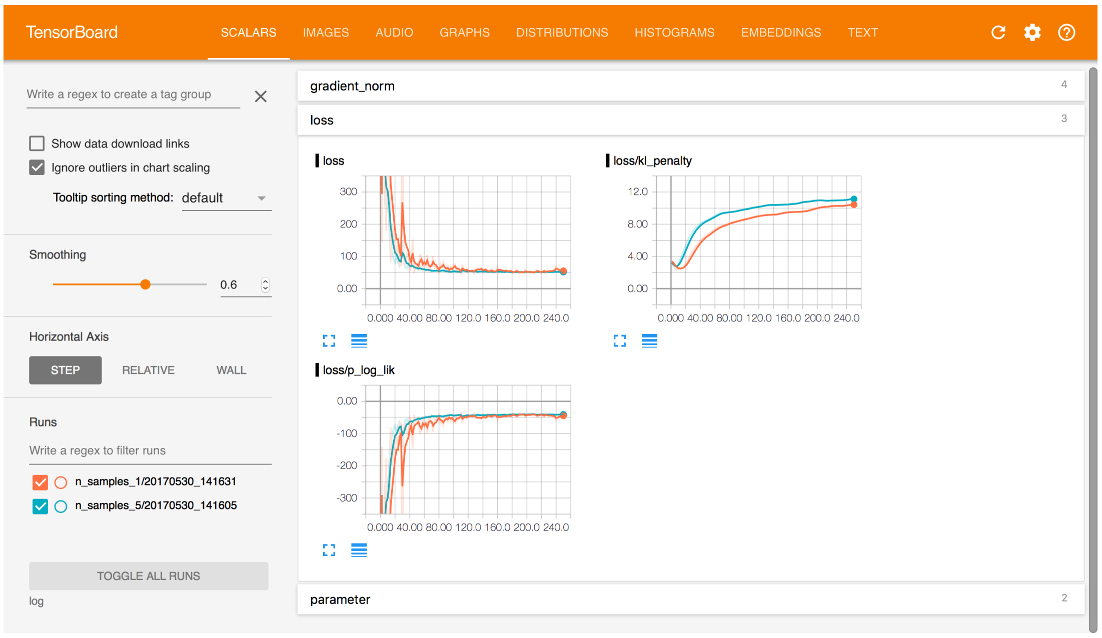


FIGURE 4.3: TensorBoard interface.

4.1.2 TensorFlow API

An Application Programming Interface, also know as API, allows system integration and information exchange between systems with different programming languages. Has many advantages such as data protection and security, performance optimization and process automation. This interface adds a set of standards, routines and programming functions defined through software that enables the use of application functionalities for the use of services without the need for their involvement in overly complex implementation processes. In short, an API is a connector interface that interconnects different applications systems with different programming languages quickly and safely.

TensorFlow provides an API for developers to use for both Python and C++ languages. It provides the user with the possibility to specify the subgraph that should be executed and also to select zero or more edges to feed input tensors into the dataflow, and one or more edges to fetch from the output tensors. Every invocation of the API is called a step, and TensorFlow supports multiple concurrent steps on the same graph. Steps can share data and synchronize when necessary. A typical training application can have multiple subgraphs that execute concurrently and interact through shared variables and queues. The core training subgraph depends on a set of model parameters and on a queue of input batches that update the model, to implement data-parallel training. The input queue is filled with several concurrent pre-processing steps that transform individual input records and a separate I/O subgraph reads records from a distributed file system. A checkpoint subgraph runs periodically to check for fault tolerance and to save execution state [87]. The training process is illustrated in figure 4.4.

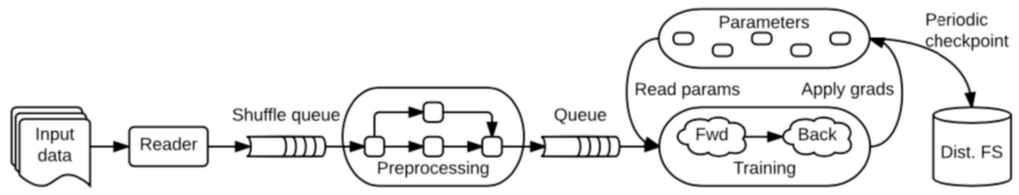


FIGURE 4.4: TensorFlow API training diagram (Source: [87]).

4.1.3 OpenCV

OpenCV is a open source computer vision library written in C and C++ that runs in multiple platforms. Computer vision means using still or real time video to make decisions or new representations based on the video input. OpenCV focus is exactly in real-time video applications. This type of applications are usually computationally expensive because they require advantage of multi-core processors. Besides containing over 500 functions that can be used in many areas such as security and medical imaging, it also contains a Machine Learning Library (MLL). This library is where this project will focus its attention on. OpenCV MLL allows statistical pattern recognition and clustering [88].

In this project OpenCV will be used with the custom pre-trained model created with TensorFlow, to identify and classify with localization a fire using real-time video captured by Raspberry Pi's camera. Once the fire is detected this will allow the fire alert notification to be sent, with the fire coordinates, to the user.

4.2 Training a Custom Object Detector

To train a custom model object detector, the TensorFlow API will be used. However, in order to train a model to detect fire, a custom dataset must first be created. The dataset is a collection of data that is normally associated with database scenarios. This fire dataset is a gathering of data to compose the database for this project. The only difference is, instead of values and strings, it will be a collection of pictures.

There are plenty of datasets already created for a lot of object types. However, because this project uses a specific angle of fire images and it is supposed to detect fires in the forest, the dataset has to be really specific and there is no available reliable datasets online. Therefore, to automatically detect a forest fire, a custom dataset must be created. When creating a dataset, the main dataset is split in 2 subsets which are called the "training" and "test" datasets. And one of the most important things to be aware of is the train/test dataset ratios: the training dataset trains the model and the testing dataset evaluates it. This ratio is very important, otherwise the model may end up overfitting or underfitting. Overfitting is the production of an analysis that corresponds too closely or exactly to a particular set of data, and may therefore fail to fit additional data or predict future observations reliably [89]. Underfitting occurs when a statistical model cannot adequately capture the underlying structure of the data. An underfitted model is a model where some parameters or terms that would appear in a correctly specified model are missing [90]. The ratio used for this dataset is of 90% of images for training and 10% for testing. The model for this project will have a total of 266 images, 240 (90.2%) for training and 26 (9.8%) for testing.

After collecting the dataset and dividing it, the next step is to classify all the images present in the training dataset. This process is called "image labeling". In order to teach TensorFlow what "the real aspect of fire is" and let it learn from each image, the fire object (technically referred as "fire" class) must be identified in every image with a specific location and size. For labelling every image, *LabelImg* will be used. This application creates a *.xml* file for each image that will contain the location and size of the fire object to let TensorFlow locate the pattern it needs to learn. An example of the labelling process is shown in figure 4.5.

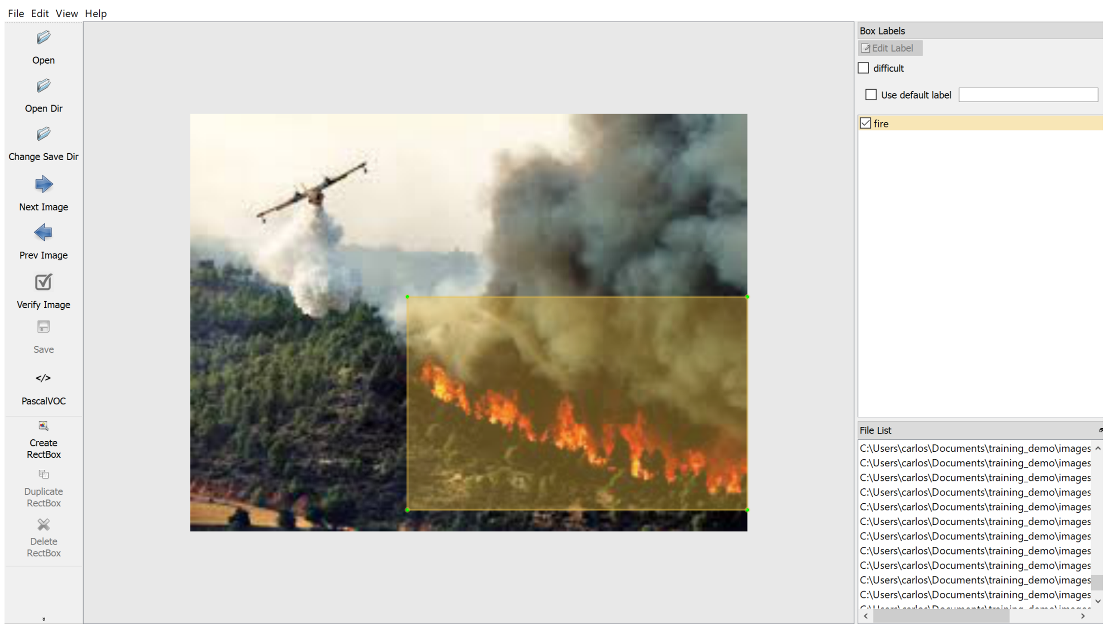

File Edit View Help
Open
Open Dir
Change Save Dir
Next Image
Prev Image
Verify Image
Save
PascalVOC
Create RectBox
Duplicate RectBox
Delete RectBox
Box Labels
Edit Label
difficult
Use default label
fire
File List
C:\Users\carlos\Documents\training_demo\images
C:\Users\carlos\Documents\training_demo\images
C:\Users\carlos\Documents\training_demo\images
C:\Users\carlos\Documents\training_demo\images
C:\Users\carlos\Documents\training_demo\images
C:\Users\carlos\Documents\training_demo\images
C:\Users\carlos\Documents\training_demo\images
C:\Users\carlos\Documents\training_demo\images
C:\Users\carlos\Documents\training_demo\images
C:\Users\carlos\Documents\training_demo\images
C:\Users\carlos\Documents\training_demo\images

FIGURE 4.5: Fire classification using LabelImg.

To train a model, TensorFlow also requires a label map, which namely maps each of the used labels to an integer value. This label map is afterwards used for both training and detection processes.

```
item {
    id: 1
    name: 'fire'
}
```

After creating the label map, a choice has to be made on the pre-trained model to use. Using a pre-trained model will simplify the task ahead because a lot of experiments are needed in order to build a proper CNN architecture. There are lots of models provided by TensorFlow. Since the Raspberry Pi has a weak processor, the model to use will be *ssd_ mobilenet_ v2_ cocoo* and because it takes less processing power. Even though the model will run faster, it comes with the disadvantage of having lower accuracy. After choosing the pre-trained model and labeling all the images, the training of the custom model can begin [91].

```
INFO:tensorflow:Starting Queues.
INFO:tensorflow:global_step/sec: 0
INFO:tensorflow:global step 1: loss = 13.8886 (12.339 sec/step)
INFO:tensorflow:global step 2: loss = 16.2202 (0.937 sec/step)
INFO:tensorflow:global step 3: loss = 13.7876 (0.904 sec/step)
INFO:tensorflow:global step 4: loss = 12.9230 (0.894 sec/step)
INFO:tensorflow:global step 5: loss = 12.7497 (0.922 sec/step)
INFO:tensorflow:global step 6: loss = 11.7563 (0.936 sec/step)
INFO:tensorflow:global step 7: loss = 11.7245 (0.910 sec/step)
(..)
INFO:tensorflow:Recording summary at step 64.
```

The previous program output demonstrates the output of the model while running the training process. The training time can variate according to the hardware specifications (either CPU or GPU). TensorFlow saves the state of the learning procedure every 1 minute. The model finishes training when it reaches a TotalLoss of at least 1 or lower for more accurate results. The TotalLoss value represent the model's accuracy to detect the desired pattern, in order to monitor it as well as other metrics, while the model is training, TensorBoard, will be used to analyze the performance of the generated model. Figure 4.6 shows it took 60000 steps to get to TotalLoss value of 0.4, to train the model.

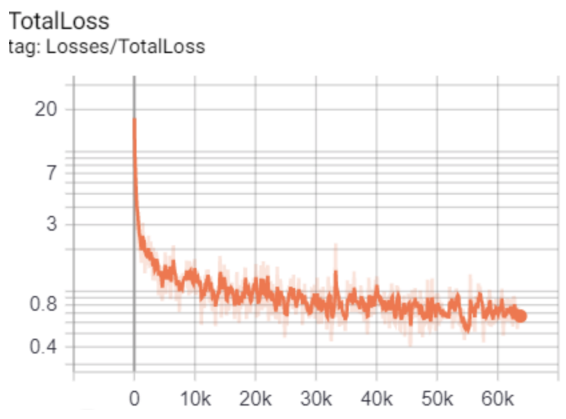


FIGURE 4.6: TotalLoss graph output.

Once the training process is completed, the frozen inference graph can be extracted. The frozen inference graph contains all of the required parameters (graph, weights, etc.) of the trained model, and it will be later used to perform the object detection using Raspberry Pi.

4.3 Detect Objects Using Real Time Video

In order to use the trained model to detect the fire object using real-time video, some hardware components are required: Raspberry Pi and Raspberry Pi camera module v2. Due to the fact that Raspberry Pi will be continuously used for extended periods of time, some heatsinks will be applied in order to avoid Raspberry Pi's CPU to become too hot. These heatsinks are necessary due to all the processing required to run the real-time video detection. High CPU temperatures will crash the Raspberry Pi's system and in more dangerous scenarios it can even burn the SD card resulting in a fire. Maintaining the CPU at lower temperatures is one top priority. To help with air circulation a small fan is also applied on top of the CPU. The Raspberry Pi's final customization is depicted in figure 4.7. The OS used with Raspberry is *Raspbian*, a specific Linux distribution for Raspberry Pi.

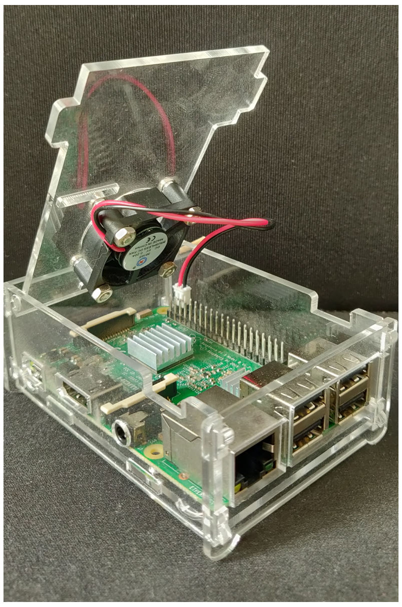


FIGURE 4.7: Customized Raspberry Pi 3.

With the hardware and the OS set up, the next step is setting up the TensorFlow environment. TensorFlow's object detection applications usually use matplotlib python library to display images. However, in this case OpenCV will be used, since it's more user friendly. At last, the installation of Protobuf, it is used by the TensorFlow object detection API and it is a Protocol Buffer data format. Protocol buffers is a method of serializing structured data in order to let developing programs communicate with each other over a wire or for storing data [92]. Currently there is no easy way to install Protobuf on the Raspberry Pi, the only way being to compile it from source and then install it. Lastly the previously custom trained model is used to create the fire detection script [93].

4.4 Fire Detection Script

The fire detector script was created using only the tools described before: OpenCV, TensorFlow, TextMagic API and two other python libraries called Keyboard and PiCamera. The script uses the model previously trained and its label map to detect fire. Firstly the label map and the TensorFlow model are loaded into the memory, then the size of the image to process is defined, which in this case it is the camera's resolution: 1280x720 pixels. To process the real-time video feed, the PiCamera library is used, because it returns frames that can later be processed accordingly using the OpenCV library. The average number of frames per second (FPS) processed are about 1.2 per second. This value is the optimal value for test experiences, due to Raspberry Pi's CPU processing limitations. Now, with the frames as input, a function can be created to process these frames. In this function named *fire_detector()* a trigger is created which uses the number of the class defined in the label map. In case of the number of the class is detected inside 8 frames the trigger will become active and send a text message to the user notifying him of the fire. The 8 frames are used to make sure the UAV gets an accurate result of fire detection, avoiding errors. This notification can be sent using the TextMagic API described in section 3.4. The trigger is activated using a frame counter, and afterwards, the Keyboard library is used to simulate pressing the 'q' key and the program exits and closes all threads and windows.

Chapter 5

Base Station and Landing Procedure

5.1 Automated Precision Landing

The Global Navigation Satellite System (GNSS) has evolved in the last decades and now new improved methods like Galileo have emerged and are used globally in all types of devices. These systems, like GPS, use radio frequencies to connect to satellites and from the obtained answer a global position can be calculated. Even though this solution is sufficient for a daily basis use, for other fields such as industry, high accuracy solutions are required. This need for higher accuracy has driven scientists to invent new solutions for this problem, one of those solutions being the Differential Global Navigation Satellite System (DGNSS).

5.1.1 Differential GPS

GPS is the most common known GNSS system and in the case of DGNSS technology it is the Differential GPS.

Differential GPS system is composed of two stations: a base station that is the reference point with its exact known 3D coordinates and a user station (known as rover). An antenna for receiving signals from GPS satellites is installed at each station [94]. The base station requires a minimum of four satellite connections and receives the positioning signal transmitted by the satellites continuously. Comparing the measured position coordinates (or pseudo range) with the known reference position coordinates (or pseudo range data) the difference is the corresponding differential correction. Once the corrections are calculated in the DGPS base station, the station sends a broadcast to all user stations in its corresponding region with the corrections by data link. Combining the user station's own positioning data and the corresponding differential corrections, the compensation operations are executed so as to improve the positioning accuracy of the user station, resulting in more accurate positioning results [95]. The DGPS accuracy is in the order of 1 m and the base station covers a range between 1 m to 150 km. A global view of a differential GPS can be seen in figure 5.1.

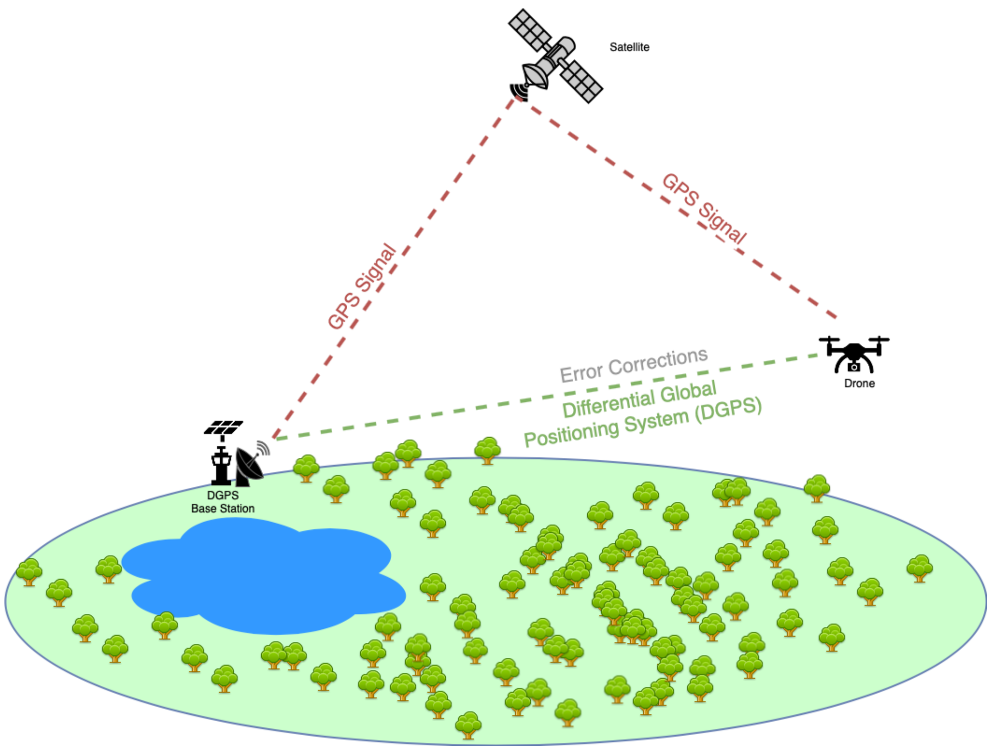


FIGURE 5.1: Differential GPS diagram.

5.1.2 RTK System

Real Time Kinematics (RTK) is also a DGNSS technique which is similar to DGPS. However because 1 meter accuracy was not sufficient, RTK solutions were developed and applied for several systems. An RTK base station covers a service area spreading about 10 or 20 km, and a real time communication channel is needed for connecting base and user station. The range is delimited by the difficulties to fix the carrier phase ambiguities as the distance between base and rover increases [96]. The main differences of RTK over DGPS are the increased accuracy, the decreased range, the faster and bigger data transmission, and the fact that RTK requires a minimum of 5 satellites [97]. This can change results from meter-lever accuracies to centimetre-level accuracies [98].

5.1.3 RTK System Architecture and Implementation

In order to implement the RTK GNSS system, several options were considered. However, the decision was simple since there was already available material. The RTK module and base station used for this project were Reach RS+ as a base station and Reach M+ as the UAV onboard module. Both products were directly acquired from the Emlid company.

The Reach M+, shown in figure 5.2, is a RTK GNSS module for precise navigation and UAV mapping. This module is mounted on the UAV and connects to the Pixhawk controller. The input voltage is 4.75 - 5.5 V and the average current consumption is 200 mA. It supports various constellations, such as GPS/QZSS L1, GLONASS G1, BeiDou B1, Galileo E1, SBAS and the update rate is 14 Hz / 5 Hz. It receives the corrections errors via LoRa 868/915 MHz communication protocol, has internal storage of 8 GB and reads the correction input in both RTCM2 and RTCM3 protocols. This module has two types of operating modes: transmitter and receiver. The user can even connect directly to the module via Wi-Fi or Bluetooth [99].

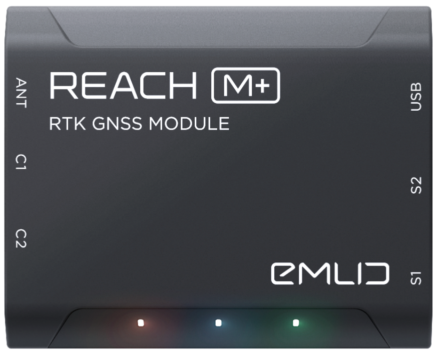


FIGURE 5.2: Reach M+ module.

The Reach RS+, shown in figure 5.3, is the designated base station, which is where the differential corrections are calculated. Multiple rovers can be connected to one base station at the same time. This module is configured in transmitter mode which is connected to the LoRa antenna and to a battery supply that can work up to 30 hours. The GNSS characteristics are the same as in the Reach M+ module and it has the same internal storage as well. The Reach RS+ measures its exact 3D location before the transmission of the differential correction starts. Afterwards, it sends this correction via LoRa 868/915 MHz. As previously mentioned, the user can also connect directly to the base station via Wi-Fi or Bluetooth and extract the logs or follow the UAV's location. This base station has its own app and it is available on Goggle Play and App Store [100].

In addition to the Reach M+, the HERE2 GPS module was also used, resulting in a so called GPS redundancy: the duplication of critical components or functions of a system with the intention of increasing reliability of the system. In this situation, the GPS redundancy will not only be used as a performance improvement but also as a fail-safe when there is a failure of signal loss in one of the GPS modules. UAVs with dual GPS sensor can be used in situations with electromagnetic interference or in case of having interference from objects along the GPS' signal path (ex: concrete walls). This feature is useful in cases such as powerline

Figure 5.3: Reach RS+ base station.

inspections, monitoring mining works and other environments with strong electromagnetic fields which may cause electromagnetic interference [101][102]. This system's RTK architecture and all it's tools are illustrated in figure 5.4.

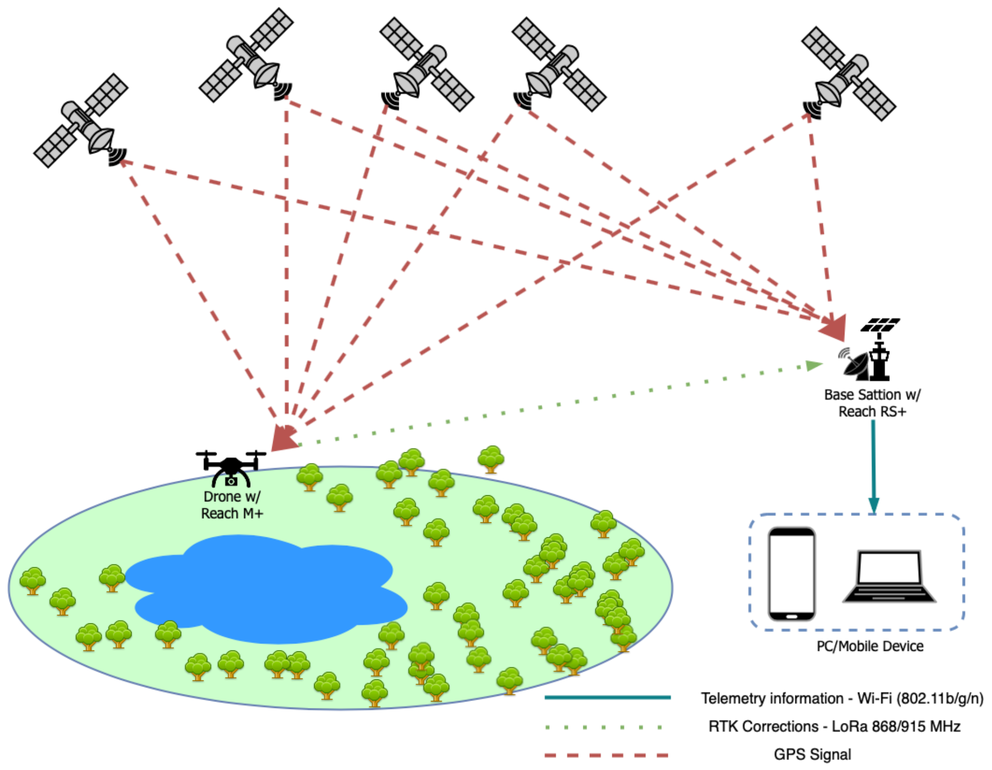

Base Sattion w/
Reach RS+
Drone w/
Reach M+
PC/Mobile Device
Telemetry information - Wi-Fi (802.11b/g/n)
RTK Corrections - LoRa 868/915 MHz
GPS Signal

Figure 5.4: RTK system architecture diagram.

With all the modules, tools and different possibilities this system can provide, it results in a complete and versatile option to consider for other types of projects such as machinery guidance, UAV mapping and precise point data collection.

5.1.4 Configuration

In order to integrate Reach RS+ base station into the system, several basic configurations must be applied. To configure the base station any Android or iPhone mobile device can be used. The application to manage both modules settings is called *ReachView*.

Firstly, the base station must be powered up and the smartphone must be connected to the Wi-fi hotspot created by the base station. To perform a successful connection, after opening the app, the desired module is selected, as shown in figure 5.5.

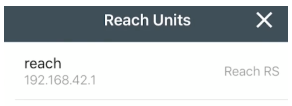
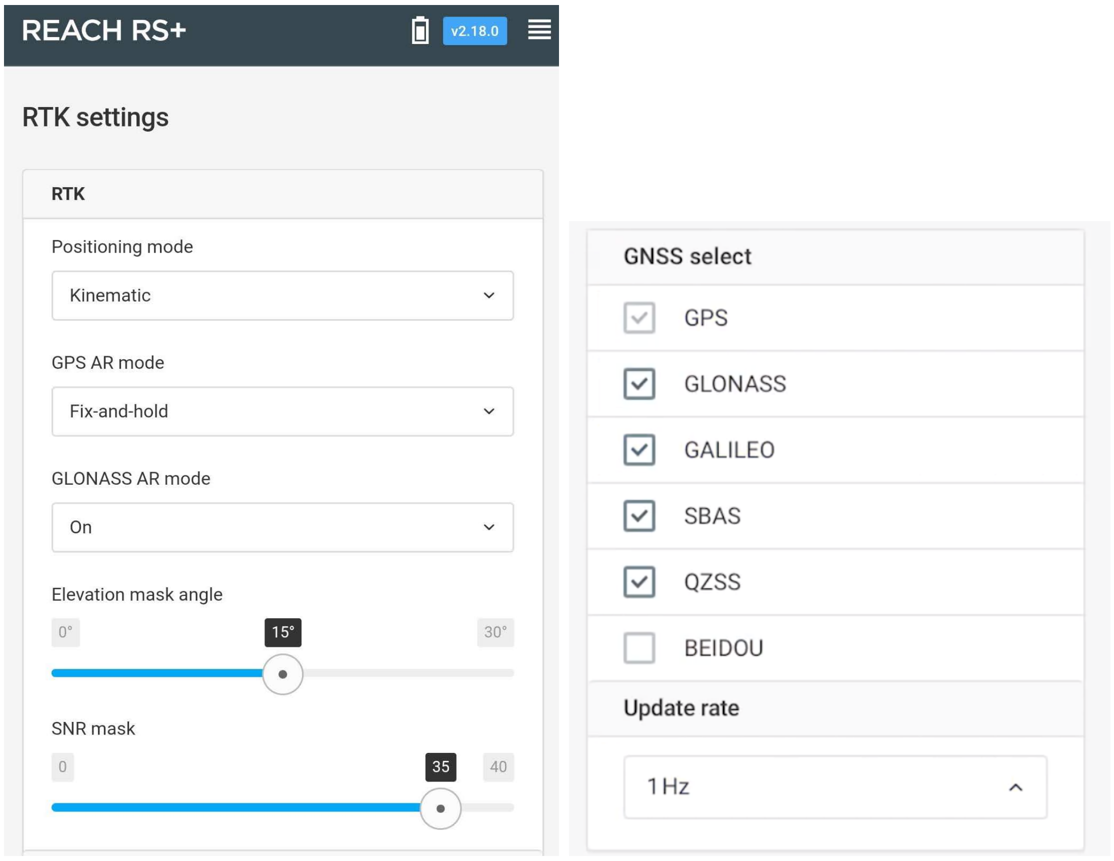


FIGURE 5.5: ReachView vehicle connection.

When already connected to the base station Wi-fi hotspot, its configuration must be changed and the base station must connect to another Wi-fi network with internet access in order see if there is any need for updates to be made in the base station's firmware and eventually reboot it. The same procedure is repeated for the Reach M+ module. Afterwards, the Rover <-> Base Station connection via Lora is ready to be configured [103].

Since both rover and base station have the same name, to start setting up the connection the first thing to do will be to change the names of the modules. The base station will be called reach-base and the rover will be called reach-rover. Starting with the reach-base station settings, one will first configure the RTK settings and the communication between base and rover. When changing the RTK settings, all the GNSS systems need to be enabled and the update rate must be set to 1 Hz, in similarity to figure 5.6.



REACH RS+ v2.18.0

RTK settings

RTK

Positioning mode: Kinematic

GPS AR mode: Fix-and-hold

GLONASS AR mode: On

Elevation mask angle: 0° — 15° — 30°

SNR mask: 0 — 35 — 40

GNSS select

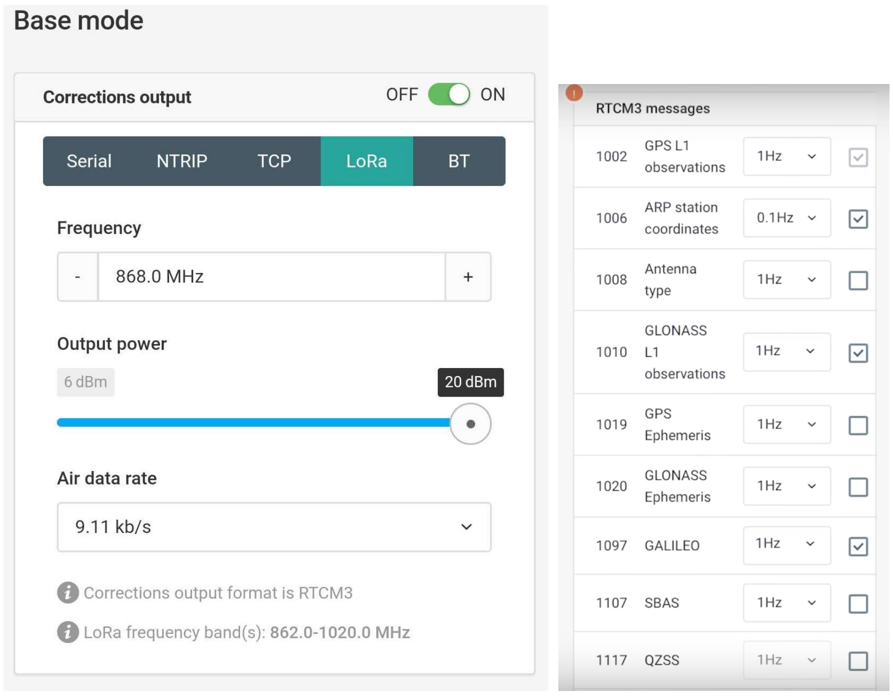
- ✓ GPS
- ✓ GLONASS
- ✓ GALILEO
- ✓ SBAS
- ✓ QZSS
- ☐ BEIDOU

Update rate: 1 Hz

FIGURE 5.6: ReachView RTK settings.

After configuring the RTK settings, the next step is setting up the LoRa radio on Reach RS+ base station to broadcast RTK corrections. In the list of RTCM3 messages the outputs GPS L1, GLONASS L1 and GALILEO observations must be set to 1 Hz and the ARP station coordinates to 0.1 Hz. The output power value will be 20 dBm and air rate will be at 9.11 kb/s. After the previous steps the RTK settings must match the one's in figure 5.7.

Having all reach-base settings done, the reach-rover follows. After connecting to it, in the RTK settings, the positioning mode is changed to Kinematic option, GPS Ambiguity resolution mode to Fix-and-hold option and GLONASS Ambiguity resolution mode to ON option. Then the same GNSS systems as for the reach-base are enabled and the update rate will be 5 Hz. After configuring the LoRa radio on the reach-rover unit to receive the corrections, in the Correction input tab the Base corrections frequency and the air rate settings must match


Base mode

Corrections output OFF ON

Serial | NTRIP | TCP | LoRa | BT

Frequency

- 868.0 MHz +

Output power

6 dBm 20 dBm

Air data rate

9.11 kb/s

Corrections output format is RTCM3

LoRa frequency band(s): 862.0-1020.0 MHz

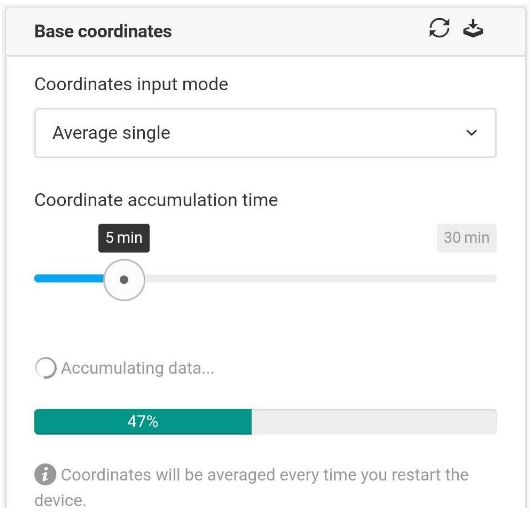
RTCM3 messages

1002	GPS L1 observations	1Hz	✓
1006	ARP station coordinates	0.1Hz	✓
1008	Antenna type	1Hz	☐
1010	GLONASS L1 observations	1Hz	✓
1019	GPS Ephemeris	1Hz	☐
1020	GLONASS Ephemeris	1Hz	☐
1097	GALILEO	1Hz	✓
1107	SBAS	1Hz	☐
1117	QZSS	1Hz	☐

FIGURE 5.7: ReachView base mode settings.

what was configured on the reach-base [104].

Having concluded all the main settings, the only required procedure is to set the mode which the base station will run on.


Base coordinates

Coordinates input mode

Average single

Coordinate accumulation time

5 min 30 min

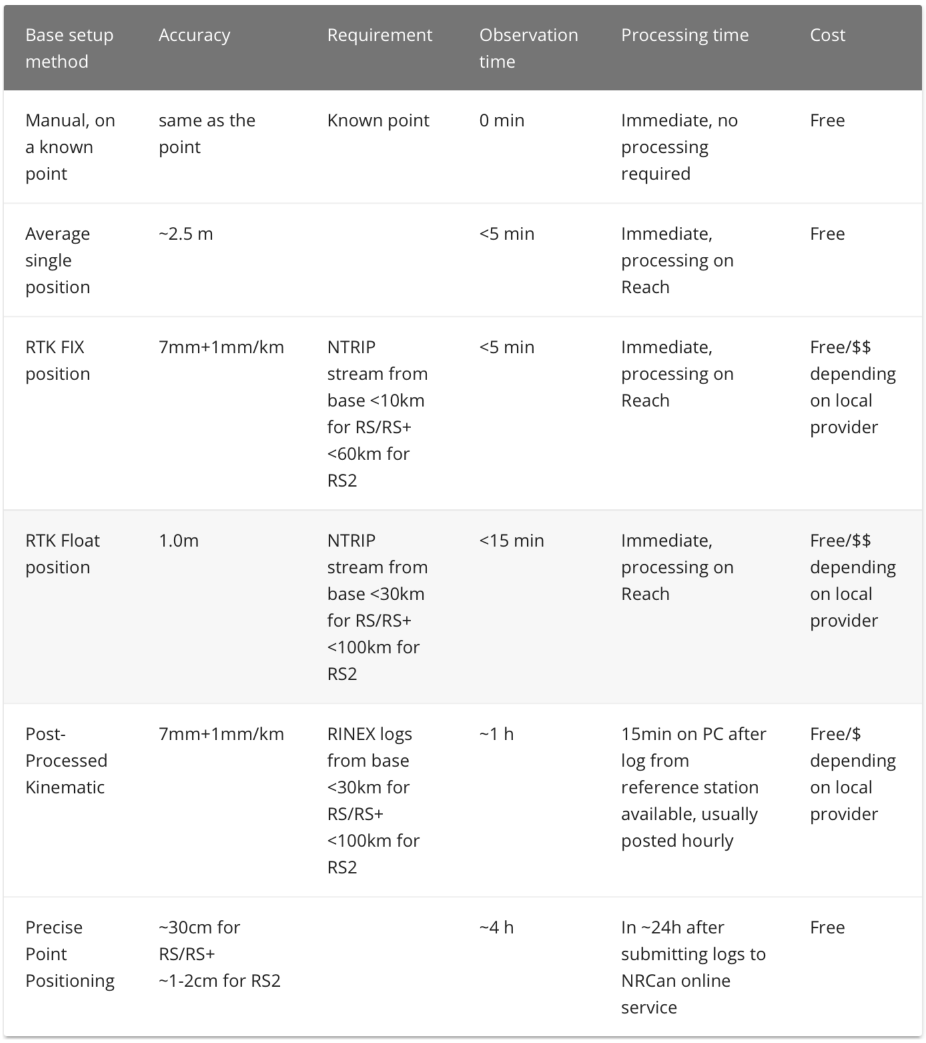
Accumulating data...

47%

Coordinates will be averaged every time you restart the device.

FIGURE 5.8: ReachView base station precision.

The mode defined in "Coordinates input mode" in figure's 5.8 defines the precision of the RTK corrections. The different modes are shown in figure 5.9.



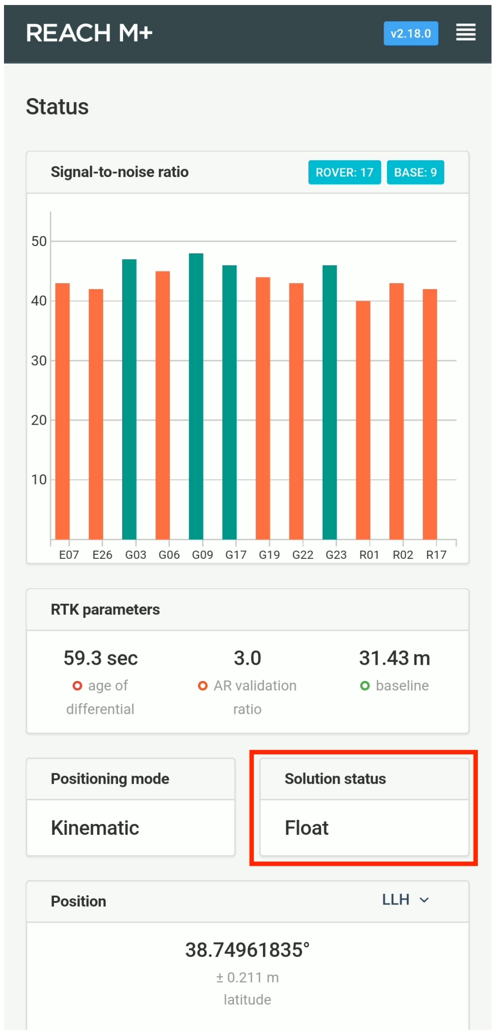
Base setup method	Accuracy	Requirement	Observation time	Processing time	Cost
Manual, on a known point	same as the point	Known point	0 min	Immediate, no processing required	Free
Average single position	~2.5 m		<5 min	Immediate, processing on Reach	Free
RTK FIX position	7mm+1mm/km	NTRIP stream from base <10km for RS/RS+ <60km for RS2	<5 min	Immediate, processing on Reach	Free/$$ depending on local provider
RTK Float position	1.0m	NTRIP stream from base <30km for RS/RS+ <100km for RS2	<15 min	Immediate, processing on Reach	Free/$$ depending on local provider
Post-Processed Kinematic	7mm+1mm/km	RINEX logs from base <30km for RS/RS+ <100km for RS2	~1 h	15min on PC after log from reference station available, usually posted hourly	Free/$ depending on local provider
Precise Point Positioning	~30cm for RS/RS+ ~1-2cm for RS2		~4 h	In ~24h after submitting logs to NRCan online service	Free

FIGURE 5.9: ReachView RTK precision levels.

In the app and in the Status tab on the reach-rover's central panel, the precision of the corrections can be tracked and the *Solution status* describes the rover's precision in real time, as shown in figure 5.10.

The *Solution status* can have three modes:

- **Single** means that rover has found a solution relying on its own receiver and base corrections are not applied. Precision in standalone mode is usually meter-level;
- **Float** means that the base corrections are now taken into consideration;



Status

Signal-to-noise ratio

RTK parameters

59.3 sec age of differential

3.0 AR validation ratio

31.43 m baseline

Positioning mode: Kinematic

Solution status: Float

Position

38.74961835°
± 0.211 m
latitude

FIGURE 5.10: ReachView RTK precision level on rover.

- **Fix** status means all ambiguities are resolved and RTK solution is centimetre-level accurate [104].

5.2 Recharging Station

An UAV, to become completely autonomous, requires a way to recharge without the help of human intervention. Since the UAV's typical recharging procedure requires for the user to remove the battery and manually plug it into a charger that option becomes completely impracticable. A more easy and approachable solution would be to use a technology that is used in hundreds of mobile devices, like our phones, which is called Wireless Power Transfer (WPT). This option does not require wires and does not involve removing the battery which makes it more suitable. The only problem in this system is that it requires a high precision contact between both charger and mobile device. Also the high power charger creates electromagnetic interference causing GPS signal failures. However since the UAV uses a RTK receiver and another GPS, the high precision problem is solved as well as for the electromagnetic interference with the GPS redundancy. This section reports a case study since no test fields were made.

5.2.1 Wireless Power Transfer

Wireless Power Transfer (WPT) is the transmission of electrical energy from a power source to an electrical load, without any connectors, across the air. A wireless power system is based on a two coil system, a transmitter coil and a receiver. These coils transfer energy using a oscillating magnetic field.

In order for this phenomenon to happen, Direct Current (DC) supplied from the power source is converted into high-frequency Alternating Current (AC) by specific electronics built into the transmitter. Then the AC current energizes a copper wire coil in the transmitter, resulting in a generated magnetic field. When another coil is placed within the proximity of the magnetic field, the field induces AC current in the receiving coil. The AC current is then converted again into the DC current by specific electronics.

WPT's main objective is in DC manipulation, to energize the coil at the transmitter end, the DC input must be converted to AC output because the mutual induction only happens with AC. For this purpose, there is a need for an inverter design which converts the DC input to the AC output. The DC power for the magnetic induction uses coupled inductors at a certain resonant frequency, in order to manipulate the frequency a resonator is used. After successful WPT, the transferred AC is rectified via a bridge circuit. The noise filtration of the rectified DC current allows the usage of wirelessly transferred power for an output load such as a multicopter's battery. Figure 5.11 represents the WPT system as a block diagram.

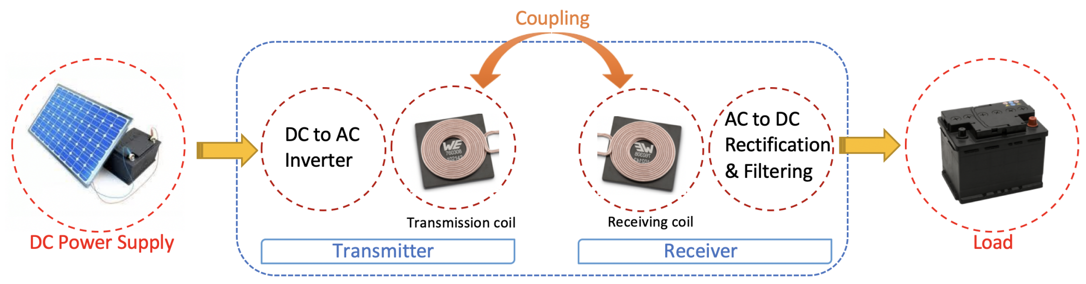


FIGURE 5.11: Block diagram for WPT.

Non-resonant coupled inductors, such as typical transformers, need a magnetic core and require the magnetic field to be covered by the secondary wire in a sufficiently short range. Non-resonant induction is highly inefficient in scenarios with larger distances and when the primary coil has high resistive losses, which discards the majority of power to transfer to the secondary coil. In order to solve this problem, resonant coupling can be used to help the coils improving the efficiency. A basic resonant circuit uses an LC circuit consisting of an inductor (coil) and a capacitor that can be used on both transmitter and receiver sides. If both coils are resonant at the same frequency, the power transferred between them can increase significantly. The space between the coils can then increase to be a few times higher than the coil's diameter, maintaining a reasonable efficiency.

A big part of the described resonant circuit is the design of the coils. The power transfer efficiency is affected by the shape of the coils because their inductance changes according to the shape, the optimal coil shapes are represented in figure 5.12 [75].

5.2.2 WPT System Architecture and Implementation

The system is composed of a solar panel, 2 lithium batteries and the wireless charging system, just as in figure 5.13. The batteries store the electrical power coming from the solar panel.

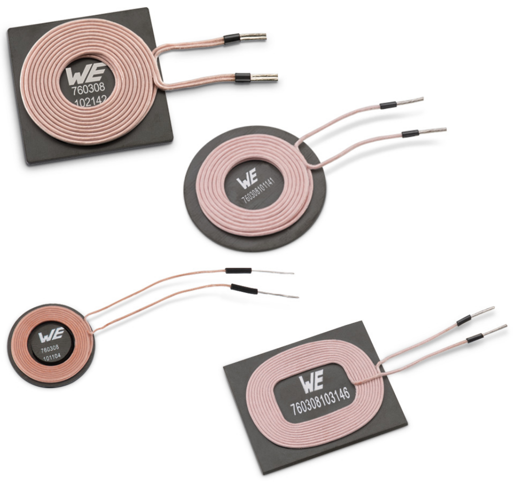


FIGURE 5.12: WPT coils.

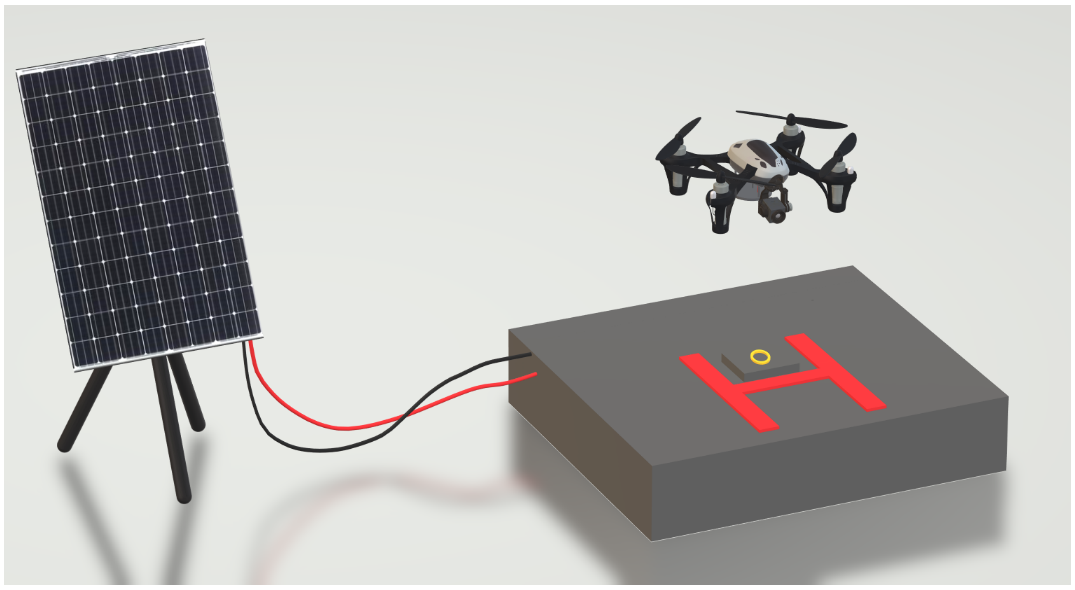


FIGURE 5.13: Base station 3D model.

For a 250 watts (W) solar panel using 4 hours of full sun ($250W \cdot 4h$) that results in 1 kWh (1.000 W hour) per panel, in one day. Multiplying 1 kWh per panel by an average of 30 days in a month, then each 250 W rated panel produces about 30 kWh in a month.

Battery capacity is measured by Ah (ampere hour). A 12 V lithium battery storage can go from 7 Ah to 300 Ah, if the selected capacity is 40 Ah and both batteries are connected in parallel then a total capacity of 80 Ah is achieved. A UAV spends at maximum the totality of its battery, which is 10.000 mAh (10 Ah).

A station fully charged has enough power to fully charge an UAV 8 times. A view of the recharging system with the wiring and battery disposition is shown in 5.14.

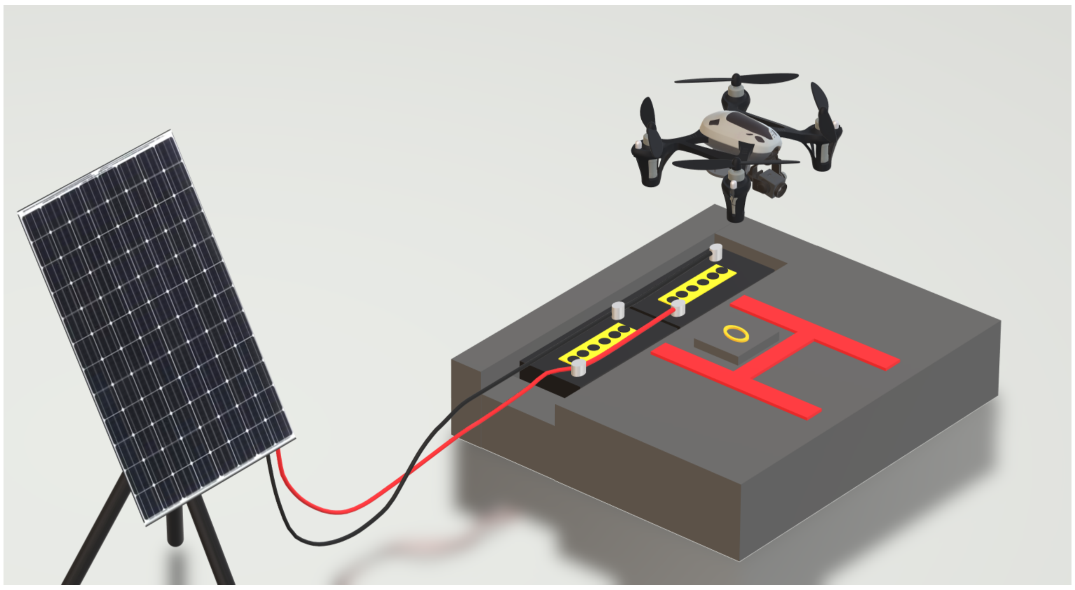


FIGURE 5.14: Base station 3D model with wiring configuration.

The selected charging kit is the Power Management IC Development Tools Wireless Power 200 W Development Kit made by Wurth Elektronik. At the ideal conditions with a maximum output of 200 W (10 A, 24 V) and if the UAV's battery is 10.000 mAh (24 V) then the UAV takes approximately 1 hour and 13 minutes to charge. Nevertheless, the charger does not always transfer the same amount of energy because it adapts to the lithium power cells' needs. Thus the charging procedure takes more than 1 hour and 13 minutes. It is however impossible to affirm the exact time without real life testing.

In a study made about wireless charging for UAVs, it has been proven that the magnetic field is really strong around the transmitter and receiving coil [105]. Nonetheless by creating a simple support to separate the receiving coil from the UAV's skeleton, as shown in the figure 5.15, the magnetic field is reduced substantially. The coils in the development kit are half the receiving coil and 8 times smaller than the transmitter coil also resulting in a reduced magnetic field. This alteration also comes with a cost: because the coils are smaller, the distance between them needs to be smaller, that is why in the base station the transmitter coil needs to be elevated to make perfect contact with the receiving coil, as in figure 5.16. The offset of these coils is 10 mm which is equivalent to 1 cm, in order to recharge at maximum transfer power. Even so the RTK system provides a centimetre accuracy making it a perfect solution for the system.

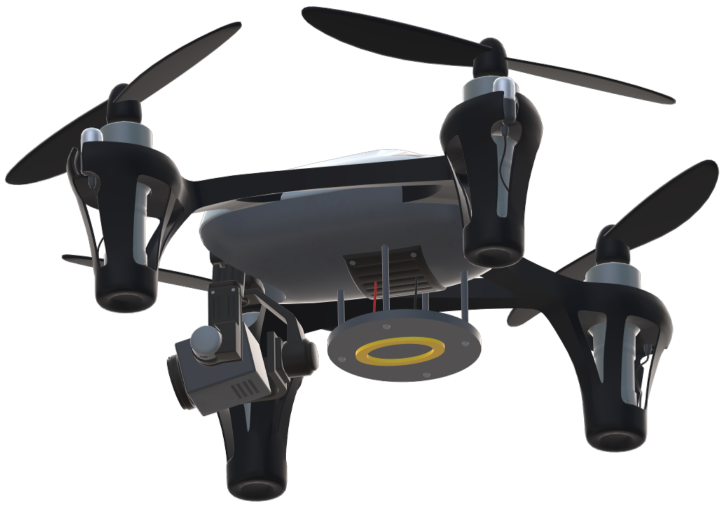


FIGURE 5.15: UAV's WPT coil application.

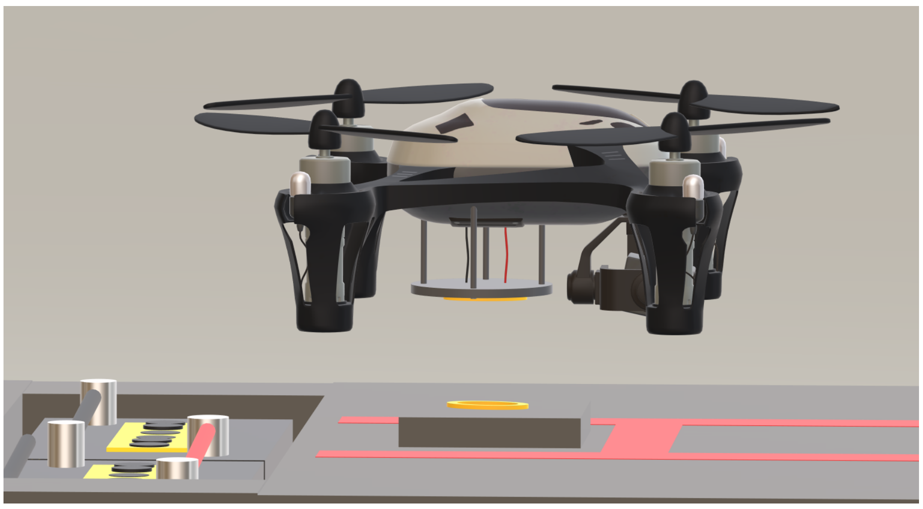


FIGURE 5.16: UAV and base WPT recharging system.

Chapter 6

Results

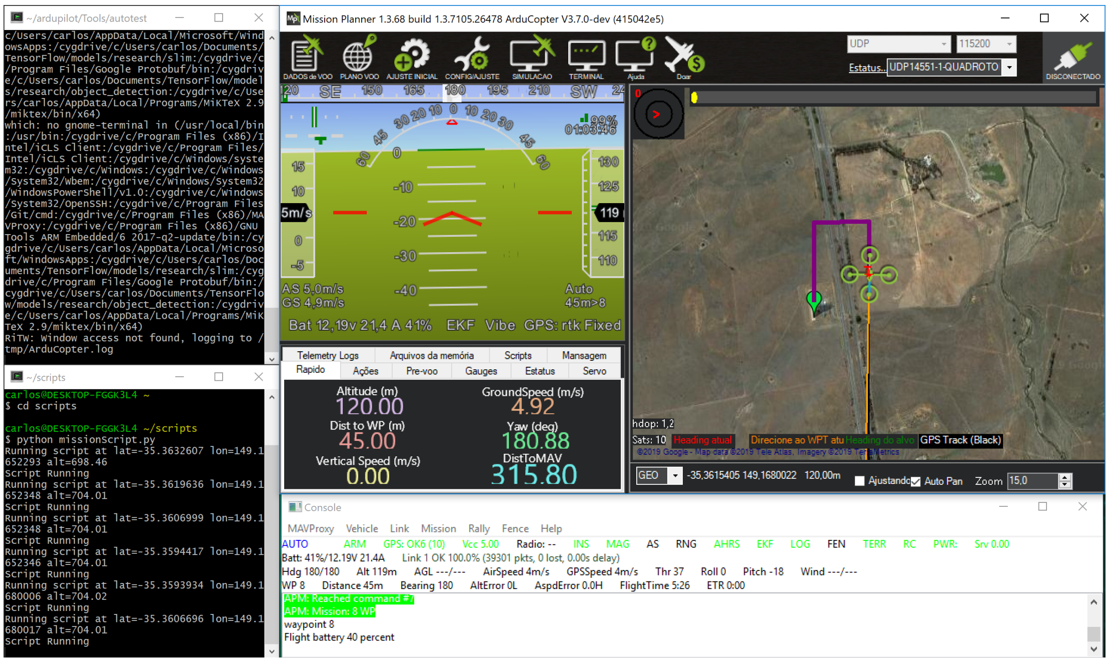
6.1 Flight Automation Simulation

In order to test the flight automation script the best available option is to use a simulator. When uploading a mission and different commands to a flight controller, lots of errors can occur. When testing scripts in a real life situation, a malfunction could happen during the flight and the UAV would probably crash. In order to avoid any critical damage, a simulator known as SITL is used as it allows to simulate a Pixhawk-like flight controller and run ArduPilot firmware directly on a PC, without any special hardware.

When running in SITL the sensor data comes from a flight dynamics model in a flight simulator. ArduPilot has a wide range of vehicle simulators built in and can interface to several external simulators allowing ArduPilot to be tested on a very wide variety of vehicle types, such as in this case a multi-rotor aircraft [106].

For a more accurate and easier way to track the mission, a GCS with a user interface, Mission Planner, previously referred in section 2.5 is used. MAVProxy which is an UAV ground station software package for MAVLink based systems is also used to help keeping track of the UAV systems [107].

Firstly, the UAV emulation is initiated using a script provided by SITL. Then, the MAVProxy console pops up and the Mission Planner is connected to the UAV, resulting in the mission ready to start. The mission environment is shown in figure 6.1.



```
c/Users/carlos/AppData/Local/Microsoft/Wind
owsApps:/cygdrive/c/Users/carlos/Documents/
TensorFlow/models/research/slim:/cygdrive/c
/Program Files/Google Protobuf/bin:/cygdriv
e/c/Users/carlos/Documents/TensorFlow/model
s/research/object_detection:/cygdrive/c/Use
rs/carlos/AppData/Local/Programs/MiKTeX 2.9
/miktex/bin/x64)
which: no gnome-terminal in (/usr/local/bin
:/usr/bin:/cygdrive/c/Program Files (x86)/I
ntel/iCLS Client:/cygdrive/c/Program Files/
Intel/iCLS Client:/cygdrive/c/Windows/syste
m32:/cygdrive/c/Windows:/cygdrive/c/Windows
/System32/Wbem:/cygdrive/c/Windows/System32
/WindowsPowerShell/v1.0:/cygdrive/c/Windows
/System32/OpenSSH:/cygdrive/c/Program Files
/Git/cmd:/cygdrive/c/Program Files (x86)/MA
VProxy:/cygdrive/c/Program Files (x86)/GNU
Tools ARM Embedded/6 2017-q2-update/bin:/cy
gdrive/c/Users/carlos/AppData/Local/Microso
ft/WindowsApps:/cygdrive/c/Users/carlos/Doc
uments/TensorFlow/models/research/slim:/cyg
drive/c/Program Files/Google Protobuf/bin:/
cygdrive/c/Users/carlos/Documents/TensorFlo
w/models/research/object_detection:/cygdriv
e/c/Users/carlos/AppData/Local/Programs/MiK
TeX 2.9/miktex/bin/x64)
RiTW: Window access not found, logging to /
tmp/ArduCopter.log
```

```
carlos@DESKTOP-FGGK3L4 ~
$ cd scripts

carlos@DESKTOP-FGGK3L4 ~/scripts
$ python missionScript.py
Running script at lat=-35.3632607 lon=149.1
652293 alt=698.46
Script Running
Running script at lat=-35.3619636 lon=149.1
652348 alt=704.01
Script Running
Running script at lat=-35.3606999 lon=149.1
652348 alt=704.01
Script Running
Running script at lat=-35.3594417 lon=149.1
652346 alt=704.01
Script Running
Running script at lat=-35.3593934 lon=149.1
680006 alt=704.02
Script Running
Running script at lat=-35.3606696 lon=149.1
680017 alt=704.01
Script Running
```

```
AUTO    ARM    GPS: OK6 (10)    Vcc 5.00    Radio: --    INS    MAG    AS    RNG    AHRS    EKF    LOG    FEN    TERR    RC    PWR:    Srv 0.00
Batt: 41%/12.19V 21.4A    Link 1 OK 100.0% (39301 pkts, 0 lost, 0.00s delay)
Hdg 180/180    Alt 119m    AGL ---/---    AirSpeed 4m/s    GPSSpeed 4m/s    Thr 37    Roll 0    Pitch -18    Wind ---/---
WP 8    Distance 45m    Bearing 180    AltError 0L    AspdError 0.0H    FlightTime 5:26    ETR 0:00
APM: Reached command #7
APM: Mission: 8 WP
waypoint 8
Flight battery 40 percent
```

FIGURE 6.1: Simulation environment.

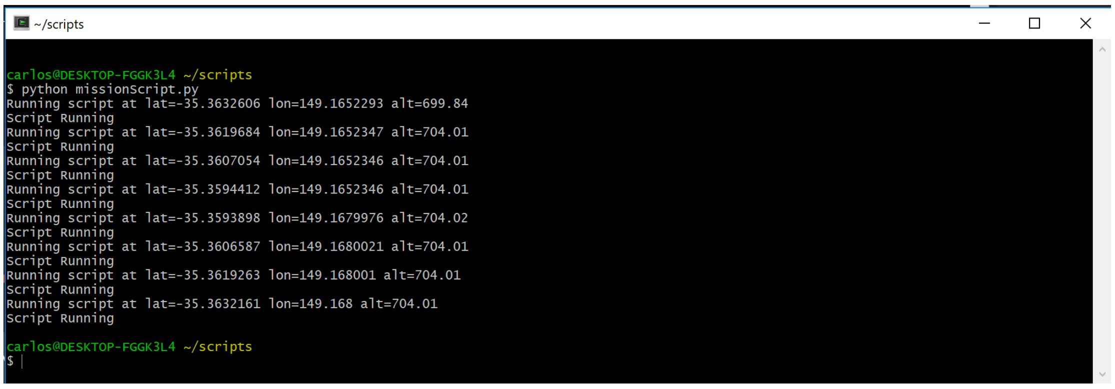
In order to start the mission, the mission's script is executed using the command line. The first output the script provides is the UAV's connection and the system's status as referred in section 3.1.2. Afterwards, any existing commands are erased and a new mission is uploaded to the controller, the UAV will then start the take-off procedure. Figure 6.2 represents the pre-flight log output, the full log file of the simulation can be found in section A.

When the take-off procedure ends, the mission is executed and the mission status can be followed in the command line, as in figure 6.3.

After the mission completes, a summary can be seen in the MAVProxy console where the most important events are registered, such as the flight duration and battery percentage left. A small part of the MAVProxy console of the testing simulation can be found in figure 6.4.

```
[INFO]:  Connecting to vehicle on: 127.0.0.1:14550
[INFO]:  Connected
[DEBUG]: Starting system analysis
[INFO]:  Global Location: LocationGlobal:lat=0.0,lon=0.0,alt=None
[INFO]:  Global Location (relative altitude):
[INFO]:  LocationGlobalRelative:lat=0.0,lon=0.0,alt=0.0
[INFO]:  Local Location: LocationLocal:north=None,east=None,down=None
[INFO]:  Attitude:
[INFO]:  Attitude:pitch=-0.000112996094686,yaw=-0.329702675343,
roll=0.000109214859549
[INFO]:  Velocity: [0.0, 0.0, 0.0]
[INFO]:  GPS: GPSInfo:fix=0,num_sat=0
[INFO]:  Gimbal status: Gimbal: pitch=None, roll=None, yaw=None
[INFO]:  Battery: Battery:voltage=0.0,current=0.0,level=100
[INFO]:  EKF OK?: False
[INFO]:  Last Heartbeat: 0.75
[INFO]:  Rangefinder: Rangefinder: distance=None, voltage=None
[INFO]:  Rangefinder distance: None
[INFO]:  Rangefinder voltage: None
[INFO]:  Heading: 341
[INFO]:  Is Armable?: False
[INFO]:  System status: ACTIVE
[INFO]:  Groundspeed: 0.0
[INFO]:  Airspeed: 0.0
[INFO]:  Mode: STABILIZE
[INFO]:  Armed: False
[INFO]:  Vehicle EKF not ok
[DEBUG]: Waiting EKF
[INFO]:  Create a new mission (for current location)
[DEBUG]: Clear any existing commands
[DEBUG]: Define/add new commands.
[DEBUG]: Uploading new commands to vehicle
[DEBUG]: Starting takeoff
```

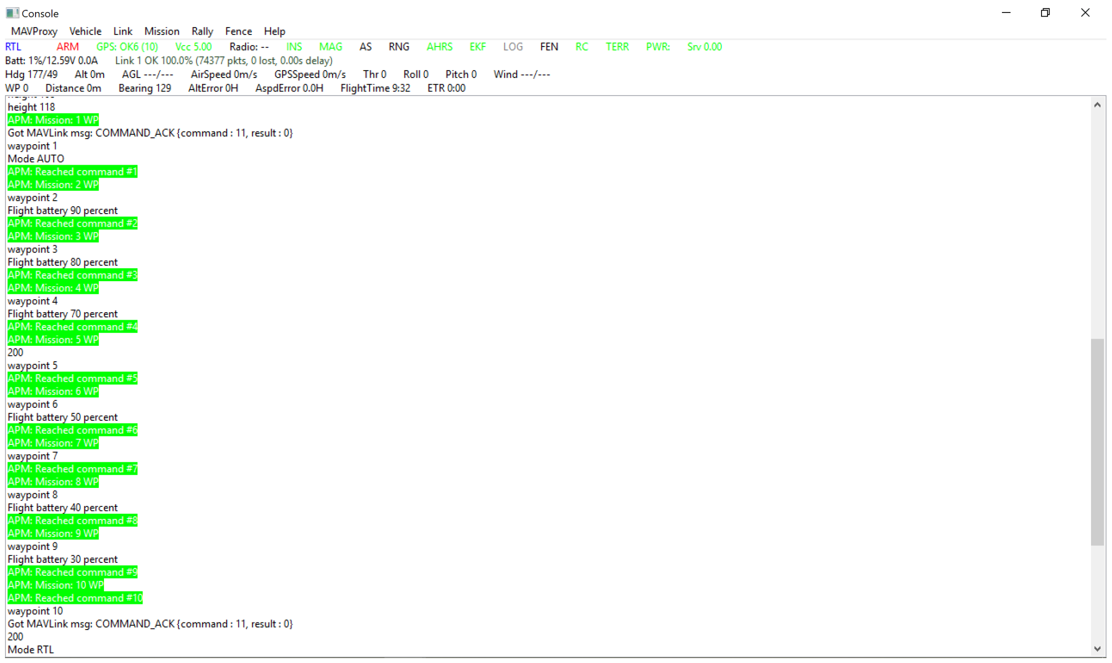
FIGURE 6.2: Pre-flight log output.



```
~/scripts

carlos@DESKTOP-FGGK3L4 ~/scripts
$ python missionScript.py
Running script at lat=-35.3632606 lon=149.1652293 alt=699.84
Script Running
Running script at lat=-35.3619684 lon=149.1652347 alt=704.01
Script Running
Running script at lat=-35.3607054 lon=149.1652346 alt=704.01
Script Running
Running script at lat=-35.3594412 lon=149.1652346 alt=704.01
Script Running
Running script at lat=-35.3593898 lon=149.1679976 alt=704.02
Script Running
Running script at lat=-35.3606587 lon=149.1680021 alt=704.01
Script Running
Running script at lat=-35.3619263 lon=149.168001 alt=704.01
Script Running
Running script at lat=-35.3632161 lon=149.168 alt=704.01
Script Running

carlos@DESKTOP-FGGK3L4 ~/scripts
$ |
```

FIGURE 6.3: Mission script outputs.

Finally, the script saves the log file to a log directory and names the file with the date and hour of when the script was initialized. The log distinguishes DEBUG from INFO procedures in order to make the log easier to read and to define which statements are more relevant.



```
RTL     ARM   GPS: OK6 (10)   Vcc 5.00   Radio: --   INS   MAG   AS   RNG   AHRS   EKF   LOG   FEN   RC   TERR   PWR:   Srv 0.00
Batt: 1%/12.59V 0.0A   Link 1 OK 100.0% (74377 pkts, 0 lost, 0.00s delay)
Hdg 177/49   Alt 0m   AGL ---/---   AirSpeed 0m/s   GPSSpeed 0m/s   Thr 0   Roll 0   Pitch 0   Wind ---/---
WP 0   Distance 0m   Bearing 129   AltError 0H   AspdError 0.0H   FlightTime 9:32   ETR 0:00
height 118
APM: Mission: 1 WP
Got MAVLink msg: COMMAND_ACK {command : 11, result : 0}
waypoint 1
Mode AUTO
APM: Reached command #1
APM: Mission: 2 WP
waypoint 2
Flight battery 90 percent
APM: Reached command #2
APM: Mission: 3 WP
waypoint 3
Flight battery 80 percent
APM: Reached command #3
APM: Mission: 4 WP
waypoint 4
Flight battery 70 percent
APM: Reached command #4
APM: Mission: 5 WP
200
waypoint 5
APM: Reached command #5
APM: Mission: 6 WP
waypoint 6
Flight battery 50 percent
APM: Reached command #6
APM: Mission: 7 WP
waypoint 7
APM: Reached command #7
APM: Mission: 8 WP
waypoint 8
Flight battery 40 percent
APM: Reached command #8
APM: Mission: 9 WP
waypoint 9
Flight battery 30 percent
APM: Reached command #9
APM: Mission: 10 WP
APM: Reached command #10
waypoint 10
Got MAVLink msg: COMMAND_ACK {command : 11, result : 0}
200
Mode RTL
```

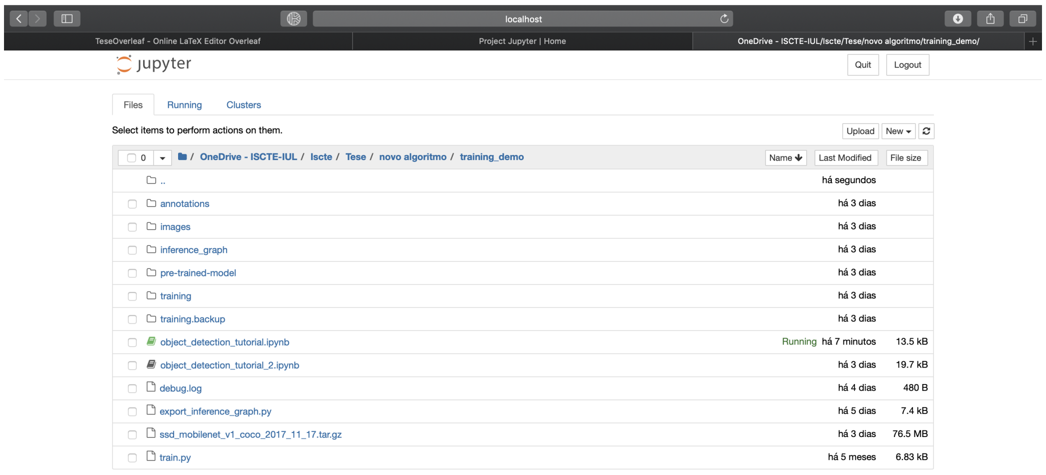
FIGURE 6.4: MAVProxy console flight log.

6.2 Fire Detection System

To test the fire detection system, two types of tests were developed: a test with the objective to assess the custom trained algorithm in various environments and situations and another test that aims to assess the developed script. These tests need to be done separately because in order to do them together it requires a real-life situation testing which is impossible due to the possibility of damaging the equipment. The fire detection script is tested using a real-life situation, however separated from the UAV, due to the previous motives.

6.2.1 Algorithm Accuracy

In order to test this algorithm, a tool known as *Jupyter Notebook* is used. This tool is an open-source web application that allows the creation of documents that contain live code, equations, visualizations and narrative text. It is used for a variety of purposes, however in this case it is used for implementation of machine learning techniques [108].



localhost
TeseOverleaf - Online LaTeX Editor Overleaf
Project Jupyter | Home
OneDrive - ISCTE-IUL/Iscte/Tese/novo algoritmo/training_demo/
jupyter
Quit
Logout
Files
Running
Clusters
Upload
New

Select items to perform actions on them.

/ OneDrive - ISCTE-IUL / Iscte / Tese / novo algoritmo / training_demo

Name	Last Modified	File size
..	há segundos	
annotations	há 3 dias	
images	há 3 dias	
inference_graph	há 3 dias	
pre-trained-model	há 3 dias	
training	há 3 dias	
training.backup	há 3 dias	
object_detection_tutorial.ipynb	Running há 7 minutos	13.5 kB
object_detection_tutorial_2.ipynb	há 3 dias	19.7 kB
debug.log	há 4 dias	480 B
export_inference_graph.py	há 5 dias	7.4 kB
ssd_mobilenet_v1_coco_2017_11_17.tar.gz	há 3 dias	76.5 MB
train.py	há 5 meses	6.83 kB

Figure 6.5: Jupyter Notebook browser view.

The tool uses the custom trained model to detect objects in images, more specifically fire. By using the exported inference graph, the label map and testing images, the model can be tested using the images as input. The program then loads all the variables and prints out all the given images with the fire object classified and located. An example of these testing outputs is shown in figure 6.6 and 6.7. The full notebook file and the image output can be found in section B.

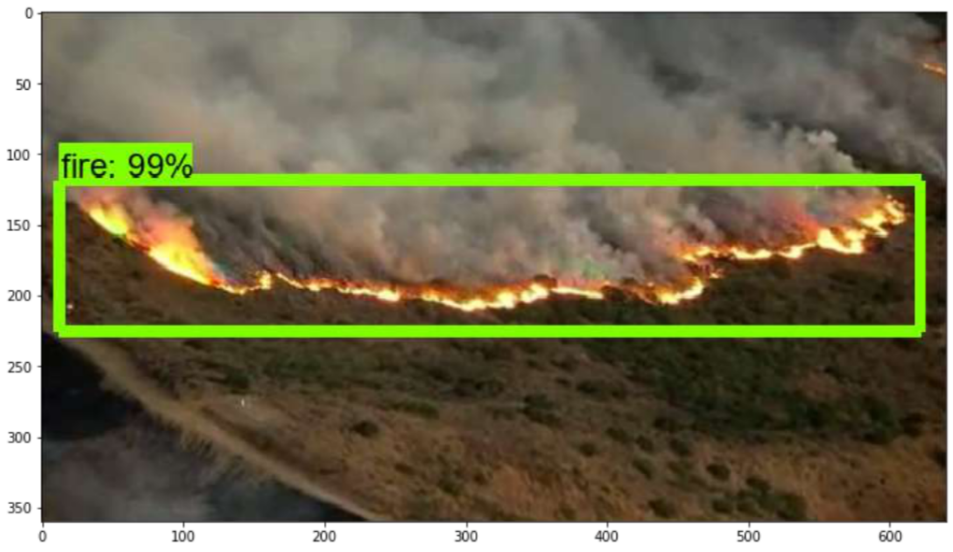


Figure 6.6: Fire detection first image output.

To test the algorithm even further, more images were added to the testing set, and the final results revealed a positive outcome from the 60 different images

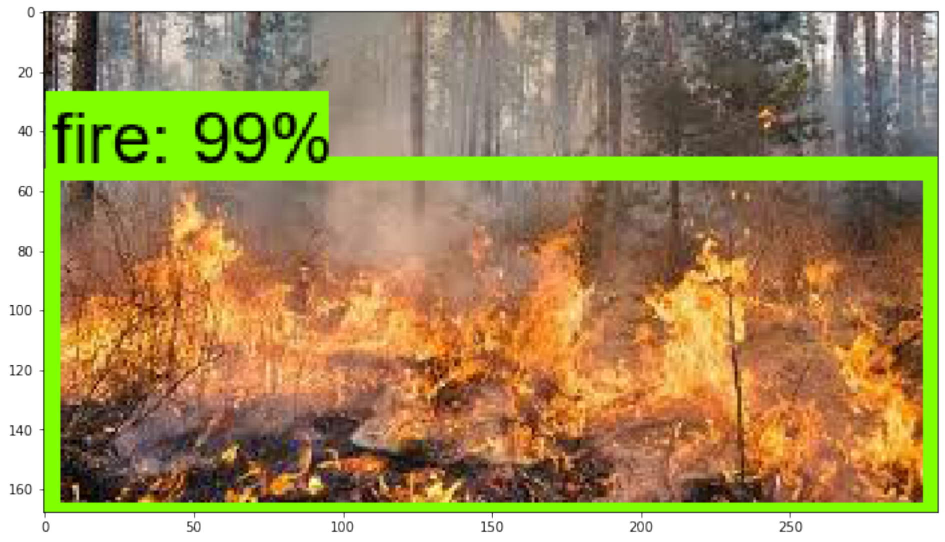


FIGURE 6.7: Fire detection second image output.

tested, against the 26 images previously defined. The graph in figure 6.8 shows that in 48 images the fire was classified correctly even if sometimes with over-classification (several fires detected where there is only one fire object). In 9 images, the fire was not detected when it should have (false negatives) and in 3 images the fire was detected incorrectly (false positives). This result revealed that the custom trained model detects fire successfully with about 80% accuracy.

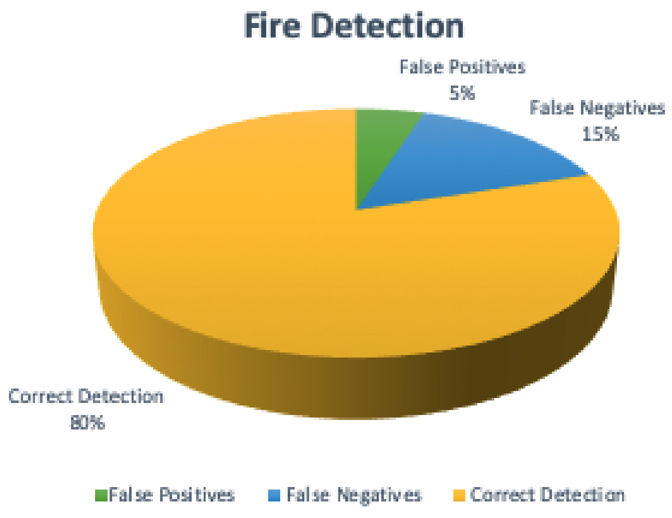


FIGURE 6.8: Fire detection statistical graph.

This results led to the conclusion that the algorithm fails in the detection of fire in situations where the fire is located across long distances, behind the smoke, where the forest environments are non-existing and where instead of one there are multiple small fires. The previous results can be improved by using a larger dataset with the specific scenarios. However, the final outcome is favorable considering the detection success rate.

6.2.2 Script Testing

In similarity to the testing done in section 6.1, the only way to test a program is to execute it and simulate the conditions where it is used. However, since recreating a forest fire is not possible and considered illegal at the time being, a small fire was created in an outside terrain and a camera was positioned at an altitude of 3 meters.

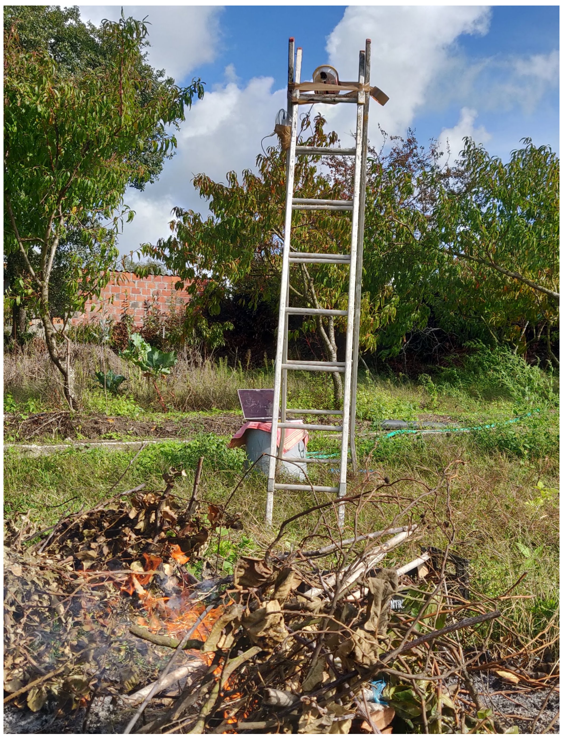
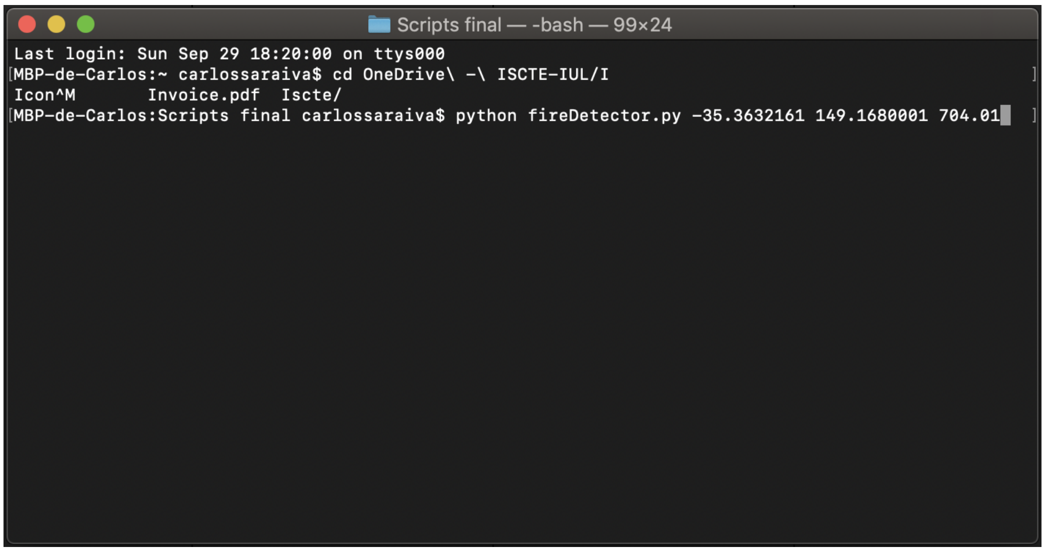


FIGURE 6.9: Executing the fire detection program.

In order to simulate the program's execution, when the command is written, the UAV's location is given as input in the command line. The input is written in a specific order, latitude, longitude and altitude. In this case, the location is the last waypoint of the flight simulation in section 6.1. The command in figure 6.9 start the program.



```
Last login: Sun Sep 29 18:20:00 on ttys000
[MBP-de-Carlos:~ carlossaraiva$ cd OneDrive\ -\ ISCTE-IUL/I                                        ]
 Icon^M       Invoice.pdf  Iscte/
[MBP-de-Carlos:Scripts final carlossaraiva$ python fireDetector.py -35.3632161 149.1680001 704.01  ]
```

FIGURE 6.10: Fire detection simulation environment.

When the program is running and, in case of the real-time video feed detecting fire, the fire object will be classified and located in the video output, as shown in figure 6.11.

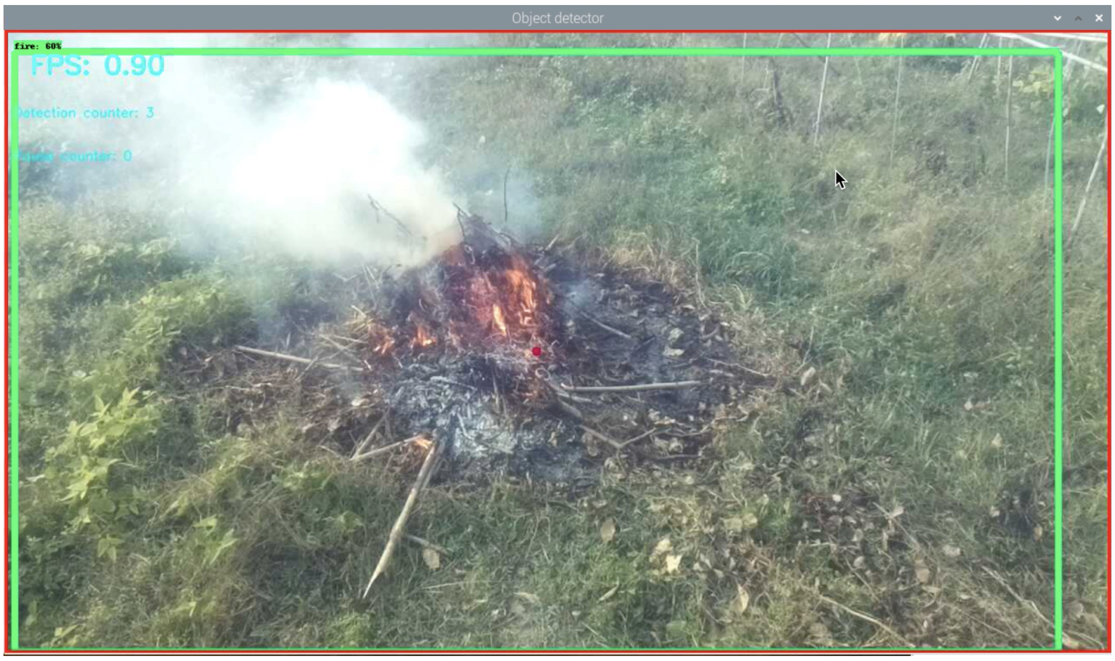
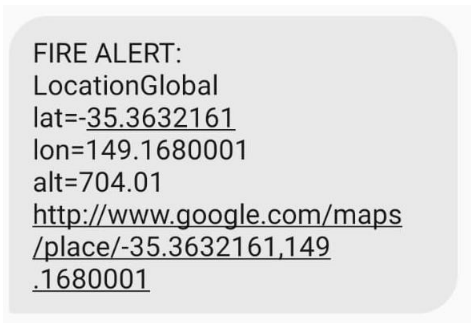


FIGURE 6.11: Successful fire detection.

If it is detected for more than 8 frames the notification via text message is sent to designated phone number.



FIRE ALERT:
LocationGlobal
lat=-35.3632161
lon=149.1680001
alt=704.01
http://www.google.com/maps
/place/-35.3632161,149
.1680001

Figure 6.12: Fire detection during simulation.

In case no fire is detected for more than 10 seconds, the program ends and kills all threads and existing processes.

The simulation was successful however, the program did not detect the fire at all times, which was expected because in sub-section 6.2.1 it was concluded that the algorithm performance is not optimal in situations with small fires. During the simulation, false positives were also detected. Such errors can be avoided by adding more unclassified images of fire to the dataset which will help the algorithm differentiate unclassified images from the classified ones.

6.3 High Precision Landing

In order to verify the possibility of integrating RTK technology for high precision landing, real-life tests are performed using the described components in section 5.1.3. The objective is to prove that during the landing procedure the UAV can in fact dock in the station with the maximum error of 3 cm. To test the components an UAV was assembled and the final set up is shown in figure 6.13.

Since the recharging station was not implemented, the landing pad is signaled with the heliport sign as in figure 6.14, that illustrates one of landing attempts.

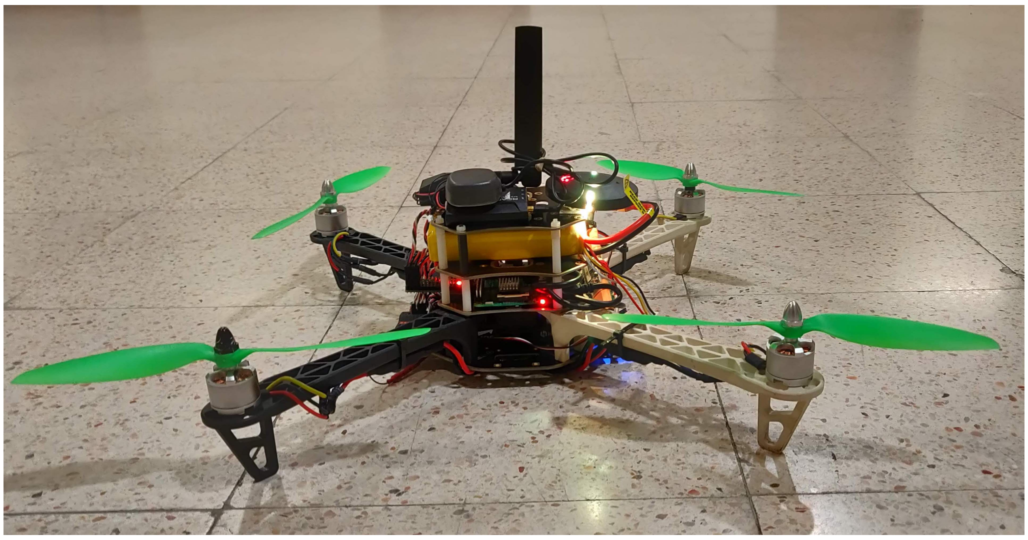


FIGURE 6.13: Final UAV build.

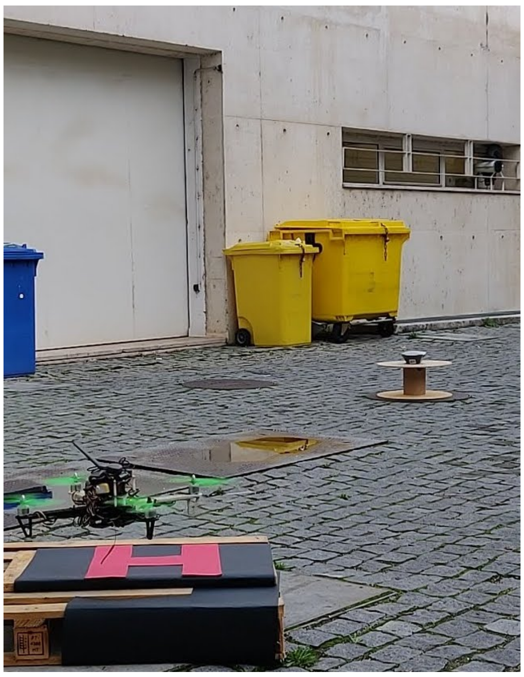
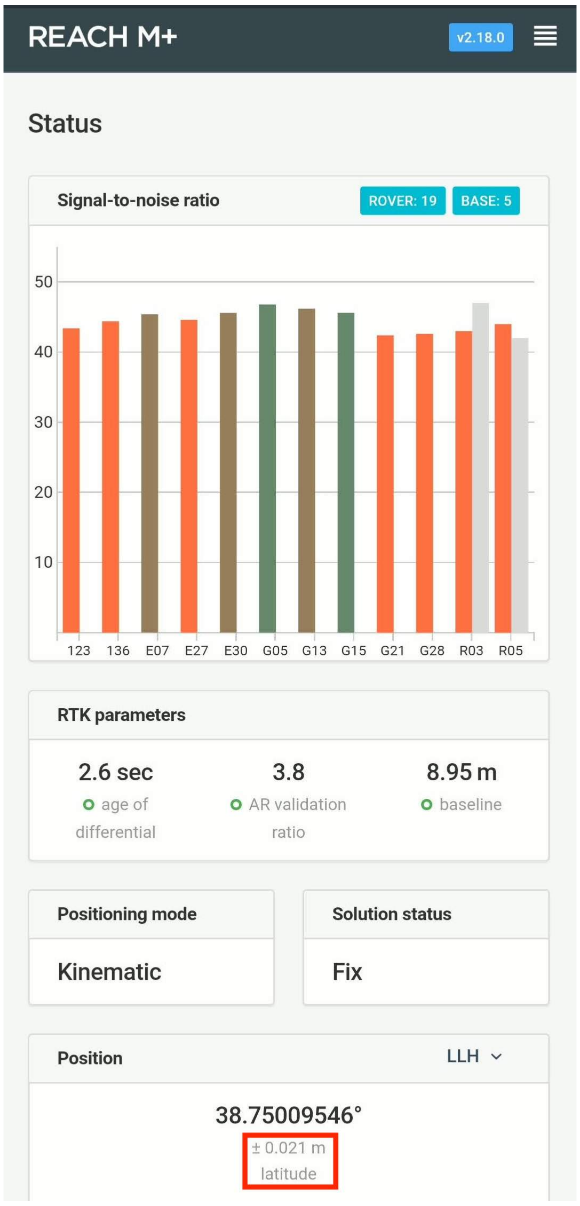


FIGURE 6.14: High precision landing testing.

During the experiment, a centimeter precision was achieved as can be seen in figure 6.15.

The experiment revealed that the UAV lands with a maximum error of 2.1 cm. Therefore the outcome of the experiment reveals that the RTK technology is accurate enough for high precision landing validating the final result of the whole system.



RTK parameters

2.6 sec age of differential

3.8 AR validation ratio

8.95 m baseline

Positioning mode: Kinematic

Solution status: Fix

Position LLH

38.75009546°
± 0.021 m
latitude

Figure 6.15: Reach precision output during experiment.

Chapter 7

Conclusion and Future Work

7.1 Main Conclusion

The purpose of this dissertation was to develop a fully autonomous system to surveil forests assuring its safety against forest fires using UAVs. The first goal, the flight automation, was successfully achieved using scripts written in python, with use of several different libraries. It has proven to be reliable after being tested in a flight simulator several times. The second and most important goal, the development of the forest fire detection system to deliver an early and accurate fire detection alarm, was also successful and it combined the use of convolutional neural networks and its training procedures with real-time video applications. This feature was successfully tested both in a real environment and with testing tools to provide an insight of the feature's accuracy and limitations. The final goal, the development of the wireless transfer solar powered station where the UAV would perform recharging procedures, was also achieved with the use of wireless power charging and RTK technology. This final step was only partially tested, proving the possibility of integrating the RTK technology for high accuracy landing. Throughout the development of this system several conclusions were taken:

- The use of python turned out to be advantageous for UAV automation due to the facility of finding useful python libraries;
- The fire detection algorithm proved to fail in situations where forest environments are non-existing and where instead of one there are multiple small fires;
- With the results presented and the use of low cost materials proved that proposed system was worth of applying for real environment situations;
- The RTK technology proved to be successful when used for high accuracy procedures;
- The low budget material proved to decrease performance in fire detection and in flight performance.

7.2 Future Work

After concluding this dissertation's research, some improvements can be considered and also future functionalities can be added to the system:

- The fire detection algorithm can be improved, not only by using a larger dataset but also integrating other sensors that allow for a more accurate detection;
- The companion computer, the Raspberry Pi, should be replaced by the Jetson TX board to allow for continuous detection and higher performance;
- The camera module should also be replaced in order to have higher image quality, therefore increasing the detection accuracy;
- A UAV with higher flight duration should also be used in order to increase mission radius;
- Optimize the UAV deployment by implementing an extra system to calculate the forest fire risk index and to predict the fire's advance;

- Implementation of swarm technology for UAV's coordinated behaviours in order to being able to use UAVs to patrol several areas simultaneously, enabling the use of nearby recharging stations.

Appendices

Appendix A

Flight Simulation Log File

```
[INFO]:  Connecting to vehicle on: 127.0.0.1:14550
[INFO]:  Connected
[DEBUG]: Starting system analysis
[INFO]:  Global Location:
[INFO]:  LocationGlobal:lat=0.0,lon=0.0,alt=None
[INFO]:  Global Location (relative altitude):
[INFO]:  LocationGlobalRelative:lat=0.0,lon=0.0,alt=0.0
[INFO]:  Local Location:
[INFO]: LocationLocal:north=None,east=None,down=None
[INFO]:  Attitude:
[INFO]:  Attitude:pitch=-0.000112996094686,
         yaw=-0.329702675343,roll=0.000109214859549
[INFO]:  Velocity: [0.0, 0.0, 0.0]
[INFO]:  GPS: GPSInfo:fix=0,num_sat=0
[INFO]:  Gimbal status: Gimbal: pitch=None, roll=None, yaw=None
[INFO]:  Battery: Battery:voltage=0.0,current=0.0,level=100
[INFO]:  EKF OK?: False
[INFO]:  Last Heartbeat: 0.75
[INFO]:  Rangefinder: Rangefinder: distance=None, voltage=None
[INFO]:  Rangefinder distance: None
```

```
[INFO]:  Rangefinder voltage: None
[INFO]:  Heading: 341
[INFO]:  Is Armable?: False
[INFO]:  System status: ACTIVE
[INFO]:  Groundspeed: 0.0
[INFO]:  Airspeed: 0.0
[INFO]:  Mode: STABILIZE
[INFO]:  Armed: False
[INFO]:  Vehicle EKF not ok
[DEBUG]:  Waiting EKF
[INFO]:  Vehicle EKF not ok
[DEBUG]:  Waiting EKF
[INFO]:  Vehicle EKF not ok
[DEBUG]:  Waiting EKF
[INFO]:  Vehicle EKF not ok
[DEBUG]:  Waiting EKF
[INFO]:  Vehicle EKF not ok
[DEBUG]:  Waiting EKF
[INFO]:  Vehicle EKF not ok
[DEBUG]:  Waiting EKF
[INFO]:  Vehicle EKF not ok
[DEBUG]:  Waiting EKF
[INFO]:  Vehicle EKF not ok
[DEBUG]:  Waiting EKF
[INFO]:  Vehicle EKF not ok
[DEBUG]:  Waiting EKF
[INFO]:  Vehicle EKF not ok
[DEBUG]:  Waiting EKF
[INFO]:  Create a new mission (for current location)
[DEBUG]:  Clear any existing commands
[DEBUG]:  Define/add new commands.
```

```
[DEBUG]:  Uploading new commands to vehicle
[DEBUG]:  Starting takeoff
[DEBUG]:  Arming motors
[DEBUG]:  Waiting for arming...
[DEBUG]:  Taking off!
[DEBUG]:  Altitude: 0.003
[DEBUG]:  Altitude: 0.003
[DEBUG]:  Altitude: 0.033
[DEBUG]:  Altitude: 1.458
[DEBUG]:  Altitude: 3.577
[DEBUG]:  Altitude: 5.834
[DEBUG]:  Altitude: 8.173
[DEBUG]:  Altitude: 10.564
[DEBUG]:  Altitude: 12.991
[DEBUG]:  Altitude: 15.443
[DEBUG]:  Altitude: 17.909
[DEBUG]:  Altitude: 20.381
[DEBUG]:  Altitude: 22.859
[DEBUG]:  Altitude: 25.343
[DEBUG]:  Altitude: 27.831
[DEBUG]:  Altitude: 30.32
[DEBUG]:  Altitude: 32.807
[DEBUG]:  Altitude: 35.3
[DEBUG]:  Altitude: 37.793
[DEBUG]:  Altitude: 40.286
[DEBUG]:  Altitude: 42.786
[DEBUG]:  Altitude: 45.281
[DEBUG]:  Altitude: 47.775
[DEBUG]:  Altitude: 50.271
[DEBUG]:  Altitude: 52.769
[DEBUG]:  Altitude: 55.265
```

```
[DEBUG]:  Altitude: 57.761
[DEBUG]:  Altitude: 60.257
[DEBUG]:  Altitude: 62.755
[DEBUG]:  Altitude: 65.253
[DEBUG]:  Altitude: 67.75
[DEBUG]:  Altitude: 70.25
[DEBUG]:  Altitude: 72.752
[DEBUG]:  Altitude: 74.628
[DEBUG]:  Altitude: 77.167
[DEBUG]:  Altitude: 79.667
[DEBUG]:  Altitude: 82.168
[DEBUG]:  Altitude: 84.667
[DEBUG]:  Altitude: 87.168
[DEBUG]:  Altitude: 89.668
[DEBUG]:  Altitude: 92.17
[DEBUG]:  Altitude: 94.673
[DEBUG]:  Altitude: 97.175
[DEBUG]:  Altitude: 99.679
[DEBUG]:  Altitude: 102.183
[DEBUG]:  Altitude: 104.688
[DEBUG]:  Altitude: 107.781
[DEBUG]:  Altitude: 109.656
[DEBUG]:  Altitude: 112.154
[DEBUG]:  Altitude: 114.651
[DEBUG]: Reached target altitude
[INFO]:  Take off complete
[INFO]: Starting mission
[DEBUG]: Distance to waypoint (0): None
[INFO]: Running fire dection script at lat=-35.3632607
        lon=149.1652293 alt=698.66
[DEBUG]: Distance to waypoint (2): 131.195474933
```

```
[DEBUG]: Distance to waypoint (2): 129.258487301
[DEBUG]: Distance to waypoint (2): 127.232487179
[DEBUG]: Distance to waypoint (2): 125.139696346
[DEBUG]: Distance to waypoint (2): 122.980114879
[DEBUG]: Distance to waypoint (2): 122.980114879
[DEBUG]: Distance to waypoint (2): 119.807534959
[DEBUG]: Distance to waypoint (2): 117.547768357
[DEBUG]: Distance to waypoint (2): 115.254606961
[DEBUG]: Distance to waypoint (2): 112.950314544
[DEBUG]: Distance to waypoint (2): 110.61262749
[DEBUG]: Distance to waypoint (2): 108.252677732
[DEBUG]: Distance to waypoint (2): 105.881597197
[DEBUG]: Distance to waypoint (2): 103.488254147
[DEBUG]: Distance to waypoint (2): 101.094912339
[DEBUG]: Distance to waypoint (2): 97.3212839615
[DEBUG]: Distance to waypoint (2): 94.9056836266
[DEBUG]: Distance to waypoint (2): 92.4678214323
[DEBUG]: Distance to waypoint (2): 90.0299611256
[DEBUG]: Distance to waypoint (2): 87.5809710961
[DEBUG]: Distance to waypoint (2): 85.1319185334
[DEBUG]: Distance to waypoint (2): 82.6717995142
[DEBUG]: Distance to waypoint (2): 80.2005513252
[DEBUG]: Distance to waypoint (2): 77.7293059785
[DEBUG]: Distance to waypoint (2): 75.2580637556
[DEBUG]: Distance to waypoint (2): 72.7868249739
[DEBUG]: Distance to waypoint (2): 70.3044583154
[DEBUG]: Distance to waypoint (2): 67.8221772251
[DEBUG]: Distance to waypoint (2): 65.3398226616
[DEBUG]: Distance to waypoint (2): 62.8574736488
[DEBUG]: Distance to waypoint (2): 60.3639993062
[DEBUG]: Distance to waypoint (2): 57.8816636026
```

```
[DEBUG]: Distance to waypoint (2): 55.3882043931
[DEBUG]: Distance to waypoint (2): 52.8947543838
[DEBUG]: Distance to waypoint (2): 50.4013149419
[DEBUG]: Distance to waypoint (2): 47.896756375
[DEBUG]: Distance to waypoint (2): 45.4033434468
[DEBUG]: Distance to waypoint (2): 42.9099472394
[DEBUG]: Distance to waypoint (2): 40.4164344066
[DEBUG]: Distance to waypoint (2): 37.9119417462
[DEBUG]: Distance to waypoint (2): 35.4186077124
[DEBUG]: Distance to waypoint (2): 32.9253074182
[DEBUG]: Distance to waypoint (2): 30.4209186505
[DEBUG]: Distance to waypoint (2): 27.927713946
[DEBUG]: Distance to waypoint (2): 25.4234483893
[DEBUG]: Distance to waypoint (2): 22.9304047297
[DEBUG]: Distance to waypoint (2): 20.4260936194
[DEBUG]: Distance to waypoint (2): 17.9333168145
[DEBUG]: Distance to waypoint (2): 15.4296653015
[DEBUG]: Distance to waypoint (2): 12.9597886765
[DEBUG]: Distance to waypoint (2): 10.5689322873
[DEBUG]: Distance to waypoint (2): 8.27889665931
[DEBUG]: Distance to waypoint (2): 6.16969424305
[DEBUG]: Distance to waypoint (2): 4.28758102529
[DEBUG]: Distance to waypoint (2): 2.68313362062
[INFO]: Running fire dection script at lat=-35.3619731
        lon=149.1652346 alt=704.01
[DEBUG]: Distance to waypoint (3): 115.268291138
[DEBUG]: Distance to waypoint (3): 112.941731304
[DEBUG]: Distance to waypoint (3): 110.604040357
[DEBUG]: Distance to waypoint (3): 108.255264715
[DEBUG]: Distance to waypoint (3): 105.884180911
[DEBUG]: Distance to waypoint (3): 103.490884102
```

```
[DEBUG]: Distance to waypoint (3): 101.086460123
[DEBUG]: Distance to waypoint (3): 98.681986538
[DEBUG]: Distance to waypoint (3): 96.2663825203
[DEBUG]: Distance to waypoint (3): 93.8285163916
[DEBUG]: Distance to waypoint (3): 91.3906519126
[DEBUG]: Distance to waypoint (3): 89.3312700617
[DEBUG]: Distance to waypoint (3): 86.8822772119
[DEBUG]: Distance to waypoint (3): 84.4221546887
[DEBUG]: Distance to waypoint (3): 81.9620344715
[DEBUG]: Distance to waypoint (3): 79.5019167753
[DEBUG]: Distance to waypoint (3): 77.0306017366
[DEBUG]: Distance to waypoint (3): 74.5593558661
[DEBUG]: Distance to waypoint (3): 72.0769815049
[DEBUG]: Distance to waypoint (3): 69.5945368505
[DEBUG]: Distance to waypoint (3): 67.1121673593
[DEBUG]: Distance to waypoint (3): 64.6186704474
[DEBUG]: Distance to waypoint (3): 62.1363099936
[DEBUG]: Distance to waypoint (3): 59.6427391278
[DEBUG]: Distance to waypoint (3): 57.1603864656
[DEBUG]: Distance to waypoint (3): 54.6669087763
[DEBUG]: Distance to waypoint (3): 52.1734386497
[DEBUG]: Distance to waypoint (3): 49.6799772237
[DEBUG]: Distance to waypoint (3): 47.1866322376
[DEBUG]: Distance to waypoint (3): 44.6931985921
[DEBUG]: Distance to waypoint (3): 42.1997795005
[DEBUG]: Distance to waypoint (3): 39.7063777037
[DEBUG]: Distance to waypoint (3): 37.2131348758
[DEBUG]: Distance to waypoint (3): 34.7086580835
[DEBUG]: Distance to waypoint (3): 32.2153450373
[DEBUG]: Distance to waypoint (3): 29.7109423166
[DEBUG]: Distance to waypoint (3): 27.2177225647
```

```
[DEBUG]: Distance to waypoint (3): 24.7134408344
[DEBUG]: Distance to waypoint (3): 22.2203803861
[DEBUG]: Distance to waypoint (3): 19.7163165821
[DEBUG]: Distance to waypoint (3): 17.2235624813
[DEBUG]: Distance to waypoint (3): 14.731080088
[DEBUG]: Distance to waypoint (3): 12.2835308188
[DEBUG]: Distance to waypoint (3): 9.91455443863
[DEBUG]: Distance to waypoint (3): 7.69135794599
[DEBUG]: Distance to waypoint (3): 5.66121617438
[DEBUG]: Distance to waypoint (3): 3.88021506499
[DEBUG]: Distance to waypoint (3): 2.39418609838
[INFO]: Running fire dection script at lat=-35.3607038
        lon=149.1652347 alt=704.01
[DEBUG]: Distance to waypoint (4): 113.868243554
[DEBUG]: Distance to waypoint (4): 111.541721264
[DEBUG]: Distance to waypoint (4): 109.19293873
[DEBUG]: Distance to waypoint (4): 106.82189623
[DEBUG]: Distance to waypoint (4): 104.439725921
[DEBUG]: Distance to waypoint (4): 102.046428116
[DEBUG]: Distance to waypoint (4): 99.6308197124
[DEBUG]: Distance to waypoint (4): 97.2152654395
[DEBUG]: Distance to waypoint (4): 94.7885291236
[DEBUG]: Distance to waypoint (4): 92.3507195412
[DEBUG]: Distance to waypoint (4): 89.9128561273
[DEBUG]: Distance to waypoint (4): 87.4638628048
[DEBUG]: Distance to waypoint (4): 85.0037397641
[DEBUG]: Distance to waypoint (4): 82.543618984
[DEBUG]: Distance to waypoint (4): 80.0835006711
[DEBUG]: Distance to waypoint (4): 77.6122533231
[DEBUG]: Distance to waypoint (4): 75.1409388809
[DEBUG]: Distance to waypoint (4): 72.6696954476
```

```
[DEBUG]: Distance to waypoint (4): 70.1761920901
[DEBUG]: Distance to waypoint (4): 67.7048800829
[DEBUG]: Distance to waypoint (4): 65.211382108
[DEBUG]: Distance to waypoint (4): 62.7178087891
[DEBUG]: Distance to waypoint (4): 60.2243174361
[DEBUG]: Distance to waypoint (4): 57.7308317377
[DEBUG]: Distance to waypoint (4): 55.2373524585
[DEBUG]: Distance to waypoint (4): 52.7550120626
[DEBUG]: Distance to waypoint (4): 50.2504169806
[DEBUG]: Distance to waypoint (4): 47.7680946582
[DEBUG]: Distance to waypoint (4): 45.2635207527
[DEBUG]: Distance to waypoint (4): 42.7812229941
[DEBUG]: Distance to waypoint (4): 40.2878096234
[DEBUG]: Distance to waypoint (4): 37.7944151782
[DEBUG]: Distance to waypoint (4): 35.3010436686
[DEBUG]: Distance to waypoint (4): 32.7965693997
[DEBUG]: Distance to waypoint (4): 30.3032613456
[DEBUG]: Distance to waypoint (4): 29.045492685
[DEBUG]: Distance to waypoint (4): 25.3054685726
[DEBUG]: Distance to waypoint (4): 22.8125334534
[DEBUG]: Distance to waypoint (4): 20.2972514552
[DEBUG]: Distance to waypoint (4): 17.8043788006
[DEBUG]: Distance to waypoint (4): 15.3002817925
[DEBUG]: Distance to waypoint (4): 12.8416823497
[DEBUG]: Distance to waypoint (4): 10.4500818554
[DEBUG]: Distance to waypoint (4): 8.18241214838
[DEBUG]: Distance to waypoint (4): 6.08372673857
[DEBUG]: Distance to waypoint (4): 4.22536633168
[DEBUG]: Distance to waypoint (4): 2.65574542218
[INFO]: Running fire dection script at lat=-35.3594396
        lon=149.1652346 alt=704.02
```

```
[DEBUG]: Distance to waypoint (5): 274.269157542

[DEBUG]: Distance to waypoint (5): 271.352597819

[DEBUG]: Distance to waypoint (5): 268.413602482

[DEBUG]: Distance to waypoint (5): 265.44121616

[DEBUG]: Distance to waypoint (5): 262.468830564

[DEBUG]: Distance to waypoint (5): 259.474006739

[DEBUG]: Distance to waypoint (5): 256.468052429

[DEBUG]: Distance to waypoint (5): 253.450967637

[DEBUG]: Distance to waypoint (5): 250.422752377

[DEBUG]: Distance to waypoint (5): 247.383228553

[DEBUG]: Distance to waypoint (5): 244.35483547

[DEBUG]: Distance to waypoint (5): 241.304358981

[DEBUG]: Distance to waypoint (5): 238.264835455

[DEBUG]: Distance to waypoint (5): 235.214181607

[DEBUG]: Distance to waypoint (5): 232.163527976

[DEBUG]: Distance to waypoint (5): 229.101744016

[DEBUG]: Distance to waypoint (5): 226.051090839

[DEBUG]: Distance to waypoint (5): 223.000437907

[DEBUG]: Distance to waypoint (5): 219.949785229

[DEBUG]: Distance to waypoint (5): 216.876871662

[DEBUG]: Distance to waypoint (5): 213.826394273

[DEBUG]: Distance to waypoint (5): 210.753480841

[DEBUG]: Distance to waypoint (5): 207.702828845

[DEBUG]: Distance to waypoint (5): 204.641046559

[DEBUG]: Distance to waypoint (5): 201.579264571

[DEBUG]: Distance to waypoint (5): 198.506352292

[DEBUG]: Distance to waypoint (5): 195.444570945

[DEBUG]: Distance to waypoint (5): 192.382789944

[DEBUG]: Distance to waypoint (5): 189.321181155

[DEBUG]: Distance to waypoint (5): 186.248269767

[DEBUG]: Distance to waypoint (5): 183.186489381
```

```
[DEBUG]: Distance to waypoint (5): 180.124709397
[DEBUG]: Distance to waypoint (5): 177.063100286
[DEBUG]: Distance to waypoint (5): 173.990189961
[DEBUG]: Distance to waypoint (5): 170.92841072
[DEBUG]: Distance to waypoint (5): 167.855332557
[DEBUG]: Distance to waypoint (5): 164.793723012
[DEBUG]: Distance to waypoint (5): 161.731945265
[DEBUG]: Distance to waypoint (5): 158.670168072
[DEBUG]: Distance to waypoint (5): 155.597260766
[DEBUG]: Distance to waypoint (5): 152.535484783
[DEBUG]: Distance to waypoint (5): 149.462578736
[DEBUG]: Distance to waypoint (5): 146.400804112
[DEBUG]: Distance to waypoint (5): 143.328063372
[DEBUG]: Distance to waypoint (5): 139.798634549
[DEBUG]: Distance to waypoint (5): 139.798634549
[DEBUG]: Distance to waypoint (5): 136.737024499
[DEBUG]: Distance to waypoint (5): 130.613479825
[DEBUG]: Distance to waypoint (5): 130.613479825
[DEBUG]: Distance to waypoint (5): 124.935330864
[DEBUG]: Distance to waypoint (5): 121.873561672
[DEBUG]: Distance to waypoint (5): 118.800662829
[DEBUG]: Distance to waypoint (5): 115.739052998
[DEBUG]: Distance to waypoint (5): 112.666155548
[DEBUG]: Distance to waypoint (5): 109.593415027
[DEBUG]: Distance to waypoint (5): 106.531650034
[DEBUG]: Distance to waypoint (5): 103.458755756
[DEBUG]: Distance to waypoint (5): 100.385863198
[DEBUG]: Distance to waypoint (5): 97.3131234828
[DEBUG]: Distance to waypoint (5): 94.2402332127
[DEBUG]: Distance to waypoint (5): 91.1786249605
[DEBUG]: Distance to waypoint (5): 88.1057374234
```

```
[DEBUG]: Distance to waypoint (5): 85.0441299153
[DEBUG]: Distance to waypoint (5): 81.9711011995
[DEBUG]: Distance to waypoint (5): 78.8983643502
[DEBUG]: Distance to waypoint (5): 75.8254863782
[DEBUG]: Distance to waypoint (5): 72.7637432428
[DEBUG]: Distance to waypoint (5): 69.7020044217
[DEBUG]: Distance to waypoint (5): 66.6402705029
[DEBUG]: Distance to waypoint (5): 63.556279792
[DEBUG]: Distance to waypoint (5): 60.4946839543
[DEBUG]: Distance to waypoint (5): 57.4218341923
[DEBUG]: Distance to waypoint (5): 54.3601239489
[DEBUG]: Distance to waypoint (5): 51.2872921206
[DEBUG]: Distance to waypoint (5): 48.2256032556
[DEBUG]: Distance to waypoint (5): 45.1639284016
[DEBUG]: Distance to waypoint (5): 42.1022706123
[DEBUG]: Distance to waypoint (5): 39.0295897661
[DEBUG]: Distance to waypoint (5): 35.9679696763
[DEBUG]: Distance to waypoint (5): 32.8952498698
[DEBUG]: Distance to waypoint (5): 29.8337036173
[DEBUG]: Distance to waypoint (5): 26.7610817289
[DEBUG]: Distance to waypoint (5): 23.6996882956
[DEBUG]: Distance to waypoint (5): 20.6272385747
[DEBUG]: Distance to waypoint (5): 17.5661112129
[DEBUG]: Distance to waypoint (5): 14.4941330112
[DEBUG]: Distance to waypoint (6): 12.5703001955
[DEBUG]: Distance to waypoint (6): 10.0194751381
[DEBUG]: Distance to waypoint (6): 7.85858158935
[DEBUG]: Distance to waypoint (6): 6.10591790313
[DEBUG]: Distance to waypoint (6): 4.76903804325
[DEBUG]: Distance to waypoint (6): 3.7402907689
[DEBUG]: Distance to waypoint (6): 2.85856282386
```

```
[INFO]: Running fire dection script at lat=-35.3593908
        lon=149.1679985 alt=704.02
[DEBUG]: Distance to waypoint (7): 117.205955361
[DEBUG]: Distance to waypoint (7): 114.923797479
[DEBUG]: Distance to waypoint (7): 112.608346967
[DEBUG]: Distance to waypoint (7): 110.270429848
[DEBUG]: Distance to waypoint (7): 107.921584716
[DEBUG]: Distance to waypoint (7): 105.550374894
[DEBUG]: Distance to waypoint (7): 103.168136322
[DEBUG]: Distance to waypoint (7): 100.785898918
[DEBUG]: Distance to waypoint (7): 98.336874089
[DEBUG]: Distance to waypoint (7): 95.9212450646
[DEBUG]: Distance to waypoint (7): 93.5058376391
[DEBUG]: Distance to waypoint (7): 91.0790835511
[DEBUG]: Distance to waypoint (7): 88.6414322419
[DEBUG]: Distance to waypoint (7): 86.2036747621
[DEBUG]: Distance to waypoint (7): 83.7437879712
[DEBUG]: Distance to waypoint (7): 81.2949201084
[DEBUG]: Distance to waypoint (7): 78.8239344147
[DEBUG]: Distance to waypoint (7): 76.3641012843
[DEBUG]: Distance to waypoint (7): 73.8930139011
[DEBUG]: Distance to waypoint (7): 71.4220925999
[DEBUG]: Distance to waypoint (7): 68.9399098066
[DEBUG]: Distance to waypoint (7): 66.457745095
[DEBUG]: Distance to waypoint (7): 63.975772043
[DEBUG]: Distance to waypoint (7): 61.482526568
[DEBUG]: Distance to waypoint (7): 59.0004377901
[DEBUG]: Distance to waypoint (7): 56.5183785333
[DEBUG]: Distance to waypoint (7): 54.0363528647
[DEBUG]: Distance to waypoint (7): 51.5432357426
[DEBUG]: Distance to waypoint (7): 49.0501633195
```

```
[DEBUG]: Distance to waypoint (7): 46.5571427778
[DEBUG]: Distance to waypoint (7): 44.0641829225
[DEBUG]: Distance to waypoint (7): 41.5710307858
[DEBUG]: Distance to waypoint (7): 39.0782110087
[DEBUG]: Distance to waypoint (7): 36.5851952232
[DEBUG]: Distance to waypoint (7): 34.0925767077
[DEBUG]: Distance to waypoint (7): 31.610893774
[DEBUG]: Distance to waypoint (7): 29.1185794306
[DEBUG]: Distance to waypoint (7): 26.626497741
[DEBUG]: Distance to waypoint (7): 24.1240420594
[DEBUG]: Distance to waypoint (7): 21.6433537794
[DEBUG]: Distance to waypoint (7): 19.1531165645
[DEBUG]: Distance to waypoint (7): 16.662585267
[DEBUG]: Distance to waypoint (7): 14.1849809803
[DEBUG]: Distance to waypoint (7): 11.7537980341
[DEBUG]: Distance to waypoint (7): 9.39143384147
[DEBUG]: Distance to waypoint (7): 7.17842750819
[DEBUG]: Distance to waypoint (7): 5.15953057788
[DEBUG]: Distance to waypoint (7): 3.39501095383
[DEBUG]: Distance to waypoint (7): 1.94780587042
[INFO]: Running fire dection script at lat=-35.3606668
        lon=149.1680018 alt=704.01
[DEBUG]: Distance to waypoint (8): 114.424558865
[DEBUG]: Distance to waypoint (8): 112.109009024
[DEBUG]: Distance to waypoint (8): 109.782327708
[DEBUG]: Distance to waypoint (8): 107.422251725
[DEBUG]: Distance to waypoint (8): 105.051044254
[DEBUG]: Distance to waypoint (8): 102.668705292
[DEBUG]: Distance to waypoint (8): 100.27532323
[DEBUG]: Distance to waypoint (8): 97.8597692046
[DEBUG]: Distance to waypoint (8): 95.444126321
```

```
[DEBUG]: Distance to waypoint (8): 93.0174466598

[DEBUG]: Distance to waypoint (8): 90.5795435662

[DEBUG]: Distance to waypoint (8): 88.1306077758

[DEBUG]: Distance to waypoint (8): 85.6816773138

[DEBUG]: Distance to waypoint (8): 83.2216211894

[DEBUG]: Distance to waypoint (8): 80.7616796633

[DEBUG]: Distance to waypoint (8): 78.3016403123

[DEBUG]: Distance to waypoint (8): 75.8304775056

[DEBUG]: Distance to waypoint (8): 73.3594442842

[DEBUG]: Distance to waypoint (8): 70.8771728975

[DEBUG]: Distance to waypoint (8): 69.6416069128

[DEBUG]: Distance to waypoint (8): 65.9237959276

[DEBUG]: Distance to waypoint (8): 63.4417035294

[DEBUG]: Distance to waypoint (8): 60.9483598152

[DEBUG]: Distance to waypoint (8): 58.4661651528

[DEBUG]: Distance to waypoint (8): 57.2195098979

[DEBUG]: Distance to waypoint (8): 54.7373469415

[DEBUG]: Distance to waypoint (8): 52.2440782632

[DEBUG]: Distance to waypoint (8): 49.750838466

[DEBUG]: Distance to waypoint (8): 47.25763212

[DEBUG]: Distance to waypoint (8): 44.7644648146

[DEBUG]: Distance to waypoint (8): 42.2713434576

[DEBUG]: Distance to waypoint (8): 39.7782766886

[DEBUG]: Distance to waypoint (8): 37.2850344146

[DEBUG]: Distance to waypoint (8): 34.8029698613

[DEBUG]: Distance to waypoint (8): 32.3101062604

[DEBUG]: Distance to waypoint (8): 29.8173585849

[DEBUG]: Distance to waypoint (8): 27.3244387229

[DEBUG]: Distance to waypoint (8): 24.8319987105

[DEBUG]: Distance to waypoint (8): 22.3509331297

[DEBUG]: Distance to waypoint (8): 19.8590837152
```

```
[DEBUG]: Distance to waypoint (8): 17.3566036879
[DEBUG]: Distance to waypoint (8): 14.8771014506
[DEBUG]: Distance to waypoint (8): 12.4320988442
[DEBUG]: Distance to waypoint (8): 10.0778555398
[DEBUG]: Distance to waypoint (8): 7.83676924953
[DEBUG]: Distance to waypoint (8): 5.79893003811
[DEBUG]: Distance to waypoint (8): 4.00222568433
[DEBUG]: Distance to waypoint (8): 2.49430287438
[INFO]: Running fire dection script at lat=-35.3619279
        lon=149.1680009 alt=704.01
[DEBUG]: Distance to waypoint (9): 115.11608645
[DEBUG]: Distance to waypoint (9): 112.811677872
[DEBUG]: Distance to waypoint (9): 110.473944756
[DEBUG]: Distance to waypoint (9): 108.125080044
[DEBUG]: Distance to waypoint (9): 105.753881107
[DEBUG]: Distance to waypoint (9): 103.371621097
[DEBUG]: Distance to waypoint (9): 100.978301343
[DEBUG]: Distance to waypoint (9): 99.7760038148
[DEBUG]: Distance to waypoint (9): 96.1581245982
[DEBUG]: Distance to waypoint (9): 93.7314137485
[DEBUG]: Distance to waypoint (9): 91.2934973758
[DEBUG]: Distance to waypoint (9): 88.8556587737
[DEBUG]: Distance to waypoint (9): 86.406691809
[DEBUG]: Distance to waypoint (9): 85.1822096643
[DEBUG]: Distance to waypoint (9): 82.7333329127
[DEBUG]: Distance to waypoint (9): 80.2732466003
[DEBUG]: Distance to waypoint (9): 77.8020334284
[DEBUG]: Distance to waypoint (9): 75.3308254868
[DEBUG]: Distance to waypoint (9): 72.8484917816
[DEBUG]: Distance to waypoint (9): 70.3772960045
[DEBUG]: Distance to waypoint (9): 67.8949758319
```

```
[DEBUG]: Distance to waypoint (9): 65.4126636051
[DEBUG]: Distance to waypoint (9): 62.9303602634
[DEBUG]: Distance to waypoint (9): 60.4369355644
[DEBUG]: Distance to waypoint (9): 57.9546535179
[DEBUG]: Distance to waypoint (9): 55.461253031
[DEBUG]: Distance to waypoint (9): 52.9789984119
[DEBUG]: Distance to waypoint (9): 50.4856294583
[DEBUG]: Distance to waypoint (9): 48.0034110013
[DEBUG]: Distance to waypoint (9): 45.5100841866
[DEBUG]: Distance to waypoint (9): 43.0167840338
[DEBUG]: Distance to waypoint (9): 40.5235154642
[DEBUG]: Distance to waypoint (9): 38.0302846896
[DEBUG]: Distance to waypoint (9): 35.5370996641
[DEBUG]: Distance to waypoint (9): 33.0551006414
[DEBUG]: Distance to waypoint (9): 30.5620412145
[DEBUG]: Distance to waypoint (9): 28.0690701487
[DEBUG]: Distance to waypoint (9): 25.5762132821
[DEBUG]: Distance to waypoint (9): 23.072379875
[DEBUG]: Distance to waypoint (9): 20.5798813957
[DEBUG]: Distance to waypoint (9): 18.0876746568
[DEBUG]: Distance to waypoint (9): 15.5958995172
[DEBUG]: Distance to waypoint (9): 13.1376257622
[DEBUG]: Distance to waypoint (9): 10.0808384039
[DEBUG]: Distance to waypoint (9): 8.480676772
[DEBUG]: Distance to waypoint (9): 6.38298361538
[DEBUG]: Distance to waypoint (9): 4.51206526061
[DEBUG]: Distance to waypoint (9): 2.91503088543
[DEBUG]: Distance to waypoint (9): 1.64284659915
[DEBUG]: Distance to waypoint (9): 0.786232321484
[DEBUG]: Distance to waypoint (9): 0.534765072211
[DEBUG]: Distance to waypoint (9): 0.593682626936
```

```
[DEBUG]: Distance to waypoint (9): 0.612537150277
[DEBUG]: Distance to waypoint (9): 0.578633873118
[DEBUG]: Distance to waypoint (9): 0.539415886783
[DEBUG]: Distance to waypoint (9): 0.527946651012
[DEBUG]: Distance to waypoint (9): 0.545744546681
[DEBUG]: Distance to waypoint (9): 0.558513933201
[DEBUG]: Distance to waypoint (9): 0.58277793585
[DEBUG]: Distance to waypoint (9): 0.593841648196
[DEBUG]: Distance to waypoint (9): 0.603812446962
[DEBUG]: Distance to waypoint (9): 0.60292060704
[DEBUG]: Distance to waypoint (9): 0.602233249089
[DEBUG]: Distance to waypoint (9): 0.590625713663
[DEBUG]: Distance to waypoint (9): 0.590344003789
[DEBUG]: Distance to waypoint (9): 0.590344003789
[DEBUG]: Distance to waypoint (9): 0.590272134364
[DEBUG]: Distance to waypoint (9): 0.590272134364
[DEBUG]: Distance to waypoint (10): 0.590272134364
[INFO]: Running fire dection script at lat=-35.3632161
        lon=149.1680001 alt=704.01
[INFO]: Exit 'standard' mission
[INFO]: Return to launch
[DEBUG]: Close vehicle object
```

Appendix B

Fire Detection Algorithm Testing Output

Imports

In [12]:
```
import numpy as np
import os
import six.moves.urllib as urllib
import sys
import tarfile
import tensorflow as tf
import zipfile

from distutils.version import StrictVersion
from collections import defaultdict
from io import StringIO
from matplotlib import pyplot as plt
from PIL import Image

# This is needed since the notebook is stored in the object_detection folder.
sys.path.append("..")
from object_detection.utils import ops as utils_ops

if StrictVersion(tf.__version__) < StrictVersion('1.12.0'):
  raise ImportError('Please upgrade your TensorFlow installation to v1.12.*.')
```

Object detection imports

Here are the imports from the object detection module.

In [13]:
```
%matplotlib inline

from object_detection.utils import label_map_util

from object_detection.utils import visualization_utils as vis_util
```

Model preparation

Variables

Any model exported using the `export_inference_graph.py` tool can be loaded here simply by changing `PATH_TO_FROZEN_GRAPH` to point to a new .pb file.

In [14]:
```
MODEL_NAME = 'inference_graph'
PATH_TO_FROZEN_GRAPH = MODEL_NAME + '/frozen_inference_graph.pb'
PATH_TO_LABELS = 'annotations/label_map.pbtxt'
```

Load a (frozen) Tensorflow model into memory.

In [15]:
```
detection_graph = tf.Graph()
with detection_graph.as_default():
  od_graph_def = tf.GraphDef()
  with tf.gfile.GFile(PATH_TO_FROZEN_GRAPH, 'rb') as fid:
    serialized_graph = fid.read()
    od_graph_def.ParseFromString(serialized_graph)
    tf.import_graph_def(od_graph_def, name='')
```

Loading label map

Label maps map indices to category names.

```
In [16]: category_index = label_map_util.create_category_index_from_labelmap(PATH_TO_LABELS, use_display_name
=True)
```

Helper code

```
In [17]: def load_image_into_numpy_array(image):
           (im_width, im_height) = image.size
           return np.array(image.getdata()).reshape(
               (im_height, im_width, 3)).astype(np.uint8)
```

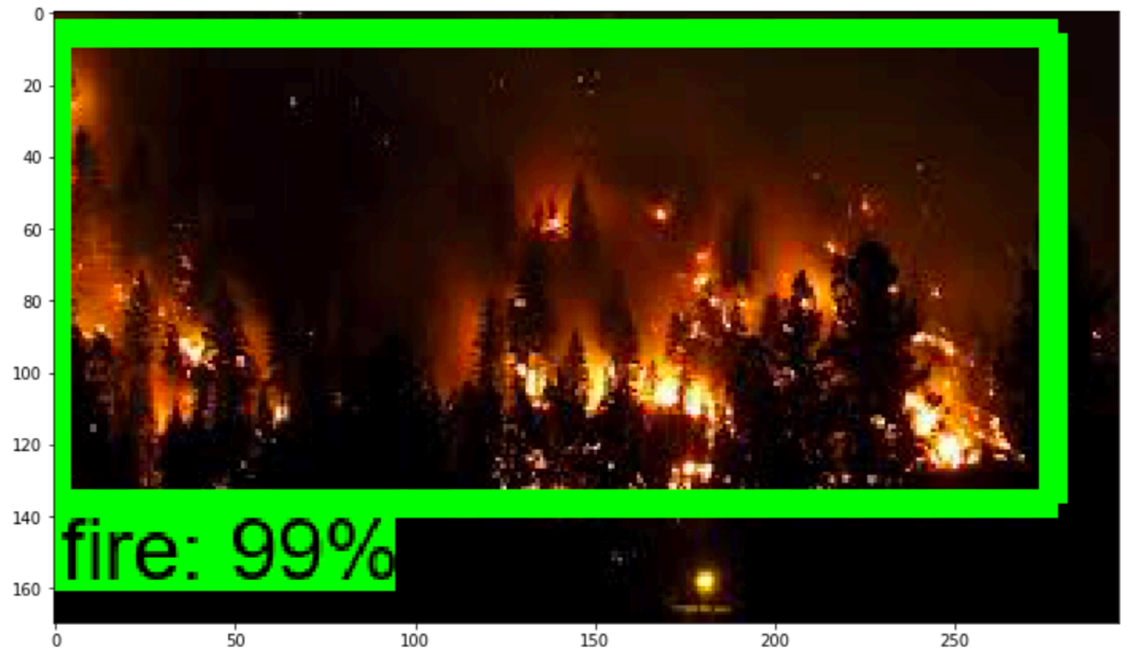
Detection

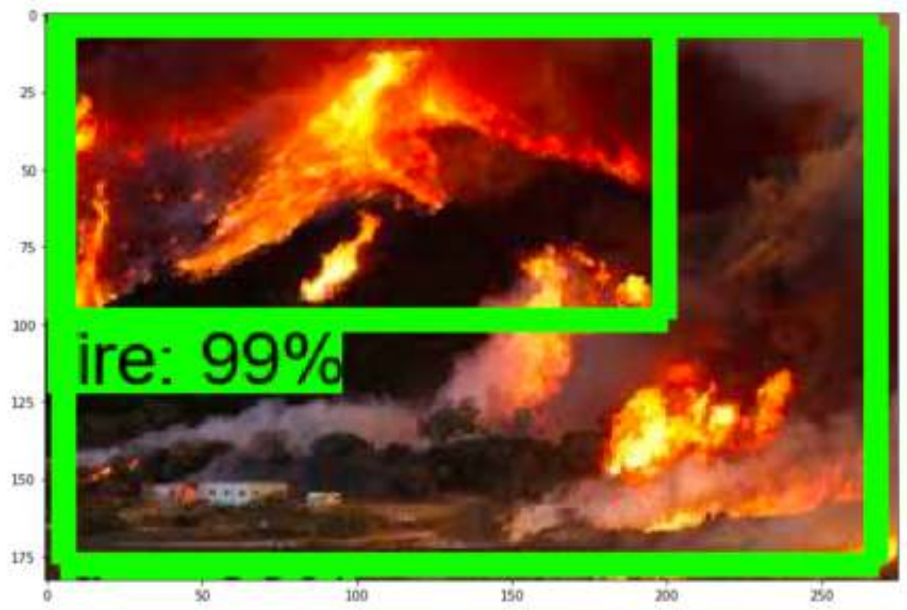
```
In [18]: PATH_TO_TEST_IMAGES_DIR = 'images\\test'
TEST_IMAGE_PATHS = [ os.path.join(PATH_TO_TEST_IMAGES_DIR, 'imgtest{}.png'.format(i)) for i in range
(1, 61) ]

# Size, in inches, of the output images.
IMAGE_SIZE = (12, 8)
```

```
In [19]: def run_inference_for_single_image(image, graph):
  with graph.as_default():
    with tf.Session() as sess:
      # Get handles to input and output tensors
      ops = tf.get_default_graph().get_operations()
      all_tensor_names = {output.name for op in ops for output in op.outputs}
      tensor_dict = {}
      for key in [
          'num_detections', 'detection_boxes', 'detection_scores',
          'detection_classes', 'detection_masks'
      ]:
        tensor_name = key + ':0'
        if tensor_name in all_tensor_names:
          tensor_dict[key] = tf.get_default_graph().get_tensor_by_name(
              tensor_name)
      if 'detection_masks' in tensor_dict:
        # The following processing is only for single image
        detection_boxes = tf.squeeze(tensor_dict['detection_boxes'], [0])
        detection_masks = tf.squeeze(tensor_dict['detection_masks'], [0])
        # Reframe is required to translate mask from box coordinates to image coordinates and fit th
e image size.
        real_num_detection = tf.cast(tensor_dict['num_detections'][0], tf.int32)
        detection_boxes = tf.slice(detection_boxes, [0, 0], [real_num_detection, -1])
        detection_masks = tf.slice(detection_masks, [0, 0, 0], [real_num_detection, -1, -1])
        detection_masks_reframed = utils_ops.reframe_box_masks_to_image_masks(
            detection_masks, detection_boxes, image.shape[1], image.shape[2])
        detection_masks_reframed = tf.cast(
            tf.greater(detection_masks_reframed, 0.5), tf.uint8)
        # Follow the convention by adding back the batch dimension
        tensor_dict['detection_masks'] = tf.expand_dims(
            detection_masks_reframed, 0)
      image_tensor = tf.get_default_graph().get_tensor_by_name('image_tensor:0')

      # Run inference
      output_dict = sess.run(tensor_dict,
                             feed_dict={image_tensor: image})
        # all outputs are float32 numpy arrays, so convert types as appropriate
        output_dict['num_detections'] = int(output_dict['num_detections'][0])
        output_dict['detection_classes'] = output_dict[
            'detection_classes'][0].astype(np.int64)
        output_dict['detection_boxes'] = output_dict['detection_boxes'][0]
        output_dict['detection_scores'] = output_dict['detection_scores'][0]
        if 'detection_masks' in output_dict:
          output_dict['detection_masks'] = output_dict['detection_masks'][0]
    return output_dict
```

Results

```
In [20]: for image_path in TEST_IMAGE_PATHS:
           image = Image.open(image_path)
           # the array based representation of the image will be used later in order to prepare the
           # result image with boxes and labels on it.
           image_np = load_image_into_numpy_array(image)
           # Expand dimensions since the model expects images to have shape: [1, None, None, 3]
           image_np_expanded = np.expand_dims(image_np, axis=0)
           # Actual detection.
           output_dict = run_inference_for_single_image(image_np_expanded, detection_graph)
           # Visualization of the results of a detection.
           vis_util.visualize_boxes_and_labels_on_image_array(
               image_np,
               output_dict['detection_boxes'],
               output_dict['detection_classes'],
               output_dict['detection_scores'],
               category_index,
               instance_masks=output_dict.get('detection_masks'),
               use_normalized_coordinates=True,
               line_thickness=8)
           plt.figure(figsize=IMAGE_SIZE)
           plt.imshow(image_np)
```

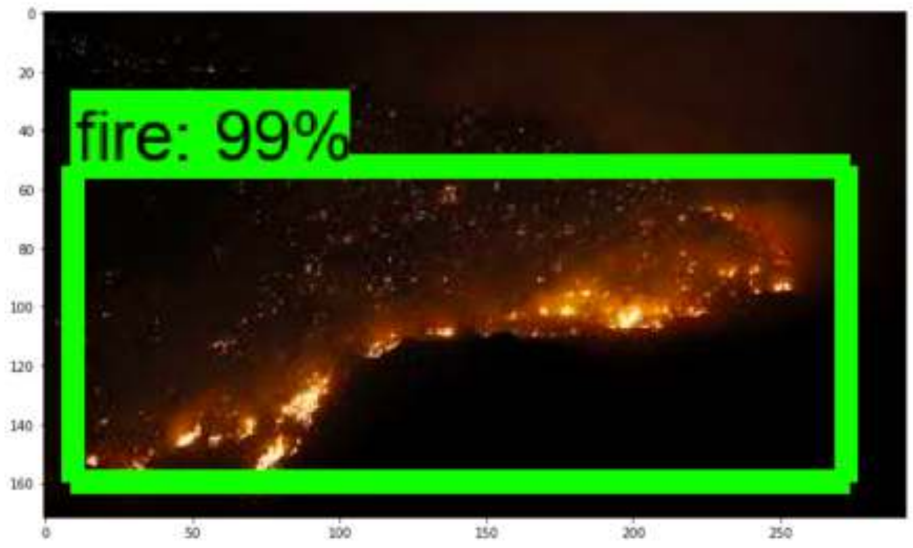



ire: 99%
0
25
50
75
100
125
150
175
0
50
100
150
200
250

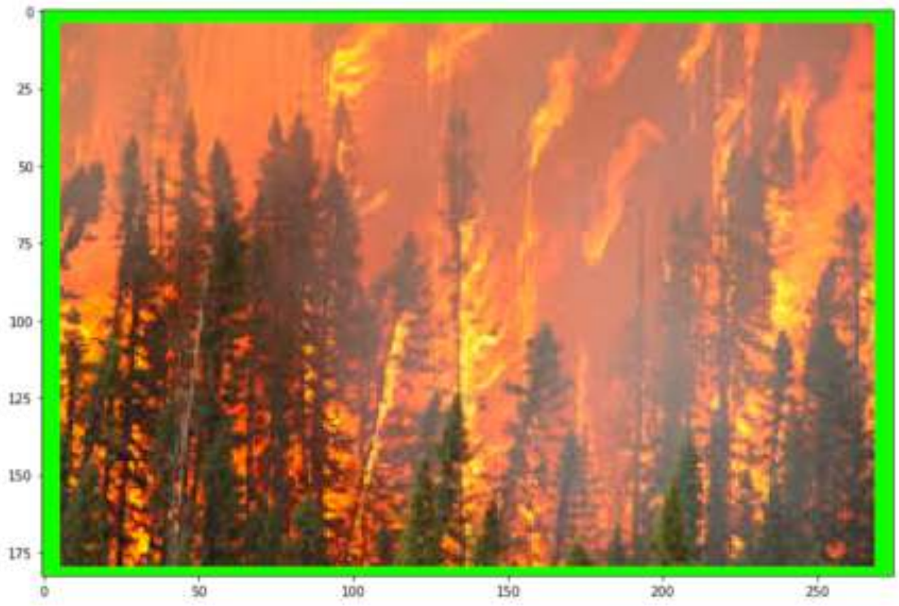
0
25
50
75
100
125
150
175
0
50
100
150
200
250

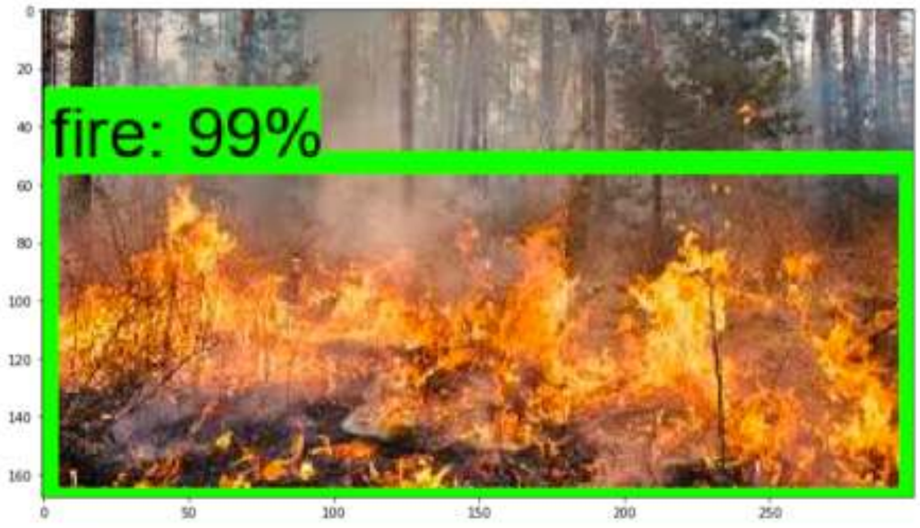
0
20
40
60
80
100
120
140
160
0
50
100
150
200
250

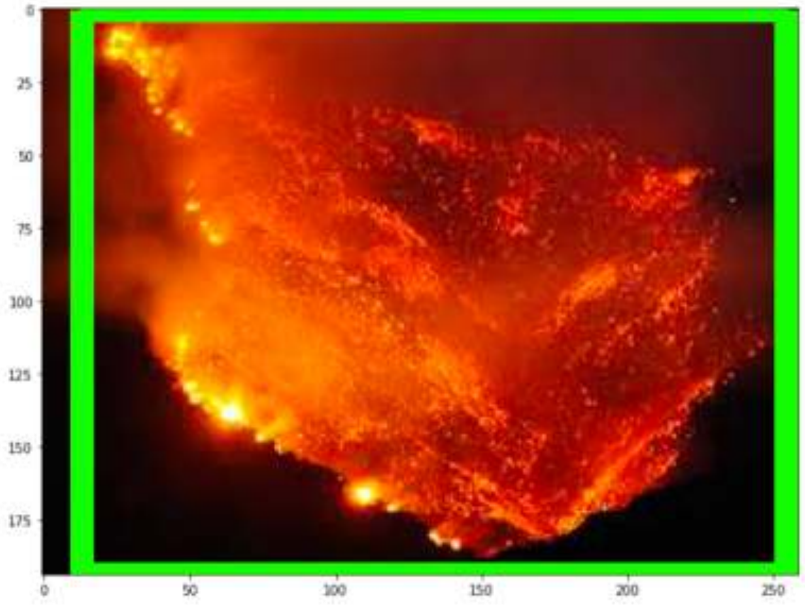
0
25
50
75
100
125
150
175
0
50
100
150
200
250


fire: 99%
0
20
40
60
80
100
120
140
160
0
50
100
150
200
250

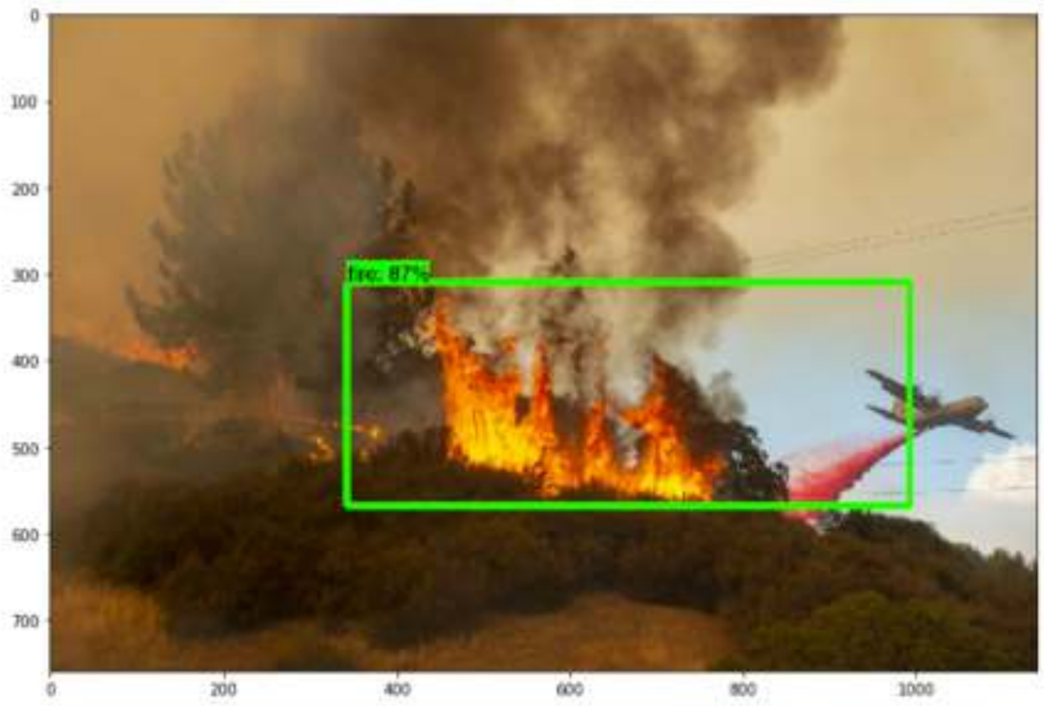
0
25
50
75
100
125
150
175
0
50
100
150
200
250

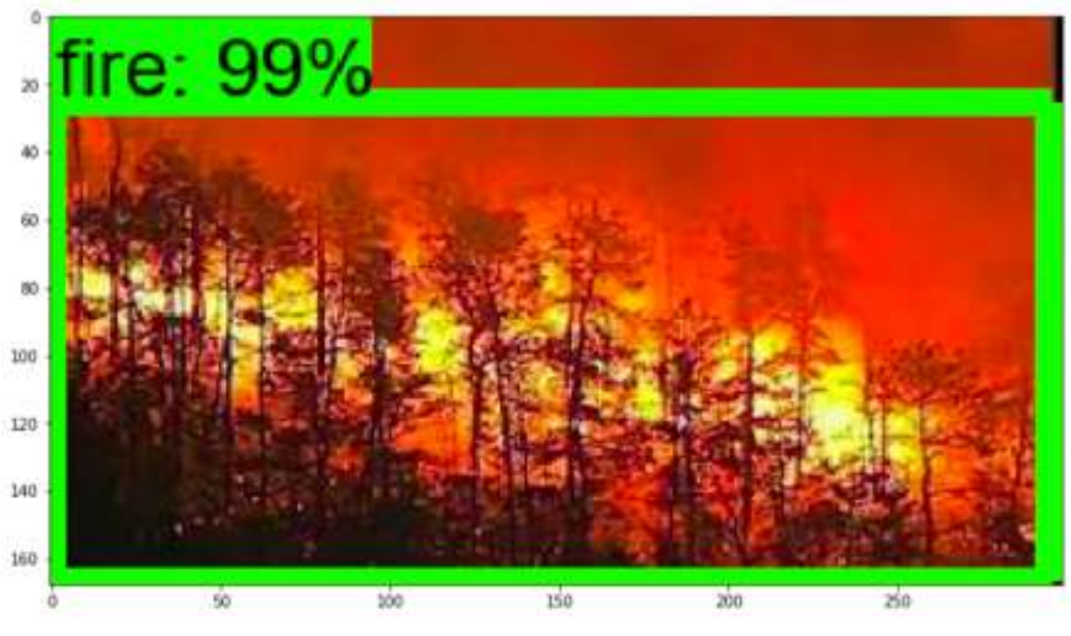


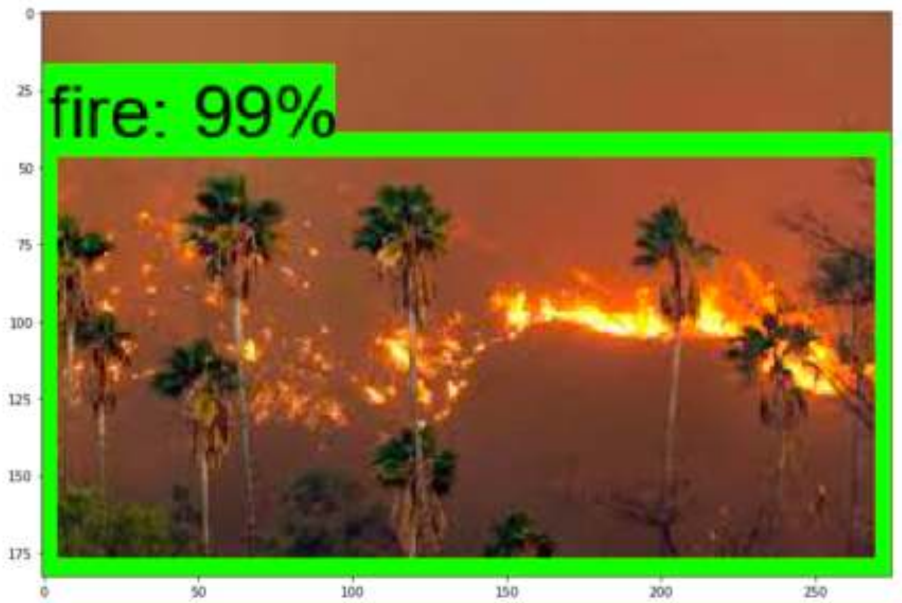

fire: 99%

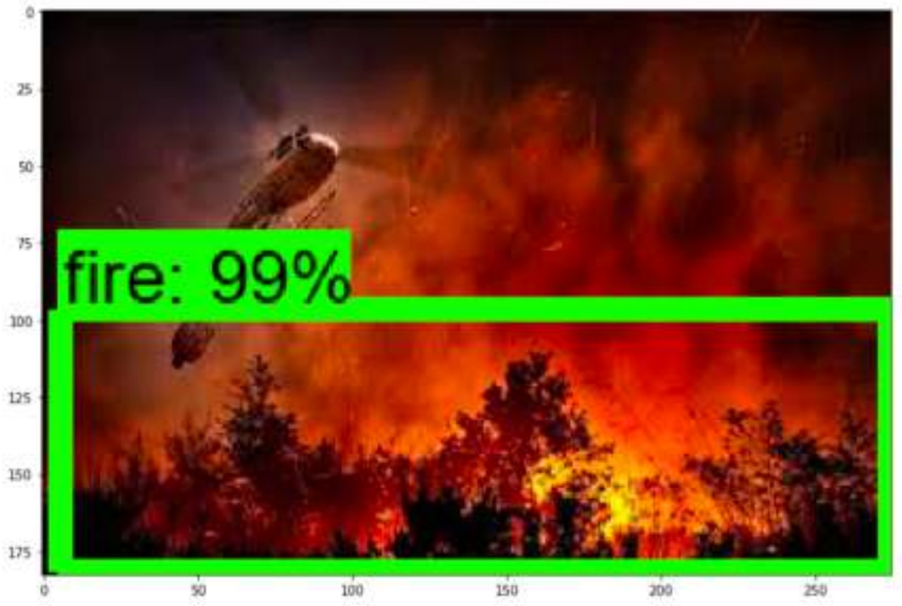


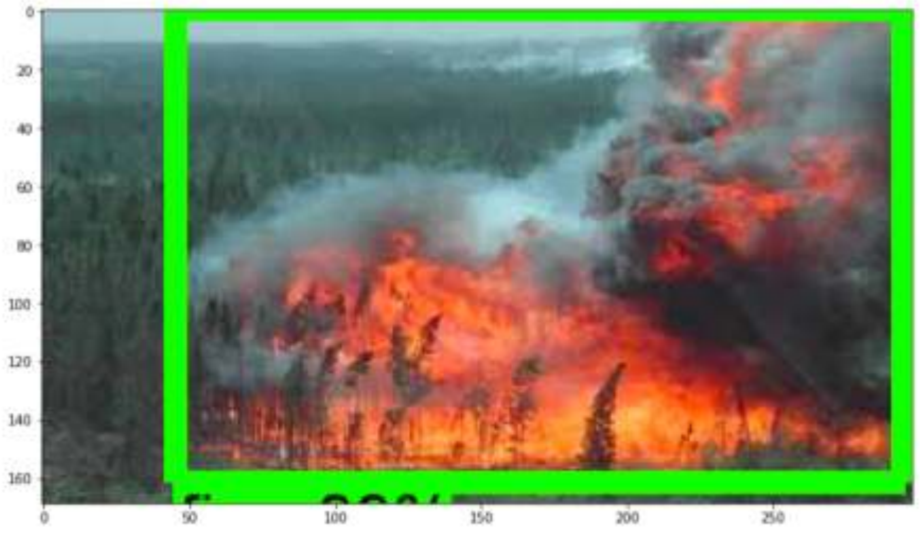
0
20
40
60
80
100
120
0
50
100
150
200
250
300
350

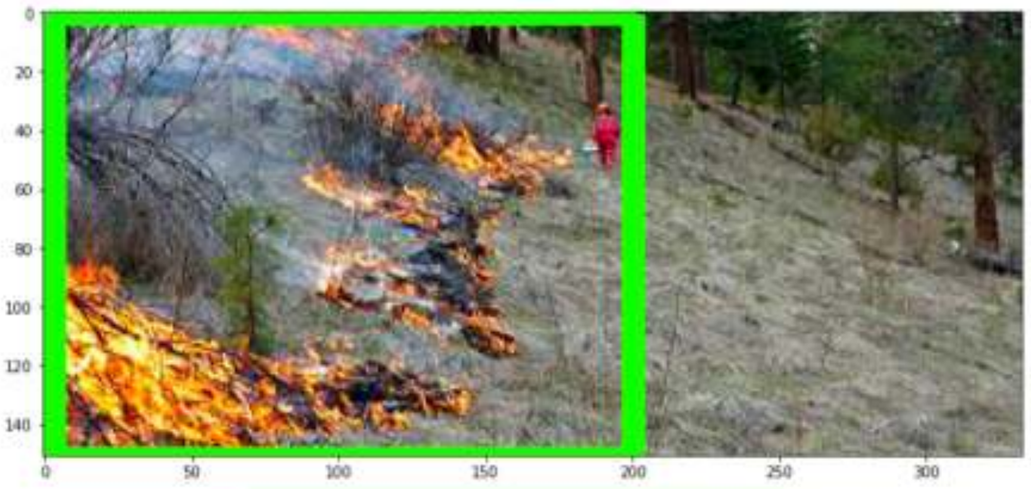

fire: 87%
0
100
200
300
400
500
600
700
0
200
400
600
800
1000


fire: 99%
0
20
40
60
80
100
120
140
160
0
50
100
150
200
250

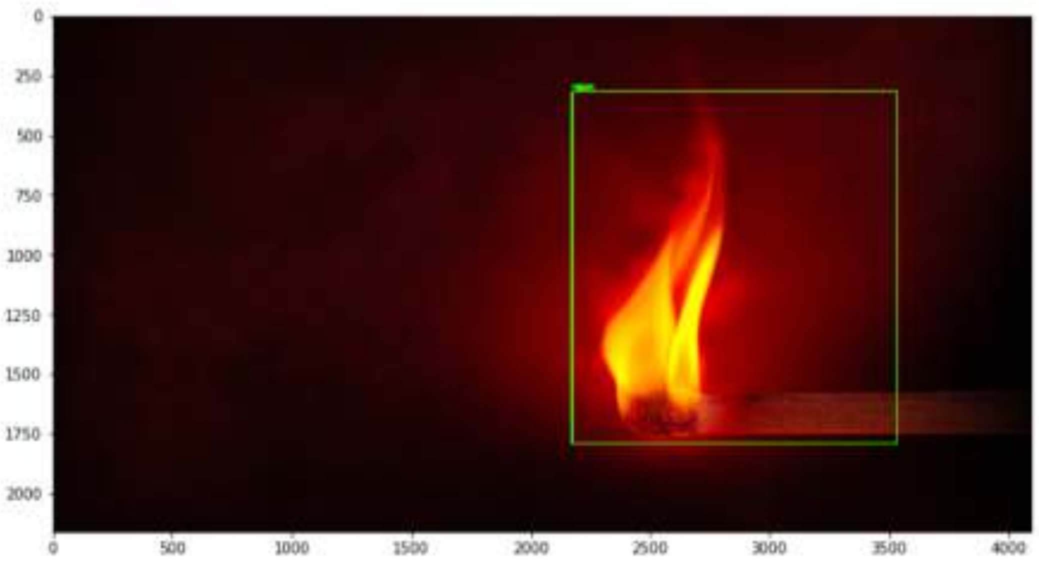

fire: 99%

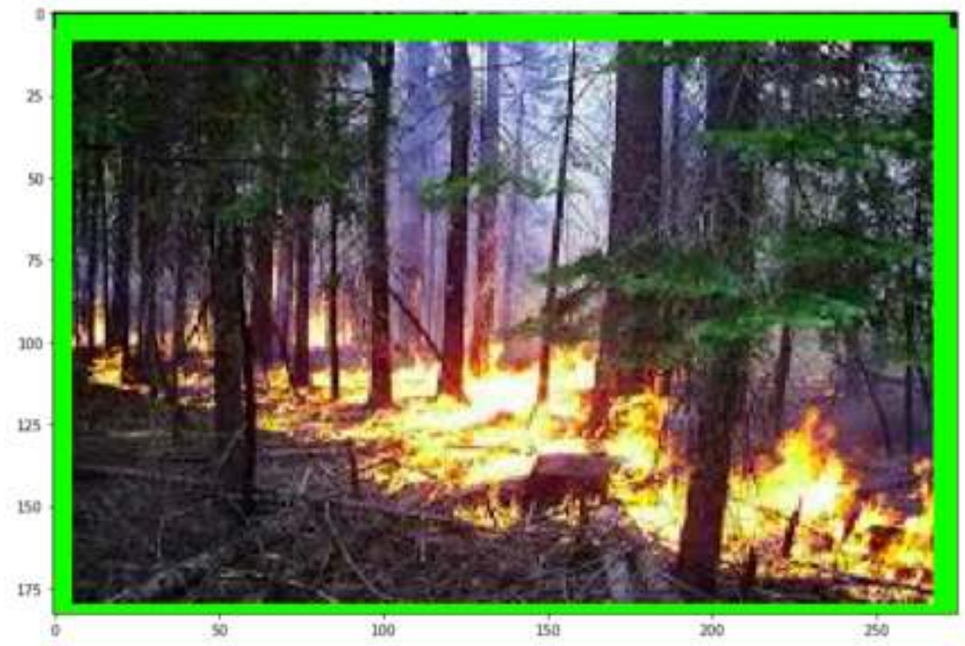

fire: 99%

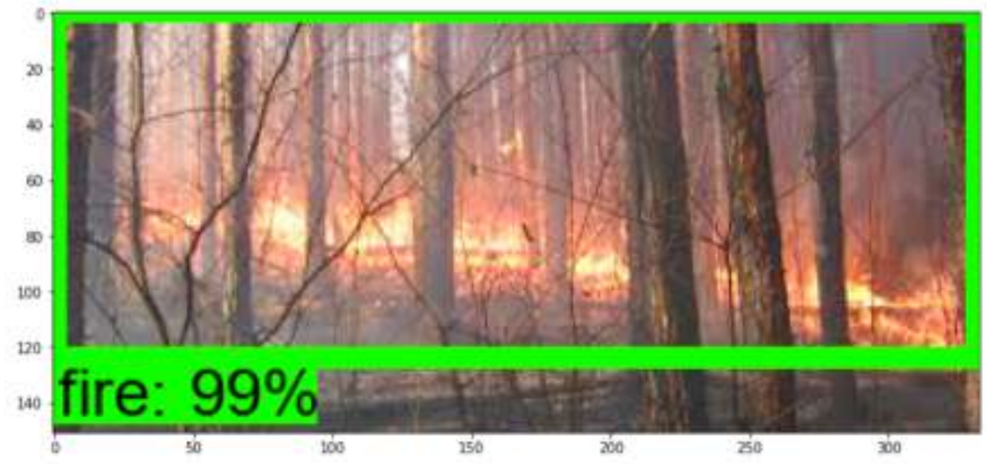


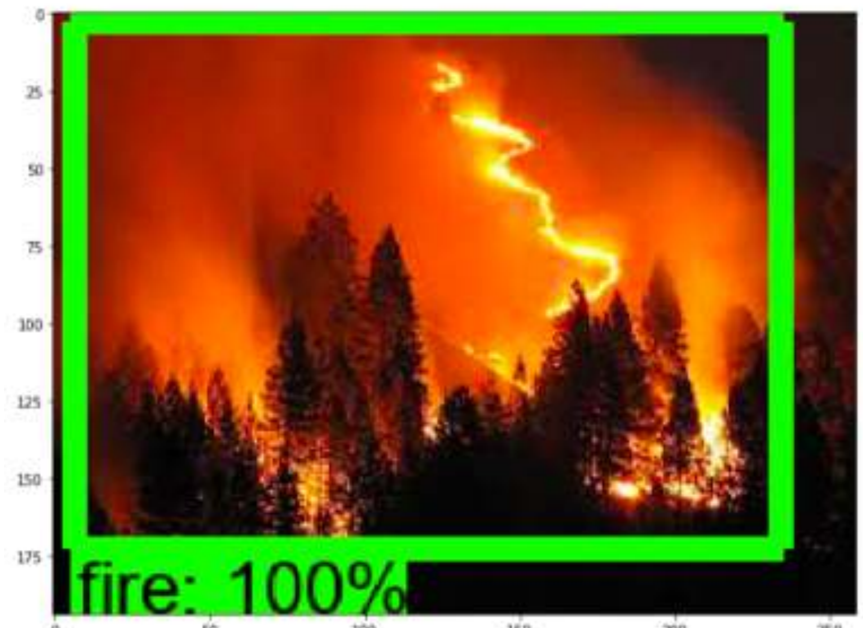

0
20
40
60
80
100
120
140
0
50
100
150
200
250
300

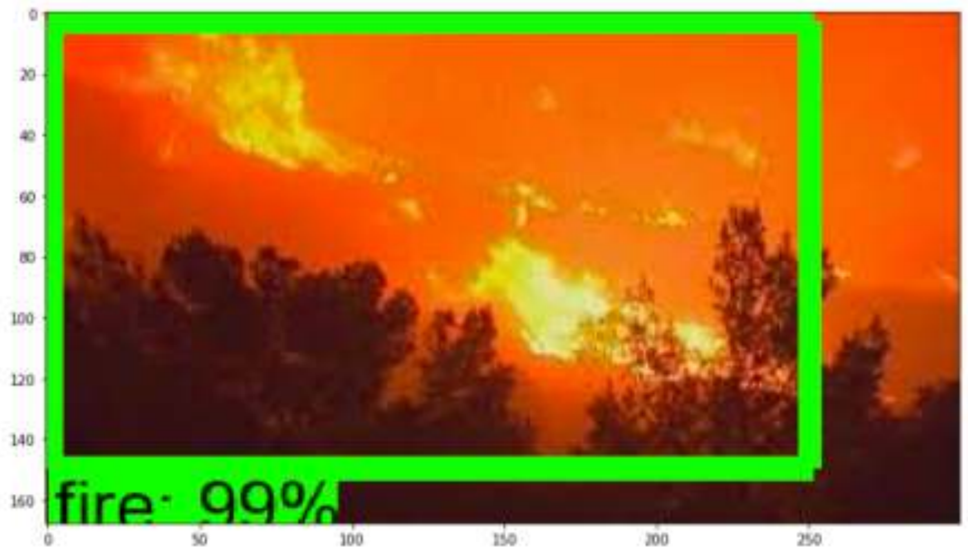
0
25
50
75
100
125
150
175
0
50
100
150
200
250

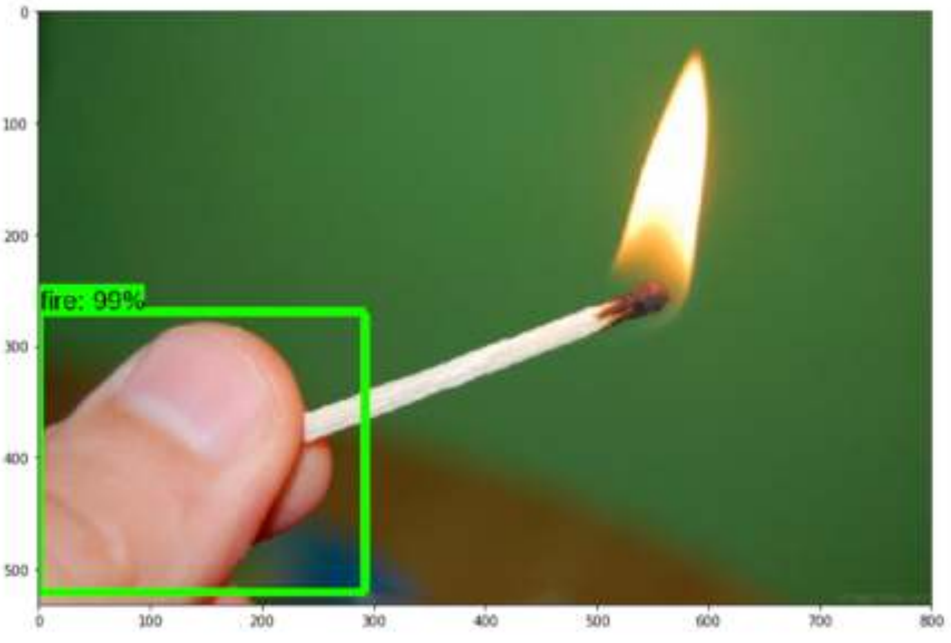

0
250
500
750
1000
1250
1500
1750
2000
0
500
1000
1500
2000
2500
3000
3500
4000

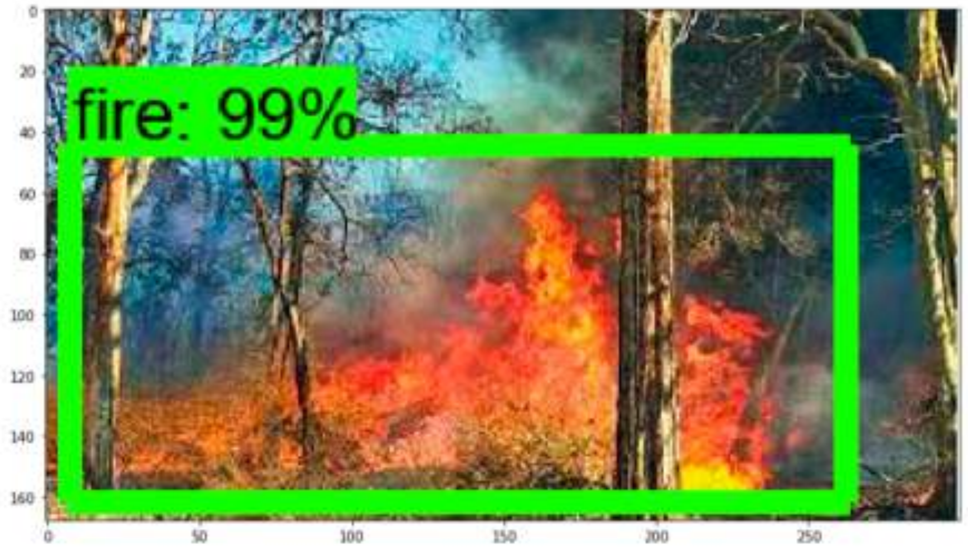

0
25
50
75
100
125
150
175
0
50
100
150
200
250

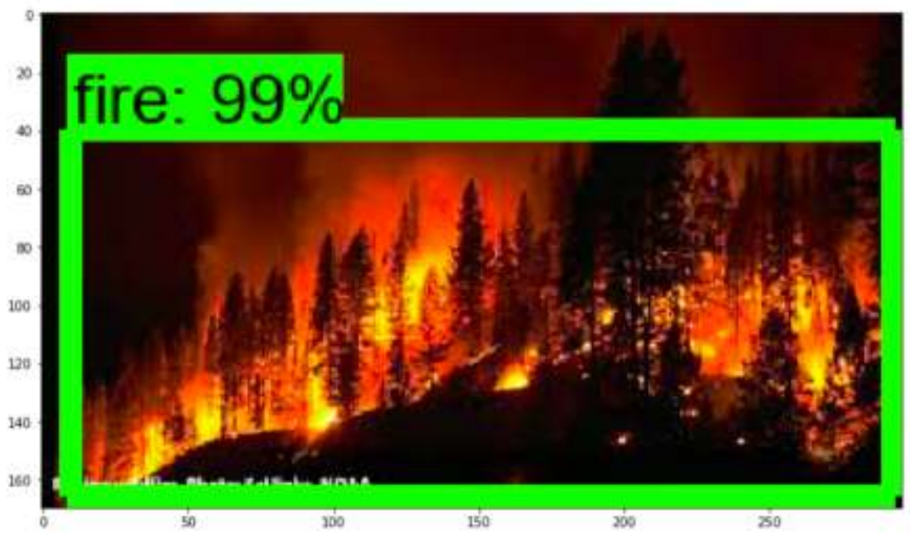

fire: 99%
0
20
40
60
80
100
120
140
0
50
100
150
200
250
300


fire: 100%
0
25
50
75
100
125
150
175
0
50
100
150
200
250

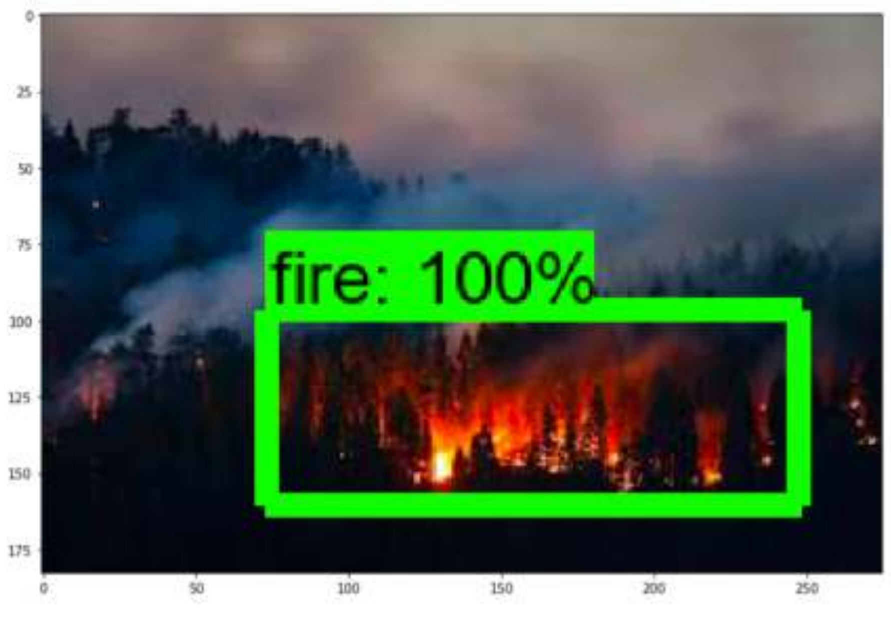

fire: 99%


fire: 99%

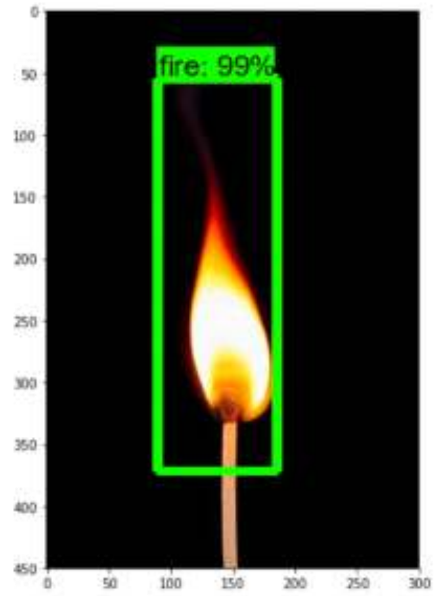

fire: 99%


fire: 99%
0
20
40
60
80
100
120
140
160
0
50
100
150
200
250

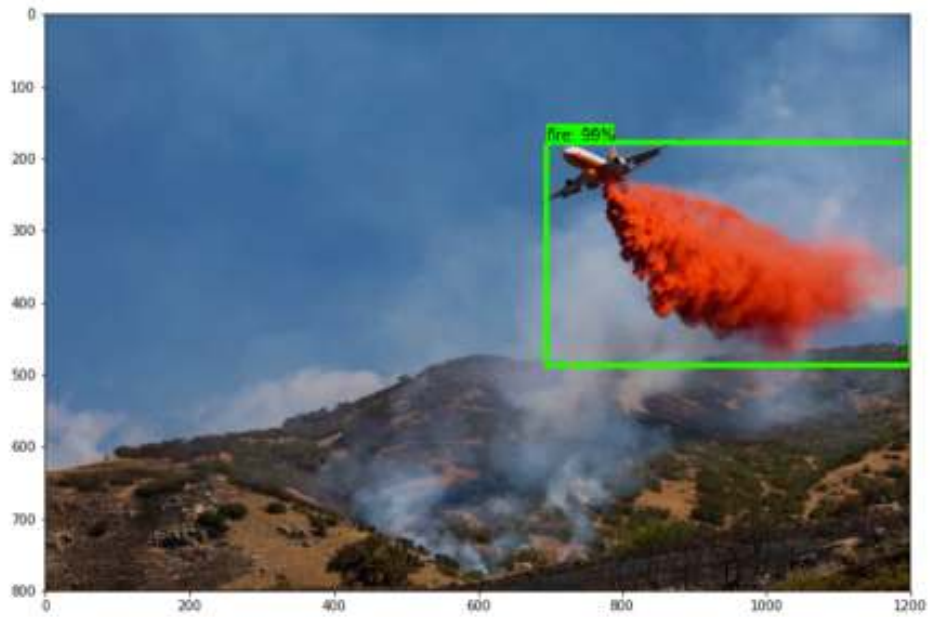
0
25
50
75
100
125
150
175
0
50
100
150
200
250


fire: 100%
0
25
50
75
100
125
150
175
0
50
100
150
200
250

fire: 99%

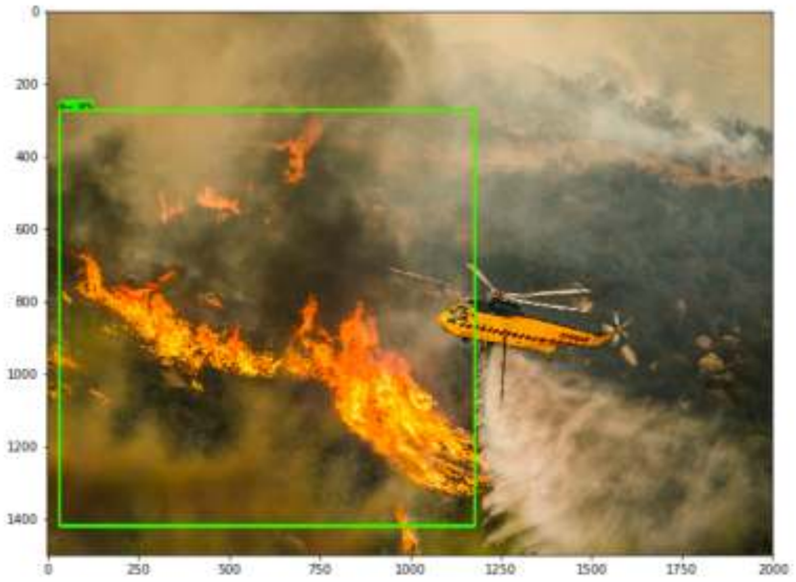

fire: 99%

fire: 60%
fire: 99%


fire: 99%

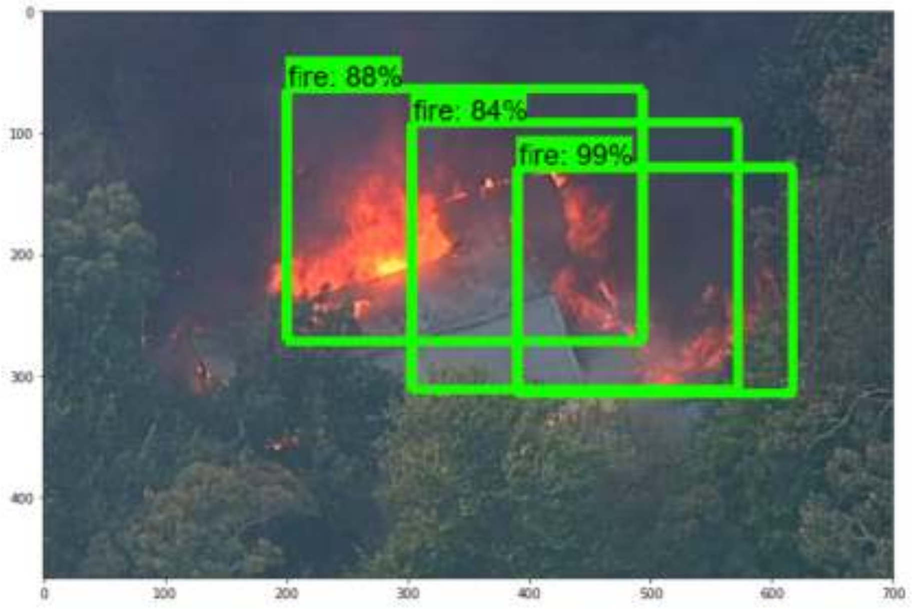
0
200
400
600
800
1000
0
200
400
600
800
1000
1200
1400

0
100
200
300
400
500
600
700
800
0
200
400
600
800
1000
1200


0
200
400
600
800
1000
1200
1400
0
250
500
750
1000
1250
1500
1750
2000

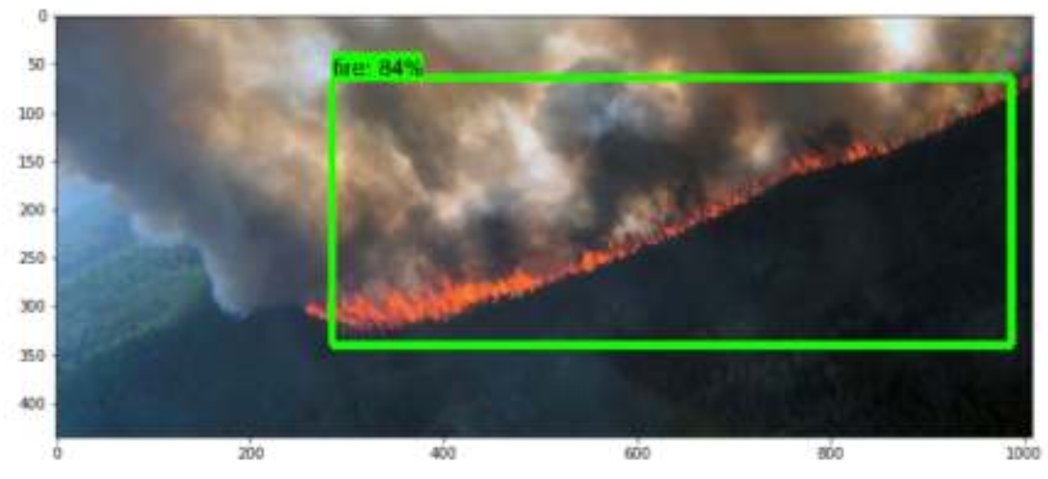
0
50
100
150
200
0
50
100
150
200
250
300
350
400

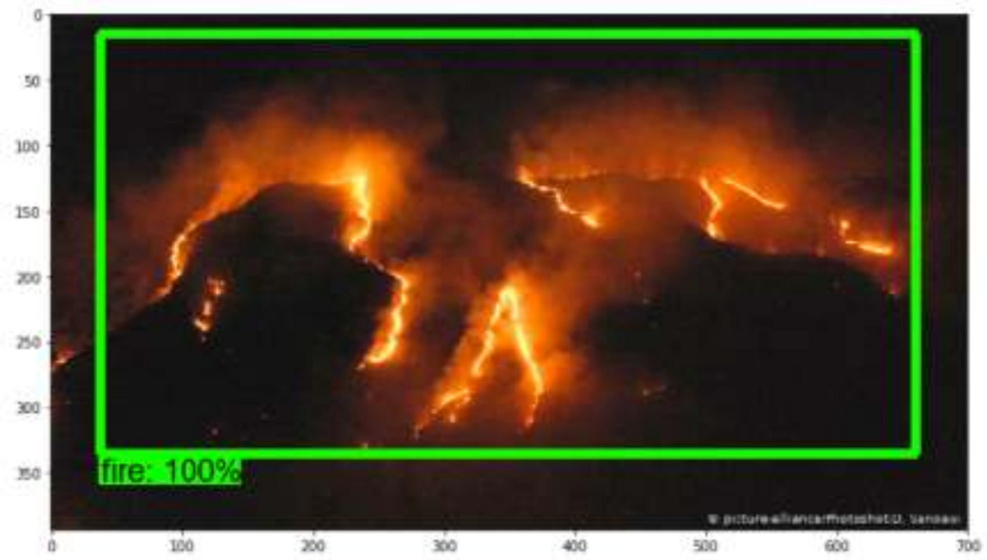
0
100
200
300
400
500
600
700
800
900
0
200
400
600
800
1000
1200


fire: 88%
fire: 84%
fire: 99%
0
100
200
300
400
0
100
200
300
400
500
600
700

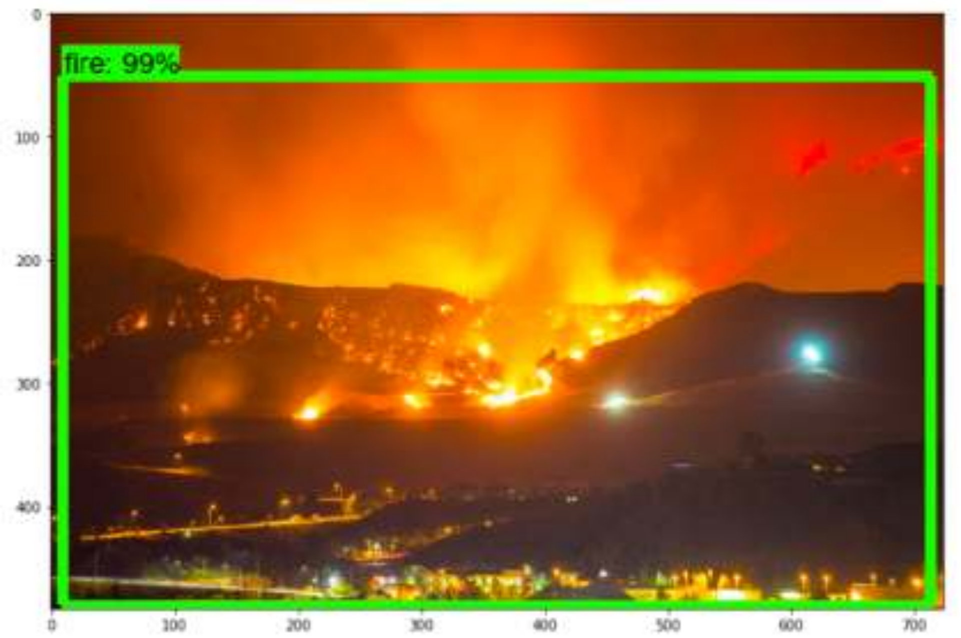
0
100
200
300
400
500
0
100
200
300
400
500
600
700

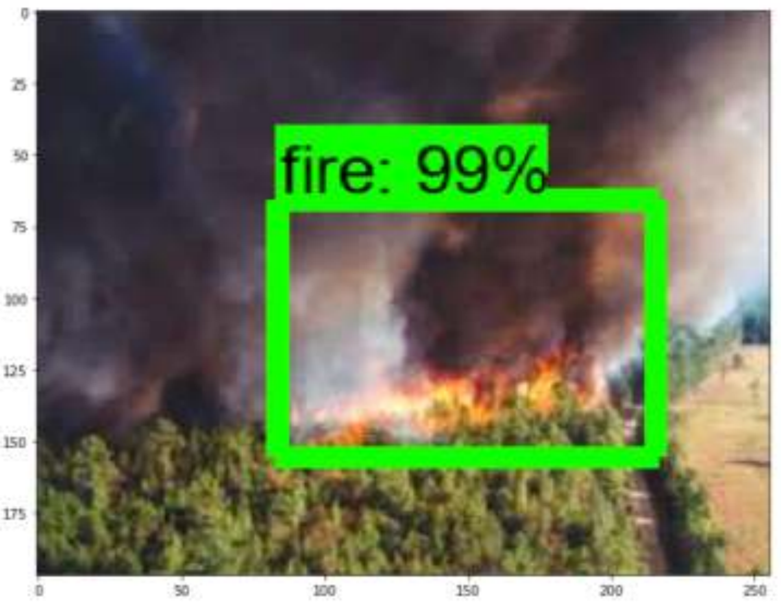
0
50
100
150
200
250
0
100
200
300
400

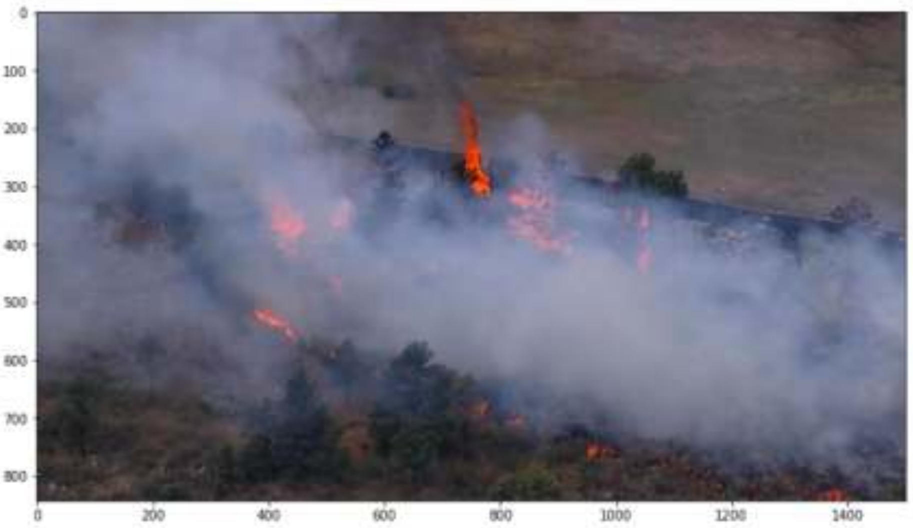

fire: 84%
0
50
100
150
200
250
300
350
400
0
200
400
600
800
1000

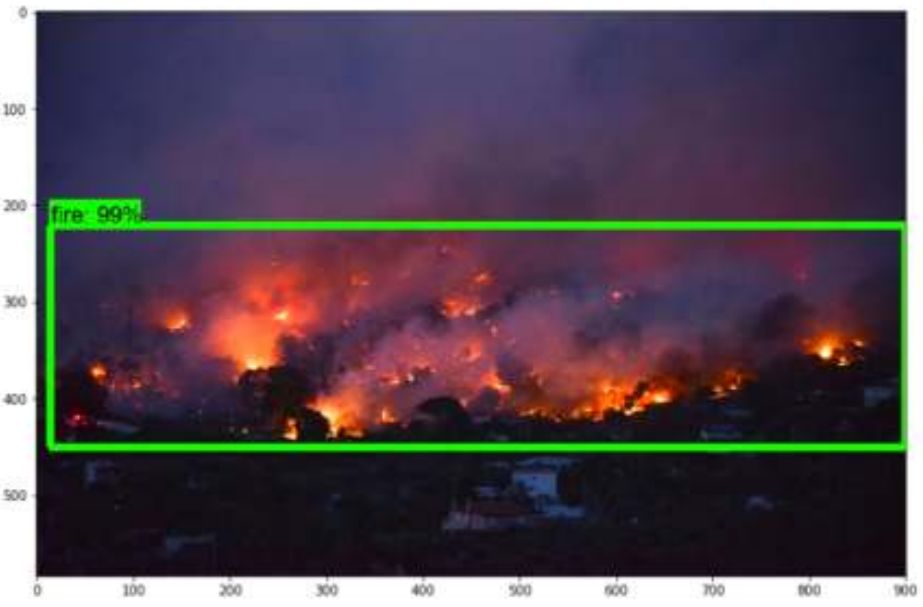

fire: 100%
© picture-alliance/Photoshot/J. Sanaeei
0
50
100
150
200
250
300
350
0
100
200
300
400
500
600
700

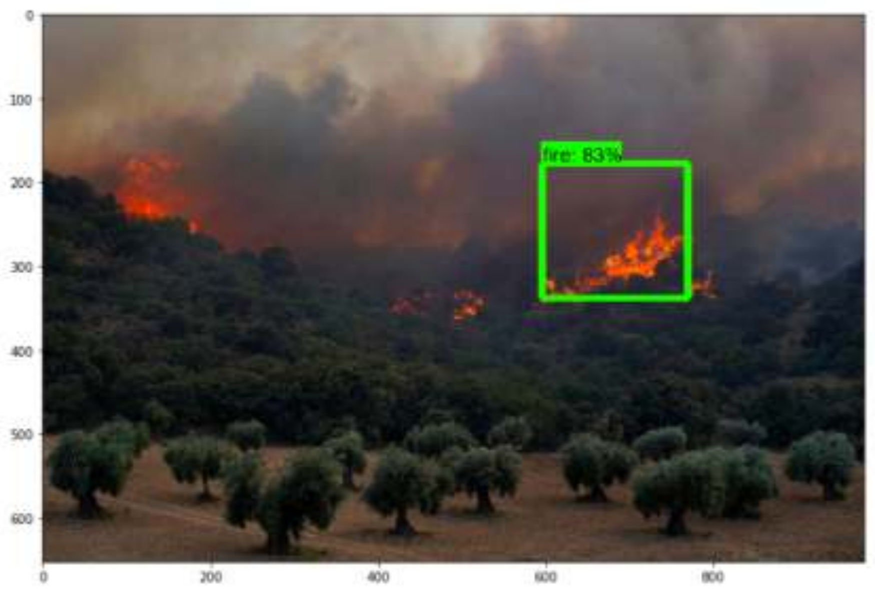
0
20
40
60
80
100
120
140
160
0
50
100
150
200
250
300

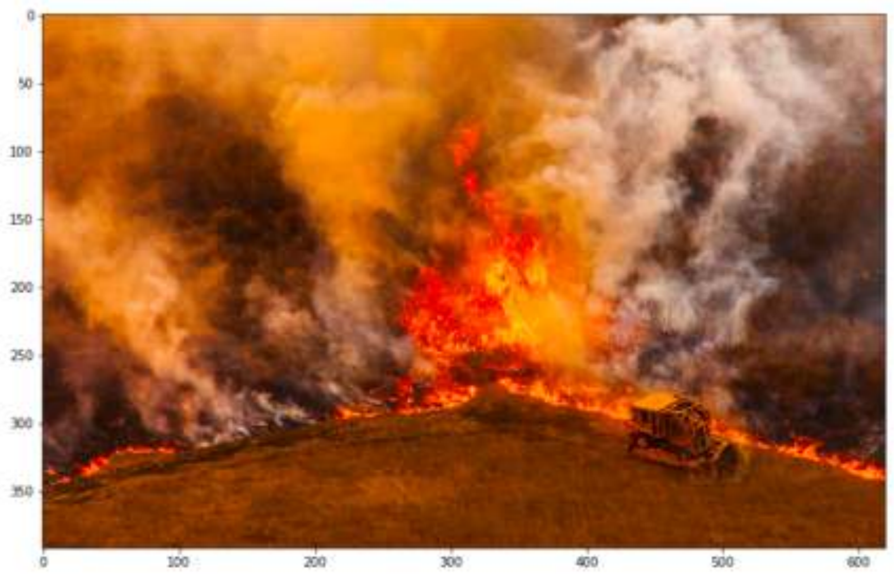

fire: 99%
0
100
200
300
400
0
100
200
300
400
500
600
700

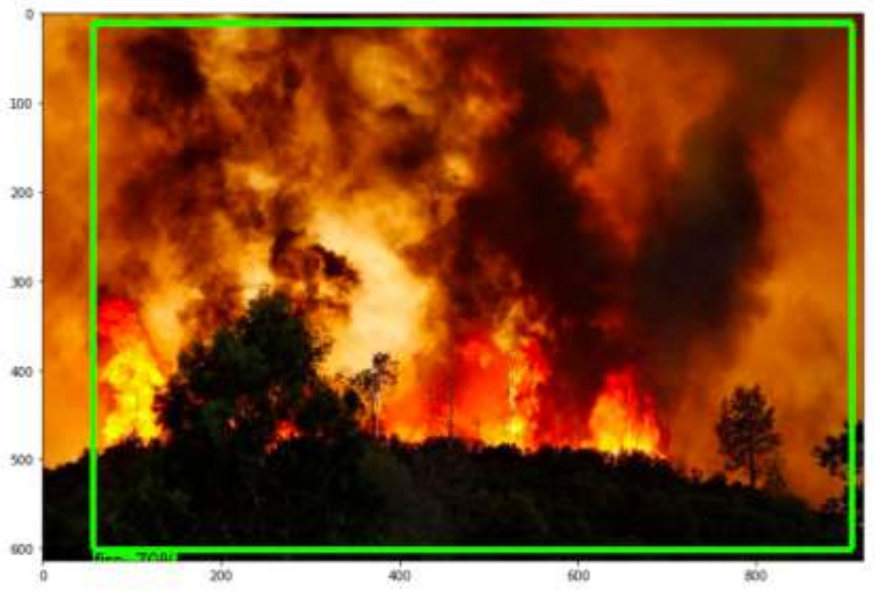

fire: 99%

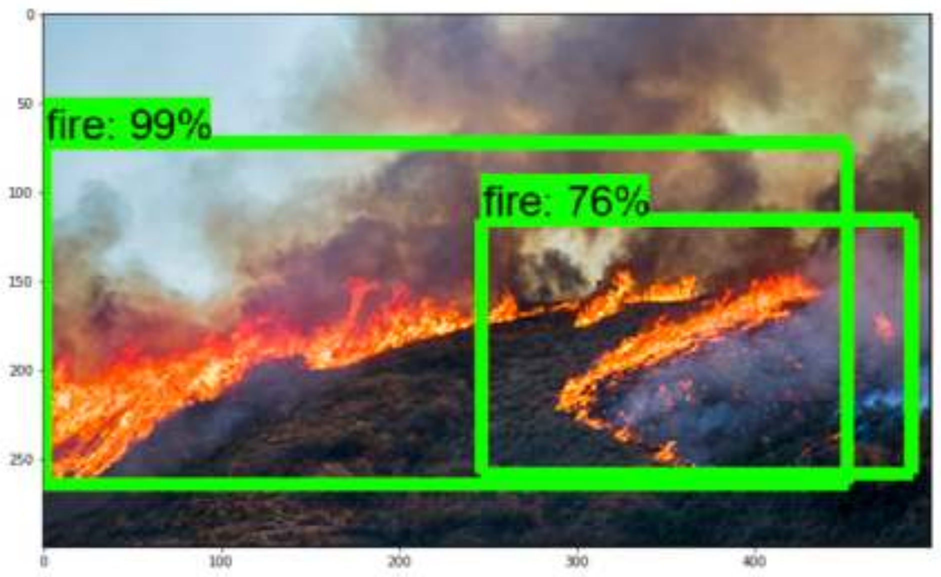


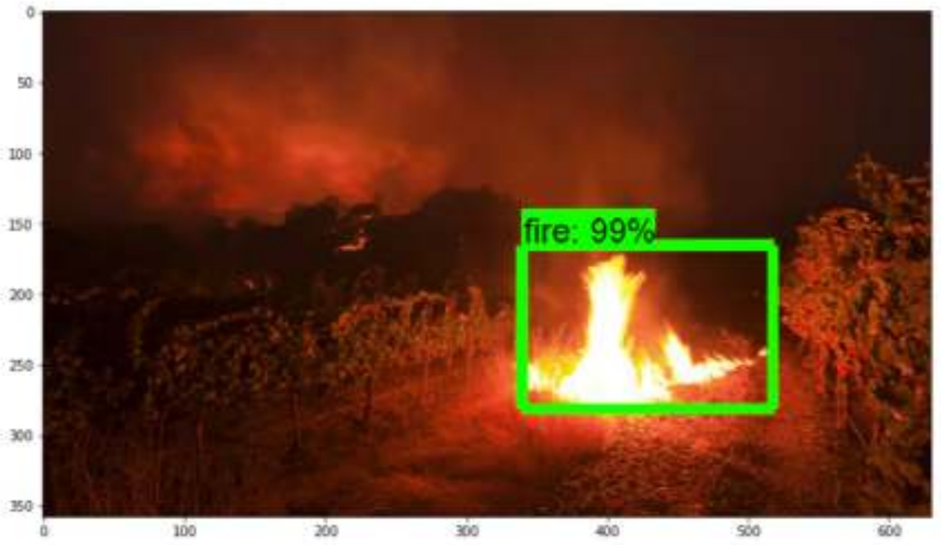

fire: 99%

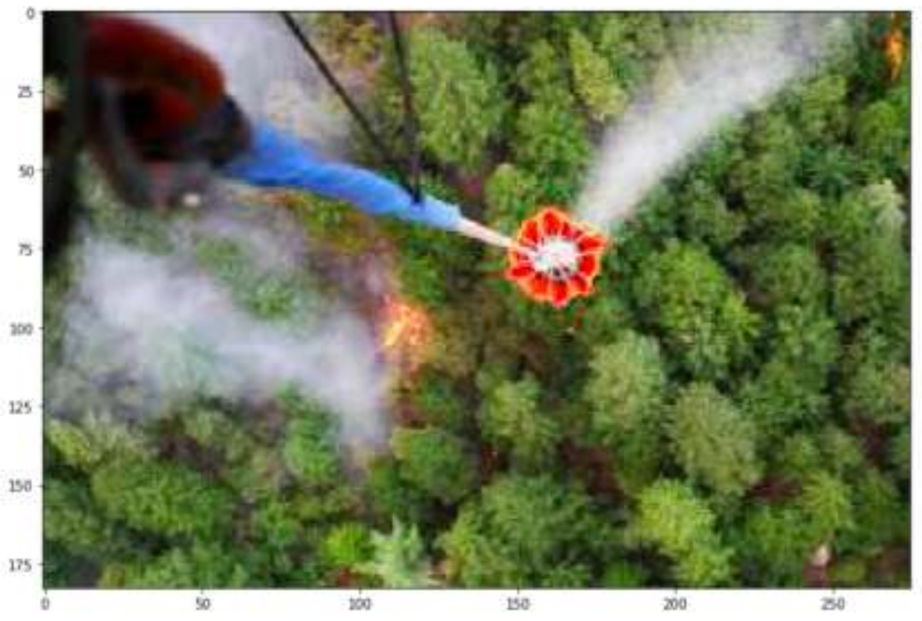

fire: 83%
0
100
200
300
400
500
600
0
200
400
600
800

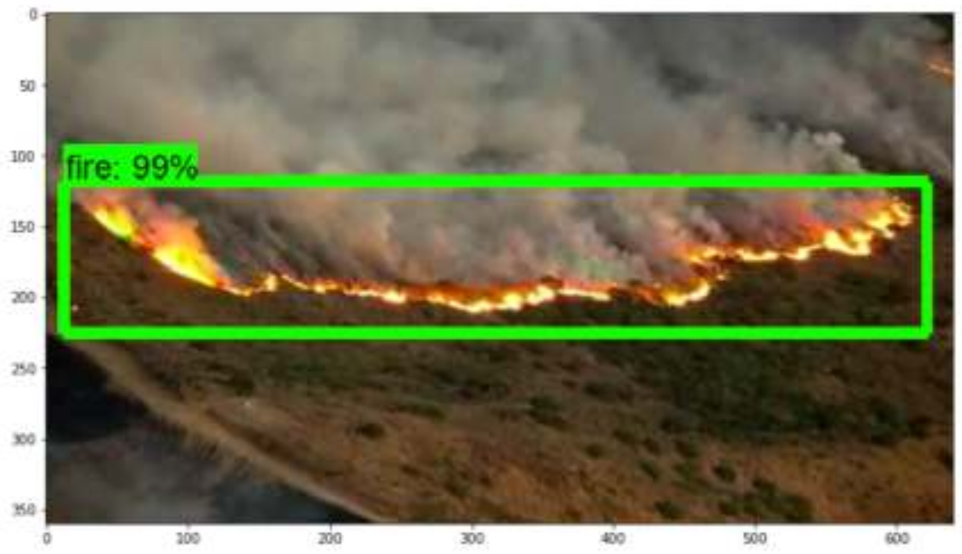

0
50
100
150
200
250
300
350
0
100
200
300
400
500
600

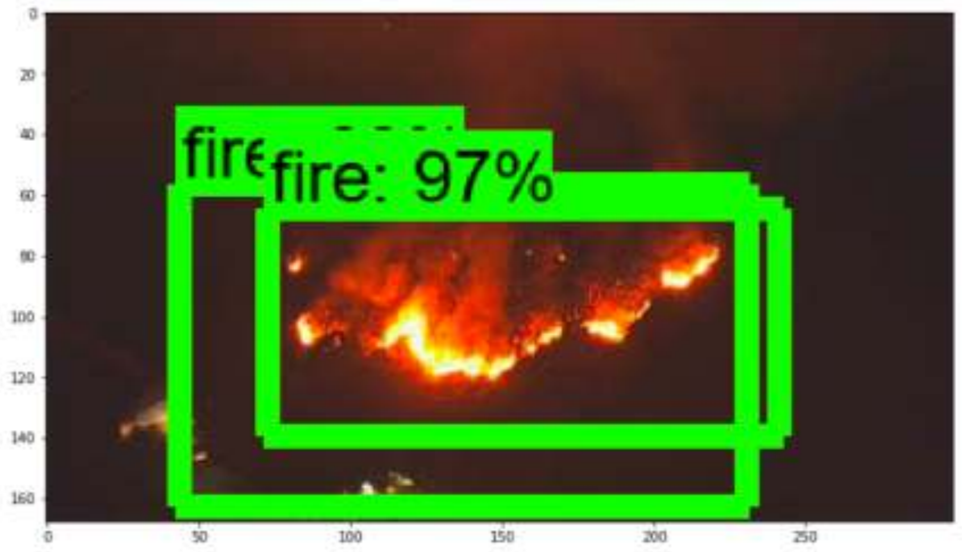

0
100
200
300
400
500
600
0
200
400
600
800

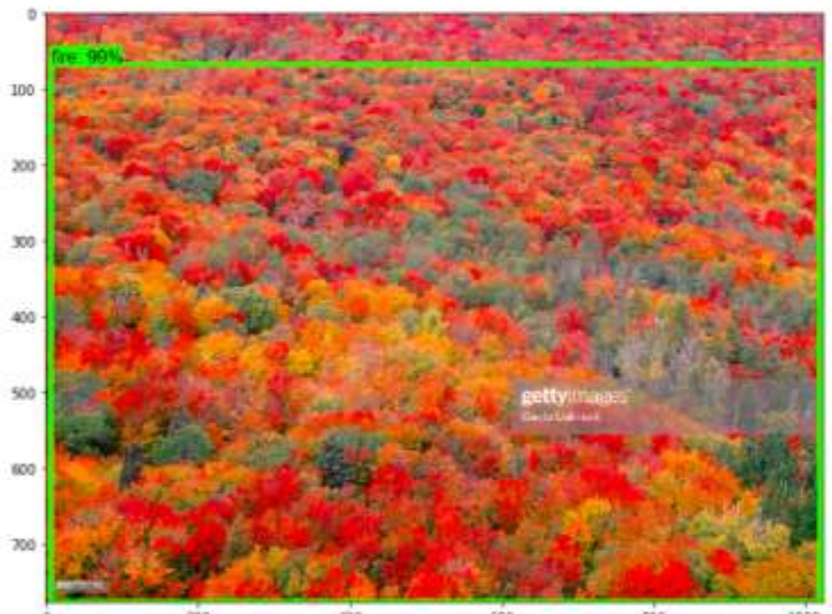

fire: 99%
fire: 76%
0
50
100
150
200
250
0
100
200
300
400


fire: 99%
0
50
100
150
200
250
300
350
0
100
200
300
400
500
600


0
25
50
75
100
125
150
175
0
50
100
150
200
250


fire: 99%


fire: 97%


fire: 99%
gettyimages

0
100
200
300
400
500
600
700
800
0
200
400
600
800
1000
1200

0
100
200
300
400
500
0
100
200
300
400
500
600
700
800
900

Appendix C

Raspberry Pi v2 Camera Module Specifications

Size	Around 25 x 24 x 9 mm
Weight	3g
Still resolution	8 Megapixels
Video modes	1080p30, 720p60 and 640 x 480p60/90
Linux integration	V4L2 driver available
C programming API	OpenMAX IL and others available
Sensor	Sony IMX219
Sensor resolution	3280 x 2464 pixels
Sensor image area	3.68 x 2.76 mm (4.6 mm diagonal)
Pixel size	1.12 µm /x 1.12 µm
Optical size	1/4"
Focal length	3.04 mm
Horizontal field of view	62.2 degrees
Vertical field of view	48.8 degrees
Focal ratio (F-Stop)	2.0

[48]

References

[1] Mohamed Hefeeda and Majid Bagheri. “Wireless Sensor Networks for Early Detection of Forest Fires”. In: *2007 IEEE Internatonal Conference on Mobile Adhoc and Sensor Systems*. 2007 IEEE Internatonal Conference on Mobile Adhoc and Sensor Systems. Pisa, Italy: IEEE, Oct. 2007, pp. 1–6. ISBN: 978-1-4244-1454-3 978-1-4244-1455-0. DOI: `10.1109/MOBHOC.2007.4428702`. URL: `http://ieeexplore.ieee.org/document/4428702/` (visited on 09/21/2019).

[2] José Martínez-de Dios et al. “Automatic Forest-Fire Measuring Using Ground Stations and Unmanned Aerial Systems”. In: *Sensors* 11.6 (June 16, 2011), pp. 6328–6353. ISSN: 1424-8220. DOI: `10.3390/s110606328`. URL: `http://www.mdpi.com/1424-8220/11/6/6328` (visited on 09/21/2019).

[3] A Laurenti and A Neri. “REMOTE SENSING, COMMUNICATIONS AND INFORMATION TECHNOLOGIES FOR VEGETATION FIRE EMERGENCIES”. In: (1996), p. 11.

[4] Andreas Kaplan and Michael Haenlein. “Siri, Siri, in my hand: Who’s the fairest in the land? On the interpretations, illustrations, and implications of artificial intelligence”. In: *Business Horizons* 62 (Nov. 1, 2018). DOI: `10.1016/j.bushor.2018.08.004`.

[5] Michael R. Genesereth and Nils J. Nilsson. *Logical Foundations of Artificial Intelligence*. Google-Books-ID: nFktBAAAQBAJ. Morgan Kaufmann, July 5, 2012. 427 pp. ISBN: 978-0-12-801554-4.

[6] John R. Koza et al. "Automated Design of Both the Topology and Sizing of Analog Electrical Circuits Using Genetic Programming". In: *Artificial Intelligence in Design '96*. Ed. by John S. Gero and Fay Sudweeks. Dordrecht: Springer Netherlands, 1996, pp. 151–170. ISBN: 978-94-009-0279-4. DOI: 10.1007/978-94-009-0279-4_9. URL: https://doi.org/10.1007/978-94-009-0279-4_9 (visited on 09/21/2019).

[7] Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Google-Books-ID: kOXDtAEACAAJ. Springer New York, Aug. 23, 2016. 738 pp. ISBN: 978-1-4939-3843-8.

[8] Jürgen Schmidhuber. "Deep learning in neural networks: An overview". In: *Neural Networks* 61 (Jan. 1, 2015), pp. 85–117. ISSN: 0893-6080. DOI: 10.1016/j.neunet.2014.09.003. URL: http://www.sciencedirect.com/science/article/pii/S0893608014002135 (visited on 09/21/2019).

[9] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. "Deep learning". In: *Nature* 521.7553 (May 2015), pp. 436–444. ISSN: 0028-0836, 1476-4687. DOI: 10.1038/nature14539. URL: http://www.nature.com/articles/nature14539 (visited on 09/21/2019).

[10] Y. Bengio, A. Courville, and P. Vincent. "Representation Learning: A Review and New Perspectives". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 35.8 (Aug. 2013), pp. 1798–1828. DOI: 10.1109/TPAMI.2013.50.

[11] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. Google-Books-ID: omivDQAAQBAJ. MIT Press, Nov. 10, 2016. 801 pp. ISBN: 978-0-262-33737-3.

[12] *A Beginner's Guide to Neural Networks and Deep Learning*. Skymind. URL: http://skymind.ai/wiki/neural-network (visited on 09/21/2019).

[13] *The building block of deep neural networks*. ResearchGate. URL: https://www.researchgate.net/figure/a-The-building-block-of-deep-neural-networks-artificial-neuron-or-node-Each-input-x_fig1_312205163 (visited on 09/21/2019).

[14] Mohamad H. Hassoun and Assistant Professor of Computer Engineering Mohamad H. Hassoun. *Fundamentals of Artificial Neural Networks*. Google-Books-ID: Otk32Y3QkxQC. MIT Press, 1995. 546 pp. ISBN: 978-0-262-08239-6.

[15] Shaoqing Ren et al. "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks". In: *Advances in Neural Information Processing Systems 28*. Ed. by C. Cortes et al. Curran Associates, Inc., 2015, pp. 91–99. URL: `http://papers.nips.cc/paper/5638-faster-r-cnn-towards-real-time-object-detection-with-region-proposal-networks.pdf` (visited on 09/21/2019).

[16] Dan C Ciresan et al. "Flexible, High Performance Convolutional Neural Networks for Image Classification". In: (), p. 6.

[17] Sumit Saha. *A Comprehensive Guide to Convolutional Neural Networks — the ELI5 way*. Medium. Dec. 17, 2018. URL: `https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53` (visited on 10/05/2019).

[18] Reg Austin. *Unmanned Aircraft Systems: UAVS Design, Development and Deployment*. John Wiley & Sons, Sept. 20, 2011. 372 pp.

[19] António Sérgio Lima Raimundo. "Autonomous Obstacle Collision Avoidance System for UAVs in rescue operations". In: (2016). URL: `https://repositorio.iscte-iul.pt/handle/10071/13045` (visited on 10/30/2019).

[20] *FAQs - Raspberry Pi Documentation*. URL: `https://www.raspberrypi.org/documentation/faqs/` (visited on 09/21/2019).

[21] Matt Richardson, Shawn Wallace, and Shawn P. Wallace. *Getting Started with Raspberry Pi*. Google-Books-ID: qKQcXBpWXPAC. "O'Reilly Media, Inc.", Dec. 24, 2012. 178 pp. ISBN: 978-1-4493-4421-4.

[22] *Jetson TX2 - eLinux.org*. URL: `https://elinux.org/Jetson_TX2` (visited on 09/22/2019).

[23] *Jetson TX2 Module*. NVIDIA Developer. May 1, 2017. URL: `https://developer.nvidia.com/embedded/jetson-tx2` (visited on 09/22/2019).

[24] *Mobile Phone Base Stations, How do mobile base stations work, Mobile Base Stations in Australia, Cell Tower, Mobile Phone Tower.* URL: `https://mobilenetworkguide.com.au/mobile_base_stations.html` (visited on 09/22/2019).

[25] Neeraj Chhabra. "Comparative Analysis of Different Wireless Technologies". In: 1.5 (2013), p. 5.

[26] Erik Dahlman, Stefan Parkvall, and Johan Skold. *4G: LTE/LTE-Advanced for Mobile Broadband.* Google-Books-ID: DLbsq9GD0zMC. Academic Press, Mar. 21, 2011. 456 pp. ISBN: 978-0-12-385490-2.

[27] Roy Consulta. "4G Mobile Networks - Technology Beyond 2.5G and 3G". In: (). URL: `https://www.academia.edu/170712/4G_Mobile_Networks_-_Technology_Beyond_2.5G_and_3G` (visited on 09/22/2019).

[28] Anis Koubaa et al. "Micro Air Vehicle Link (MAVLink) in a Nutshell: A Survey". In: *arXiv:1906.10641 [cs]* (June 22, 2019). arXiv: `1906.10641`. URL: `http://arxiv.org/abs/1906.10641` (visited on 09/22/2019).

[29] *Developer Guide - Introduction · MAVLink.* URL: `https://mavlink.io/en/` (visited on 09/22/2019).

[30] *MAVLink Basics — Dev documentation.* URL: `http://ardupilot.org/dev/docs/mavlink-basics.html` (visited on 09/22/2019).

[31] *Serialization · MAVLink Developer Guide.* URL: `https://mavlink.io/en/guide/serialization.html` (visited on 09/22/2019).

[32] *Community: — ArduPilot documentation.* URL: `http://ardupilot.org/ardupilot/` (visited on 09/22/2019).

[33] *ArduPilot/ardupilot.* original-date: 2013-01-09T00:58:52Z. Sept. 22, 2019. URL: `https://github.com/ArduPilot/ardupilot` (visited on 09/22/2019).

[34] *L3GD20H.* STMicroelectronics. URL: `https://www.st.com/en/mems-and-sensors/l3gd20h.html` (visited on 09/22/2019).

[35] *Gyroscope | Definition of Gyroscope by Lexico.* Lexico Dictionaries | English. URL: https://www.lexico.com/en/definition/gyroscope (visited on 09/22/2019).

[36] *Wolfram Demonstrations Project brings ideas to life with over 11k interactive #WolframNotebooks for education, research, recreation & more. #WolframDemo #WolfLang.* URL: /Gyroscope/ (visited on 09/22/2019).

[37] *What is a Magnetometer? - Definition from Techopedia.* Techopedia.com. URL: https://www.techopedia.com/definition/30792/magnetometer (visited on 09/22/2019).

[38] "Ultra-compact high-performance eCompass module: 3D accelerometer and 3D magnetometer". In: (), p. 52.

[39] Fintan Corrigan. *Drone Gyro Stabilization, IMU And Flight Controllers Explained.* DroneZon. July 2, 2019. URL: https://www.dronezon.com/learn-about-drones-quadcopters/three-and-six-axis-gyro-stabilized-drones/ (visited on 09/22/2019).

[40] *3-Axis Magnetometer.* URL: https://aerospace.honeywell.com/en/products/navigation-and-sensors/3-axis-magnetometer (visited on 09/22/2019).

[41] Ryan Goodrich May 31 and 2018 Tech. *Accelerometer vs. Gyroscope: What's the Difference?* livescience.com. URL: https://www.livescience.com/40103-accelerometer-vs-gyroscope.html (visited on 09/22/2019).

[42] *MPU-9250 | TDK.* URL: https://www.invensense.com/products/motion-tracking/9-axis/mpu-9250/ (visited on 09/22/2019).

[43] *Altimeter Pressure Sensor | Digital Output / Stainless Steel Cap | MS5611.* TE Connectivity. URL: https://www.te.com/usa-en/product-CAT-BLPS0036.html (visited on 09/22/2019).

[44] *High Precision Micro Barometer Module MS5611 | AMSYS.* URL: https://www.amsys.info/products/ms5611.htm (visited on 09/22/2019).

[45] Nathan Hagen and Michael Kudenov. "Review of snapshot spectral imaging technologies". In: *Optical Engineering* 52 (Sept. 1, 2013), p. 090901. DOI: 10.1117/1.OE.52.9.090901.

[46] *Remote Sensing - 3rd Edition.* URL: https://www.elsevier.com/books/remote-sensing/schowengerdt/978-0-12-369407-2 (visited on 09/22/2019).

[47] Fintan Corrigan. *10 Thermal Vision Cameras For Drones And How Thermal Imaging Works.* DroneZon. Sept. 3, 2019. URL: https://www.dronezon.com/learn-about-drones-quadcopters/9-heat-vision-cameras-for-drones-and-how-thermal-imaging-works/ (visited on 09/22/2019).

[48] *Camera Module - Raspberry Pi Documentation.* URL: https://www.raspberrypi.org/documentation/hardware/camera/ (visited on 10/25/2019).

[49] Leo Eldredge. "WAAS Performance Standard". In: (2008), p. 60.

[50] *Here2 GNSS GPS Module.* English. URL: https://www.getfpv.com/here2-gnss-gps-module.html (visited on 09/22/2019).

[51] *Hex Cube (Pixhawk 2) · PX4 v1.9.0 User Guide.* URL: https://docs.px4.io/v1.9.0/en/flight_controller/pixhawk-2.html (visited on 09/22/2019).

[52] Ioannis Pitas. *Digital Image Processing Algorithms and Applications.* Google-Books-ID: VQs_Ly4DYDMC. John Wiley & Sons, Feb. 22, 2000. 436 pp. ISBN: 978-0-471-37739-9.

[53] "Autonomous forest fire detection". In: (2003), p. 11.

[54] J A J Berni et al. "REMOTE SENSING OF VEGETATION FROM UAV PLATFORMS USING LIGHTWEIGHT MULTISPECTRAL AND THERMAL IMAGING SENSORS". In: (), p. 6.

[55] David W. Casbeer et al. "Cooperative forest fire surveillance using a team of small unmanned air vehicles". In: *International Journal of Systems Science* 37.6 (May 15, 2006), pp. 351–360. ISSN: 0020-7721. DOI: 10.1080/00207720500438480. URL: https://doi.org/10.1080/00207720500438480 (visited on 09/22/2019).

[56] Vincent G. Ambrosia and Thomas Zajkowski. "Selection of Appropriate Class UAS/Sensors to Support Fire Monitoring: Experiences in the United States". In: *Handbook of Unmanned Aerial Vehicles*. Ed. by Kimon P. Valavanis and George J. Vachtsevanos. Dordrecht: Springer Netherlands, 2015, pp. 2723–2754. ISBN: 978-90-481-9707-1. DOI: 10.1007/978-90-481-9707-1_73. URL: https://doi.org/10.1007/978-90-481-9707-1_73 (visited on 09/22/2019).

[57] Mengxin Li et al. "Review of fire detection technologies based on video image". In: *Journal of theoretical and applied information technology* 49.2 (2013), pp. 700–707. ISSN: 1992-8645. URL: https://www.safetylit.org/citations/index.php?fuseaction=citations.viewdetails&citationIds[]=citjournalarticle_490522_38 (visited on 09/22/2019).

[58] 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*. 2007 IEEE International Conference on Acoustics, Speech and Signal Processing - ICASSP '07. Vol. 1. Apr. 2007, pp. I–1205–I–1208. DOI: 10.1109/ICASSP.2007.366130.

[59] Elham Mahdipour and Chitra Dadkhah. "Automatic fire detection based on soft computing techniques: review from 2000 to 2010". In: *Artificial Intelligence Review* 42.4 (Dec. 1, 2014), pp. 895–934. ISSN: 1573-7462. DOI: 10.1007/s10462-012-9345-z. URL: https://doi.org/10.1007/s10462-012-9345-z (visited on 09/22/2019).

[60] Steve Rudz et al. "On the Evaluation of Segmentation Methods for Wildland Fire". In: *Advanced Concepts for Intelligent Vision Systems*. Ed. by Jacques Blanc-Talon et al. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2009, pp. 12–23. ISBN: 978-3-642-04697-1.

[61] Thou-Ho Chen, Ping-Hsueh Wu, and Yung-Chuen Chiou. "An early fire-detection method based on image processing". In: *2004 International Conference on Image Processing, 2004. ICIP '04*. 2004 International Conference

on Image Processing, 2004. ICIP '04. Vol. 3. Oct. 2004, 1707–1710 Vol. 3. DOI: 10.1109/ICIP.2004.1421401.

[62] B. U. Toreyin, Y. Dedeoglu, and A. E. Cetin. "Flame detection in video using hidden Markov models". In: *IEEE International Conference on Image Processing 2005*. IEEE International Conference on Image Processing 2005. Vol. 2. Sept. 2005, pp. II–1230. DOI: 10.1109/ICIP.2005.1530284.

[63] B. Uğur Töreyin et al. "Computer vision based method for real-time fire and flame detection". In: *Pattern Recognition Letters* 27.1 (Jan. 1, 2006), pp. 49–58. ISSN: 0167-8655. DOI: 10.1016/j.patrec.2005.06.015. URL: http://www.sciencedirect.com/science/article/pii/S0167865505001819 (visited on 09/22/2019).

[64] Turgay Çelik and Hasan Demirel. "Fire detection in video sequences using a generic color model". In: *Fire Safety Journal* 44.2 (Feb. 1, 2009), pp. 147–158. ISSN: 0379-7112. DOI: 10.1016/j.firesaf.2008.05.005. URL: http://www.sciencedirect.com/science/article/pii/S0379711208000568 (visited on 09/22/2019).

[65] Dengyi Zhang et al. "Forest fire and smoke detection based on video image segmentation". In: *MIPPR 2007: Pattern Recognition and Computer Vision*. MIPPR 2007: Pattern Recognition and Computer Vision. Vol. 6788. International Society for Optics and Photonics, Nov. 15, 2007, 67882H. DOI: 10.1117/12.751611. URL: https://www.spiedigitallibrary.org/conference-proceedings-of-spie/6788/67882H/Forest-fire-and-smoke-detection-based-on-video-image-segmentation/10.1117/12.751611.short (visited on 09/22/2019).

[66] C. Yuan, Z. Liu, and Y. Zhang. "UAV-based forest fire detection and tracking using image processing techniques". In: *2015 International Conference on Unmanned Aircraft Systems (ICUAS)*. 2015 International Conference on Unmanned Aircraft Systems (ICUAS). June 2015, pp. 639–643. DOI: 10.1109/ICUAS.2015.7152345.

[67] Daniel R. Dale. “Automated ground maintenance and health management for autonomous unmanned aerial vehicles”. Thesis. Massachusetts Institute of Technology, 2007. URL: `https://dspace.mit.edu/handle/1721.1/41541` (visited on 09/22/2019).

[68] Mario Valenti et al. “Mission Health Management for 24/7 Persistent Surveillance Operations”. In: 2007. DOI: `10.2514/6.2007-6508`.

[69] Milo C Silverman et al. “Staying Alive Longer: Autonomous Robot Recharging Put to the Test”. In: (), p. 7.

[70] K. A. Swieringa et al. “Autonomous battery swapping system for small-scale helicopters”. In: *2010 IEEE International Conference on Robotics and Automation*. 2010 IEEE International Conference on Robotics and Automation. May 2010, pp. 3335–3340. DOI: `10.1109/ROBOT.2010.5509165`.

[71] *Flying Machine Enabled Construction*. URL: `https://idsc.ethz.ch/research-dandrea/research-projects/archive/flying-machine-enabled-construction.html` (visited on 09/22/2019).

[72] Jeremie Leonard, Al Savvaris, and Antonios Tsourdos. “Energy Management in Swarm of Unmanned Aerial Vehicles”. In: *Journal of Intelligent & Robotic Systems* 74.1 (Apr. 1, 2014), pp. 233–250. ISSN: 1573-0409. DOI: `10.1007/s10846-013-9893-8`. URL: `https://doi.org/10.1007/s10846-013-9893-8` (visited on 09/22/2019).

[73] André Kurs et al. “Wireless Power Transfer via Strongly Coupled Magnetic Resonances”. In: *Science* 317.5834 (July 6, 2007), pp. 83–86. ISSN: 0036-8075, 1095-9203. DOI: `10.1126/science.1143254`. URL: `https://science.sciencemag.org/content/317/5834/83` (visited on 09/22/2019).

[74] Aristeidis Karalis, J. D. Joannopoulos, and Marin Soljačić. “Efficient wireless non-radiative mid-range energy transfer”. In: *Annals of Physics*. January Special Issue 2008 323.1 (Jan. 1, 2008), pp. 34–48. ISSN: 0003-4916. DOI: `10.1016/j.aop.2007.04.017`. URL: `http://www.sciencedirect.com/science/article/pii/S0003491607000619` (visited on 09/22/2019).

[75] Ali Bin Junaid, Yunseong Lee, and Yoonsoo Kim. "Design and implementation of autonomous wireless charging station for rotary-wing UAVs". In: *Aerospace Science and Technology* 54 (July 1, 2016), pp. 253–266. ISSN: 1270-9638. DOI: 10.1016/j.ast.2016.04.023. URL: http://www.sciencedirect.com/science/article/pii/S1270963816301547 (visited on 09/22/2019).

[76] Ali Bin Junaid et al. "Autonomous Wireless Self-Charging for Multi-Rotor Unmanned Aerial Vehicles". In: *Energies* 10 (June 13, 2017), p. 803. DOI: 10.3390/en10060803.

[77] *dronekit/dronekit-python*. GitHub. URL: https://github.com/dronekit/dronekit-python (visited on 09/22/2019).

[78] *Thrust ? What is Thrust?* URL: https://www.grc.nasa.gov/WWW/k-12/airplane/thrust1.html (visited on 09/22/2019).

[79] *Wayback Machine*. Feb. 27, 2009. URL: https://web.archive.org/web/20090227160410/http://www.faa.gov/library/manuals/aviation/pilot_handbook/media/PHAK%20-%20Chapter%2006.pdf (visited on 09/22/2019).

[80] *Perguntas Frequentes*. URL: https://voanaboa.pt/perguntas-frequentes (visited on 09/22/2019).

[81] *Ground sampling distance (GSD)*. Support. URL: http://support.pix4d.com/hc/en-us/articles/202559809-Ground-sampling-distance-GSD- (visited on 09/22/2019).

[82] *TOOLS - GSD calculator*. Support. URL: http://support.pix4d.com/hc/en-us/articles/202560249-TOOLS-GSD-calculator (visited on 09/22/2019).

[83] *RTL Mode — Copter documentation*. URL: http://ardupilot.org/copter/docs/rtl-mode.html (visited on 09/22/2019).

[84] *TextMagic: Bulk SMS Marketing Service Provider Since 2001*. TextMagic. URL: https://www.textmagic.com/ (visited on 09/22/2019).

[85] *Image Recognition Vs Object Detection — The Difference.* URL: https://hackernoon.com/micro-learnings-image-classification-vs-object-detection-the-difference-77110b592343 (visited on 10/12/2019).

[86] Martın Abadi et al. "TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems". In: (), p. 19.

[87] Martın Abadi et al. "TensorFlow: A system for large-scale machine learning". In: (), p. 21.

[88] Gary Bradski and Adrian Kaehler. *Learning OpenCV: Computer Vision with the OpenCV Library.* Google-Books-ID: seAgiOfu2EIC. "O'Reilly Media, Inc.", Sept. 24, 2008. 579 pp. ISBN: 978-0-596-55404-0.

[89] *Overfitting | Definition of Overfitting by Lexico.* Lexico Dictionaries | English. URL: https://www.lexico.com/en/definition/overfitting (visited on 09/22/2019).

[90] B. S. Everitt and A. Skrondal. *The Cambridge Dictionary of Statistics.* Google-Books-ID: C98wSQAACAAJ. Cambridge University Press, Aug. 19, 2010. 480 pp. ISBN: 978-0-521-76699-9.

[91] *Training Custom Object Detector — TensorFlow Object Detection API tutorial documentation.* URL: https://tensorflow-object-detection-api-tutorial.readthedocs.io/en/latest/training.html (visited on 09/22/2019).

[92] *Protocol Buffers.* Google Developers. URL: https://developers.google.com/protocol-buffers (visited on 09/22/2019).

[93] Evan. *EdjeElectronics/TensorFlow-Object-Detection-on-the-Raspberry-Pi.* original-date: 2018-06-05T03:36:26Z. Sept. 22, 2019. URL: https://github.com/EdjeElectronics/TensorFlow-Object-Detection-on-the-Raspberry-Pi (visited on 09/22/2019).

[94] S. Tansuriyong et al. “Verification experiment for drone charging station using RTK-GPS”. In: *2017 International Conference on Intelligent Informatics and Biomedical Sciences (ICIIBMS)*. 2017 International Conference on Intelligent Informatics and Biomedical Sciences (ICIIBMS). Nov. 2017, pp. 229–232. DOI: `10.1109/ICIIBMS.2017.8279762`.

[95] M. Shao and X. Sui. “Study on Differential GPS Positioning Methods”. In: *2015 International Conference on Computer Science and Mechanical Automation (CSMA)*. 2015 International Conference on Computer Science and Mechanical Automation (CSMA). Oct. 2015, pp. 223–225. DOI: `10.1109/CSMA.2015.51`.

[96] *RTK Systems - Navipedia*. URL: `https://gssc.esa.int/navipedia/index.php/RTK_Systems` (visited on 09/22/2019).

[97] *Real-Time Kinematic and Differential GPS | GEOG 862: GPS and GNSS for Geospatial Professionals*. URL: `https://www.e-education.psu.edu/geog862/node/1828` (visited on 09/22/2019).

[98] *Introduction to Network RTK*. URL: `http://www.wasoft.de/e/iagwg451/intro/introduction.html` (visited on 09/22/2019).

[99] *Reach M+ - Emlid Store*. URL: `https://store.emlid.com/product/reachm-plus/` (visited on 09/22/2019).

[100] *Reach RS+*. Emlid Store. URL: `https://store.emlid.com/product/reachrs-plus/` (visited on 09/22/2019).

[101] Jong-Hyuk Kim, Salah Sukkarieh, and Stuart Wishart. “Real-Time Navigation, Guidance, and Control of a UAV Using Low-Cost Sensors”. In: *Field and Service Robotics*. Ed. by Shin’ichi Yuta et al. Vol. 24. Berlin/Heidelberg: Springer-Verlag, 2006, pp. 299–309. ISBN: 978-3-540-32801-8. DOI: `10.1007/10991459_29`. URL: `http://link.springer.com/10.1007/10991459_29` (visited on 09/22/2019).

[102] Brent Manuel, B Surv, and Rafe Penington. “ASSESSING THE ACCURACY AND INTEGRITY OF RTK GPS BENEATH HIGH VOLTAGE POWER LINES.” In: (2001), p. 12.

[103] *First setup - Reach RS/RS+ docs.* URL: `https://docs.emlid.com/reachrs/quickstart/first-setup/` (visited on 09/22/2019).

[104] *Base and Rover setup - Reach RS/RS+ docs.* URL: `https://docs.emlid.com/reachrs/quickstart/base-rover-setup/` (visited on 09/22/2019).

[105] T. Campi et al. "Magnetic field levels in drones equipped with Wireless Power Transfer technology". In: *2016 Asia-Pacific International Symposium on Electromagnetic Compatibility (APEMC)*. 2016 Asia-Pacific International Symposium on Electromagnetic Compatibility (APEMC). Vol. 01. May 2016, pp. 544–547. DOI: `10.1109/APEMC.2016.7522793`.

[106] *SITL Simulator (Software in the Loop) — Dev documentation.* URL: `http://ardupilot.org/dev/docs/sitl-simulator-software-in-the-loop.html` (visited on 09/27/2019).

[107] *MAVProxy — MAVProxy 1.6.4 documentation.* URL: `http://ardupilot.github.io/MAVProxy/html/index.html` (visited on 09/27/2019).

[108] *Project Jupyter.* URL: `https://www.jupyter.org` (visited on 09/29/2019).