NOVA SCHOOL OF SCIENCE & TECHNOLOGY

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

AFONSO BRITO CALDEIRA VIEGAS TAVARES

BSc in Electrical and Computer Engineering

PREVENTION OF FOREST FIRES USING COMPUTER VISION

MASTER IN ELECTRICAL AND COMPUTER ENGINEERING

NOVA University Lisbon

September, 2024

NOVA SCHOOL OF
SCIENCE & TECHNOLOGY

DEPARTMENT OF ELECTRICAL
AND COMPUTER ENGINEERING

PREVENTION OF FOREST FIRES USING COMPUTER VISION

AFONSO BRITO CALDEIRA VIEGAS TAVARES

BSc in Electrical and Computer Engineering

Adviser: Daniel de Matos Silvestre
Assistant Professor, NOVA University Lisbon

MASTER IN ELECTRICAL AND COMPUTER ENGINEERING

NOVA University Lisbon
September, 2024

Prevention of forest fires using computer vision

This document was created with the (pdf/Xe/Lua)LaTeX processor and the NOVAthesis template (v7.0.3) (**novathesis-manual**).

Acknowledgements

First and foremost, I would like to express my sincerest gratitude to my thesis adviser, Professor Daniel Silvestre, for their invaluable guidance, and constant support throughout the course of this research. Even when I was quiet for months, his insightful and positive feedback never stopped, which were instrumental in shaping this work.

I would also like to extend my heartfelt thanks to my partner in crime, Margarida Ferro. With her support, patience and smile this experience that is college was a lot more smoother. I would never reach this far without the countless hours of study with her, helping me focus when needed and take a break when it was too much.

To my family, especially my parents and brothers, thank you for your unwavering belief in me. Your encouragement, understanding, and patience (and occasional nagging) kept me motivated throughout this difficult journey.

Finally, I am incredibly grateful for all the friends that I have, but there is one special group of friends I must thank, the Quarendrena. Over the past six years, we have shared many laughs, overcome many challenges, and they have never stopped encouraging me throughout the process of developing my dissertation, and for that I am grateful.

This work was partially supported by the Portuguese Fundação para a Ciência e a Tecnologia through project FirePuma (https://doi.org/10.54499/PCIF/MPG/0156/2019), through LARSyS FCT funding (DOI: 10.54499/LA/P/0083/2020, 10.54499/UIDP/50009/2020, and 10.54499/UIDB/50009/2020) and through COPELABS, University Lusófona project 10.54499/UIDB/ 04111/2020.

”

“You only find out who is swimming naked when the tide goes out”

— Warren Buffett, Berkshire Hathaway’s shareholder letter of 2001
(American businessman, investor, and philanthropist)

Abstract

The increase in both the number and size of fires demonstrates an urgent need for new and innovative interventions, not only in firefighting, but also in prevention, since firefighting is far more efficient when combined with improved prevention strategies. One of the most effective methods of preventing the spread of fires is to ensure that fields, especially tree plantations, are regularly trimmed and cleaned, preventing fire from spreading from the ground to the tree canopy, making it tremendously difficult to control and extinguish a fire.

The use of videos of tree plantations taken by an Unmanned Aerial Vehicle (UAV) and the subsequent creation of an algorithm that processes them, with the use of computer vision and OpenCV algorithms, has the effect of significantly reducing the laborious task of checking whether parts of a field are cleaned. This process eliminates the need for human intervention and reliance on a database of images of trees to create a neural network model.

The script implemented is designed to process and compare two videos, one representing a clean field of a tree plantation and the other representing the same plantation after some time (with potential overgrowth). The objective is to identify tree trunks that are occluded by vegetative growth and require cleaning.

Although further testing of the algorithm is required in order to guarantee its efficacy in a broader range of scenarios and with a greater quantity of data, the results of this work are promising.

Keywords: Computer vision, OpenCV, Fires, Fire Prevention, Tree Plantation

Resumo

O aumento do número e da dimensão dos incêndios demonstra a necessidade urgente de intervenções novas e inovadoras, não só no combate aos incêndios, mas também na prevenção, uma vez que o combate aos incêndios é muito mais eficiente quando combinado com estratégias de prevenção melhoradas. Um dos métodos mais eficazes para evitar a propagação de incêndios consiste em assegurar que os campos, especialmente as plantações de árvores, sejam regularmente aparados e limpos, impedindo que o fogo se propague do solo para a copa das árvores, o que dificulta enormemente o controlo e a extinção de um incêndio.

A utilização de vídeos de plantações de árvores captados por um drone e a subsequente criação de um algoritmo que os processa, com recurso a computer vision e algoritmos OpenCV, tem como efeito reduzir significativamente a laboriosa tarefa de verificar se partes de um campo estão limpas. Este processo elimina a necessidade de intervenção humana e a dependência de uma base de dados de imagens de árvores para criar um modelo de rede neuronal.

O script implementado foi concebido para processar e comparar dois vídeos, um que representa um campo limpo de uma plantação de árvores e outro que representa a mesma plantação após algum tempo (com potencial crescimento excessivo). O objetivo é identificar os troncos das árvores que estão ocluídos pelo crescimento vegetativo e que necessitam de limpeza.

Embora sejam necessários mais testes do algoritmo para garantir a sua eficácia numa gama mais vasta de cenários e com uma maior quantidade de dados, os resultados deste trabalho são promissores.

Palavras-chave: Computer Vision, OpenCV, Incêndios, Prevenção de Incêndios, Plantação de Árvores

Contents

List of Figures viii

Acronyms ix

1 Introduction 1
- 1.1 Motivation . . . 1
- 1.2 Objective . . . 2
- 1.3 Thesis Structure . . . 2

2 Literature Review 3
- 2.1 Computer vision . . . 3
- 2.2 Image processing . . . 3
  - 2.2.1 Pixel transforms . . . 4
  - 2.2.2 Color transforms . . . 4
  - 2.2.3 Linear filters . . . 4
  - 2.2.4 Non-linear filters . . . 5
  - 2.2.5 Optical Flow . . . 5
  - 2.2.6 Hough transform . . . 6
- 2.3 Edge detection . . . 7
  - 2.3.1 Roberts cross operator . . . 7
  - 2.3.2 Sobel operator . . . 8
  - 2.3.3 Canny edge detector . . . 8
- 2.4 Forest fire prevention techniques . . . 9
  - 2.4.1 Rudimentary techniques . . . 9
  - 2.4.2 Advanced techniques . . . 10

3 Methodology 14
- 3.1 Data Collection and Preparation . . . 14
- 3.2 Algorithm Design . . . 14
  - 3.2.1 Frame Processing . . . 15

3.2.2 Trunk Tracking . . . 16
3.2.3 New Trunk Detection . . . 17
3.2.4 Frame Comparison . . . 18
3.3 Restrictions . . . 19

4 Implementation **21**
4.1 Copy List of Lists Function . . . 21
4.2 Clean Two Arrays Function . . . 21
4.3 Touch Image Sides Function . . . 21
4.4 Median Width Trunk Function . . . 21
4.5 Find Potential Trunks Function . . . 22
4.6 Clean Trees Function . . . 26
4.7 Increase Rectangle Size Function . . . 27
4.8 Set Trunk ID Function . . . 27
4.9 Follow Function . . . 27
4.10 Buffering Begin Function . . . 29
4.11 Middle Trunks Function . . . 30
4.12 Compare Trunks Function . . . 31
4.13 Main Function . . . 32

5 Discussion **33**
5.1 Interpretation of Results . . . 33
5.2 Limitations of the Study . . . 36

6 Conclusions and Future Work **37**
6.1 Conclusion . . . 37
6.2 Future Work . . . 38

Bibliography **39**

List of Figures

2.1 Image demonstration of firebreak, fuelbreak and shaded fuelbreak Ascoli et al. 2018. . . . 10
2.2 Illustration of the proposed fire detection algorithm of paper Yuan, Liu, and Zhang 2017. . . . 12
2.3 Flowchart of fire detection algorithms of paper Yuan, Liu, and Zhang 2016. 13

3.1 Flowchart of the Algorithm Design. . . . 15
3.2 Flowchart of the Frame Processing process. . . . 15
3.3 Flowchart of the Trunk Tracking process. . . . 17
3.4 Flowchart of the New Trunk Detection process. . . . 18
3.5 Flowchart of the Frame Comparison process. . . . 19

4.1 Trunks, which exhibit disparate heights but comparable widths, being identified with red rectangles. . . . 22
4.2 Example of the process to find the vertical sides of potential trunks. . . . 23
4.3 First trunk identified. . . . 24
4.4 Second trunk identified. . . . 25
4.5 Rectangles identifying trunks. . . . 25
4.6 Example with the dilated image. . . . 26
4.7 Example with the original image. . . . 26
4.8 Example of Canny edge detection algorithm application. . . . 28
4.9 Example of dilation function result after Canny function. . . . 28
4.10 Examples of the function process. . . . 31

5.1 ID and percentage of coverage of each trunk. The ID indicates the frame and the side of the frame at which the trunk was initially detected. . . . 34
5.2 Return of the script that specifies the time stamp and the side of the image in which each trunk that requires to be checked was found. . . . 34
5.3 Frame from reference video, which shows three identified trunks. . . . 35
5.4 Frame from overgrown video, which shows three identified trunks. . . . 35

Acronyms

AGBM Above Ground Biomass Models

CHM Canopy Height Models

DTM Digital Terrain Models

LiDAR Laser imaging, Detection, and Ranging

RGB Red, Green and Blue

UAV Unmanned Aerial Vehicle

1 Introduction

Every year there are new occurrences of larger and dangerous fires all over the world MacCarthy et al. 2024; Westerling 2016. With climate change, there are areas being subject to higher temperatures and for longer periods of time like never before Moritz et al. 2014; Jolly 2015; Barbero et al. 2015; Stephens et al. 2013. These conditions make it easier for fires to ignite and spread faster without any intervention, meaning that farmers and governments should invest time and resources not only to put out fires, but also to prevent big and uncontrollable fires for ever happening.

As more and mores studies are conducted on forest fire prevention Stephens et al. 2013; Ascoli et al. 2018; Agee and Skinner 2005; Green 1977; Agee et al. 2000; Fernandes and Botelho 2003, experts tend to agree that prevention may be the best way to stop a small fire from becoming a big one. It is easy to see this agreement in real life, as many countries are starting to pass laws that force people to take some preventative measures. One of the most important is to keep the land clean, because it is easier for fires to spread when vegetation has grown out of proportion, with overgrown shrubs, dead branches on the ground and trees with low branches. All of this together makes it possible for fire to reach tree tops and consequently make it almost impossible for firefighters to control and extinguish the fire.

1.1 Motivation

A significant portion of the land is divided up into large chunks of tree plantations, which are too vast for human verification in terms of available forest fuel. The solution decided upon is to make use of an Unmanned Aerial Vehicle (UAV) equipped with a camera, making it a quick and easy way to check all the land, and discarding the need for anyone to go out in the field.

The problem with the use of a UAV is that there is still the need to have a pilot fly the UAV and to have someone watch the camera in order to check if there are parts of the land that need to be cleaned up, both of whom are spending their time on a long and tedious task. To address the necessity of a pilot, the deployment of an autonomous UAV is a

completely viable solution, as there are already numerous algorithms and options on the market Lu et al. 2018; Kanellakis and Nikolakopoulos 2017; Shakhatreh et al. 2019. Finally, the problem that remains and that will be worked on in this paper is the identification of possible uncleaned areas of the land autonomously, more specifically, the area around tree trunks.

1.2 Objective

In light of the motivation described, to ensure cost-effectiveness and feasibility for future users, the UAV is equipped with only a camera that uses Red, Green and Blue (RGB) channels to record the tree plantation, after which the program is run on a standard computer. It was also determined that the development of an algorithm that prioritizes speed and low computer power would facilitate the identification of vegetation growth around tree trunks. Given the lack of suitable databases of tree trunk images of sufficient quality and size for the creation of a neural network, it was decided to focus on computer vision techniques.

1.3 Thesis Structure

This section provides a breakdown of each chapter and its role in the research. The following is a list of the chapters that compose this paper:

Chapter 1 establishes the context for the research by providing a small context of the fire situation, and the motivation behind the research, highlighting the importance of it. It also clarifies the objective of the study, and outlines the structure of the thesis to give the reader an overview of it.

Chapter 2 provides a comprehensive overview of the existing knowledge and previous research relevant to this study. It covers relevant concepts in computer vision, image processing, and forest fire prevention techniques, thereby establishing the theoretical foundation for the research.

Chapter 3 outlines the approach used in the processes undertaken to gather and prepare the data, as well as the specific techniques developed in the algorithm design.

Chapter 4 details the development and implementation of the functions and algorithms used in the study.

Chapter 5 analyzes the results, providing insights into the effectiveness and implications of the research, and addresses the limitations encountered.

Chapter 6 provides a final reflection on the contributions of the study to the field and presents options for future studies.

2

Literature Review

2.1 Computer vision

In recent years, around the 1950s McCarthy et al. 1955; Moor 2006, the scientific field began to turn its attention towards artificial intelligence and the creation of machines with similar skills and abilities to humans, either mental or physical.

The human visual system is one of the more complicated ones as it needs not only the hardware part for capturing what is being seen, but also the software part that interpret and identify visual information from digital images. For this field of artificial vision, it was given the name of computer vision, and since then it is a wildly studied field Szeliski 2011; David A. Forsyth 2002; Bovik 2000, 2009.

In computer vision programs, the software for understanding digital images involves the development and implementation of algorithms and techniques that enable machines to perceive and understand visual data, in a similar way humans do with their visual senses.

Computer vision algorithms analyze and extract meaningful information from images, such as objects, shapes, patterns, and even human gestures and facial expressions. This technology has many applications, including object recognition, image classification, image segmentation, motion detection, and even advanced tasks such as autonomous vehicles and facial recognition systems.

By harnessing computer vision, machines can effectively interpret and interact with the visual world, opening up a wide range of possibilities for automation, surveillance, medical imaging, augmented reality, and much more.

2.2 Image processing

A digital image is a matrix (2.1) of colour values, and in order to process such an image it is necessary to change either the position or the colour values of each pixel, or both. For the position processes, it is called pixel transforms, and for the colour processes, it is called colour transforms. There are other ways to process images, such as filters.

$$f = \begin{bmatrix} f(1,1) & f(1,2) & ... & f(1,N) \\ f(2,1) & f(2,2) & ... & f(2,N) \\ \vdots & \vdots & & \vdots \\ f(M,1) & f(M,2) & ... & f(M,N) \end{bmatrix}. \tag{2.1}$$

In this section, some of this processes are going to be explained, in order to have some better comprehension of what is going to be used to solve the problem of this thesis.

2.2.1 Pixel transforms

Pixel transforms are ways to alter the image positioning. Whether you want to rotate or scale the image, it is going to be with the use of a pixel transform. To explain in simple terms, this type of transforms change the position values (x and y for 2d images) of each pixel of the digital image.

2.2.2 Color transforms

Color transforms are ways to alter the image color. Besides their position, each pixel have 3 values, the value of red, R, green, G, and blue, B.

Just like the pixel transforms, this type of transforms changes the values of the pixels, but instead of position values, it changes the RGB values.

2.2.3 Linear filters

This type of filters have this name because they apply a linear transformation to the image pixels. By making use of a kernel, the filters perform a convolution operation to the image pixels. This convolution between an image (I) and a kernel (K) at a random position (x, y) of an image is usually represented by the equation (Chapter 10 of Bovik 2009):

$$I'(x,y) = \sum K(a,b) * I(x-a, y-b), \tag{2.2}$$

where I'(x, y) is the new value of the pixel, I(x-a, y-b) is the pixel value of the original image, and K(a, b) is the kernel value.

One of the main characteristics of linear filters is the fact that the result of applying several linear filters to an image is equivalent to applying a single filter obtained by the convolution of the individual kernels. This property allows the efficient combination of several filters for more complex image processing operations.

Linear filters are widely used in both traditional image processing algorithms and modern computer vision applications, and form the basis of many advanced image filtering techniques.

2.2.4 Non-linear filters

Unlike linear filters, non-linear filters do not rely on convolution operations with a fixed kernel. This is a type of filter that applies a non-linear transformation to the image pixels.

In order to obtain new pixel values, non-linear filters use local image statistics or neighbourhood information. They are often used to enhance or suppress specific image features. They can also be used to filter noise or preserve edges.

2.2.5 Optical Flow

Optical flow is the basic concept that is behind the study and understanding of object movement in computer vision. This concept is assuming that neighboring pixels in an image tend to move together over time. There are two assumptions that are made for optical flow: that the pixels of an object remain at the same intensity between two successive frames, and that the neighbouring pixels have similar motion. By analyzing the displacement of pixels between two consecutive frames, we can estimate the velocity and motion vectors of objects that are in the frames. To calculate these characteristics the follow equation OpenCV 2023 is used:

$$\frac{\partial f}{\partial x}u + \frac{\partial f}{\partial y}v + \frac{\partial f}{\partial t} = f_x u + f_y v + f_t = 0. \tag{2.3}$$

The function f_x and f_y are image gradients and f_t is the gradient during time. Since the Optical Flow equation 2.3 has two variables (u, v) that are unknown, it is necessary to find one or more equations to solve this problem. The most common way to solve this problem is by using the Lucas-Kanade method. There are other ways to solve this issue, such as the Horn-Schunck method or variational techniques, that address these limitations.

2.2.5.1 Lucas-Kanade method

The Lucas-Kanade method Lucas and Kanade 1981, developed by Bruce D. Lucas and Takeo Kanade, solves the basic optical flow equations of the current neighboring pixels by assuming that the flow is essentially constant, meaning it is assumed that the pixel intensity remains constant within a small neighborhood in the image over time. It calculates the optical flow vectors by solving a system of equations derived from this assumption.

The most common size to the neighborhood used is a 3 by 3 square, which means that there will be 9 equations to solve with two unknown variables. In order to simplify this, it is used the least square fit method, which gives the follow equation:

$$\begin{bmatrix} u \\ v \end{bmatrix} = - \begin{bmatrix} \sum_i f_{x_i} f_{t_i} \\ \sum_i f_{y_i} f_{t_i} \end{bmatrix} \begin{bmatrix} \sum_i f_{x_i}^2 & \sum_i f_{x_i} f_{y_i} \\ \sum_i f_{x_i} f_{y_i} & \sum_i f_{y_i}^2 \end{bmatrix}^{-1}, \tag{2.4}$$

where i is a range, that specifies the size of the area chosen for the pixel neighbourhood. As mentioned above, the most common area is a 3 by 3 square, so in this case the range of i is 1 to 9.

It is important to note that the Lucas-Kanade method assumes small displacements between frames and does not handle large displacements or occlusions well.

2.2.5.2 Horn-Schunck method

The Horn-Schunck method Horn and Schunck 1981, developed by Berthold K.P. Horn and Brian G. Schunck, is an optical flow algorithm used to estimate the motion field in a sequence of images. It is a popular technique in computer vision that assumes smoothness and consistency of motion across the image.

This method provides a global estimation of optical flow, and instead of just looking to a local neighbourhood, it takes into account the entire image. To achieve this, and to discover the velocity variables (u, v) the following equations are used:

$$\begin{aligned} u^{n+1} &= \bar{u}^n - \frac{f_x[f_x\bar{u}^n + f_y\bar{v}^n + f_t]}{\alpha^2 + f_x^2 + f_y^2}, \\ v^{n+1} &= \bar{v}^n - \frac{f_y[f_x\bar{u}^n + f_y\bar{v}^n + f_t]}{\alpha^2 + f_x^2 + f_y^2}. \end{aligned} \tag{2.5}$$

It is worth noting that the Horn-Schunck method has more trouble in handling complex motion patterns and, just like the Lucas-Kanade method, large displacements. As mentioned above, this method assumes that the motion field is smooth and continuous, so the smoother the sequence of images, and the less abrupt the changes or occlusions, the better.

2.2.6 Hough transform

The Hough transform Illingworth and Kittler 1988; Burnham, Hardy, and Meadors 1995 is used for detecting and extracting simple geometric shapes, particularly lines, from an image. It works by transforming the image space into a parameter space, where the parameters represent the geometric properties of the shapes being detected.

For line detection, the parameter space is typically defined as the intercept form of a line equation: $y = mx + b$. Each pixel on an edge of the image is transformed into a curve in the parameter space. The intersection point of these curves represents a line in the original image. By finding the maximum in the parameter space, the Hough transform can identify lines that pass through a sufficient number of edge points.

The Hough transform is robust to noise and partial occlusions because it accumulates evidence from all edge points in the image. However, it can be computationally expensive, especially for images with a large number of edge points.

2.3 Edge detection

Edge detection Ziou and Tabbone 1998 is a fundamental technique in computer vision and image processing that identifies the boundaries of objects or regions within an image. It plays an important role in object recognition, image segmentation and feature extraction.

The process of edge detection involves analysing the intensity or colour variations between pixel values of an image, in order to identify significant changes. These changes are normally identified as transitions between different objects or regions.

This type of algorithms use different methods to detect transitions. One common approach is with the use of gradient. This method calculates the gradient magnitude and direction of each pixels intensity.

The gradient method is so good that advanced edge detection methods such as the Canny Edge Detector use it. The Canny edge detector uses multiple stages of smoothing, gradient calculation, non-maximum suppression and hysteresis thresholding. The Canny detector is renowned for its effectiveness in detecting and accurately locating edges while minimising noise and false detections.

In summary, edge detection is an essential part of many image analysis tasks, as it provides crucial information about the structure and boundaries of objects. With its use, it becomes possible to extract features, segment objects, perform shape analysis, and ultimately enable better computer vision algorithms to understand and interpret visual information.

2.3.1 Roberts cross operator

The Roberts cross operator Burnham, Hardy, and Meadors 1995; Shrivakshan and Chandrasekar 2012 is a simple edge detection algorithm commonly used in image processing.

The Roberts cross operator works by calculating the gradient magnitude of an image. It uses the following two 2x2 kernels:

$$G_x = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}, G_y = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}. \tag{2.6}$$

These kernels are convolved with the pixel values of the image. It then calculates the gradient magnitude $G(x, y)$ for each pixel as the square root of the sum of the squared horizontal and vertical gradient values:

$$G(x, y) = \sqrt{G_x^2 + G_y^2}. \tag{2.7}$$

The result of the equation 2.7 is the strength of the edge calculated. If it is an edge, it will have a high value. The angle of the gradient can also be calculated to determine the direction of the edges. This is discovered using the following equation:

$$\theta = atan2(G_y, G_x). \tag{2.8}$$

Although it is computationally efficient, it can be sensitive to noise due to its small kernel size.

2.3.2 Sobel operator

Like the Roberts operator, it calculates the gradient magnitude of an image to identify edges. However, the Sobel operator Burnham, Hardy, and Meadors 1995; Shrivakshan and Chandrasekar 2012 uses larger kernels and incorporates a weighted averaging scheme, which results in more robust edge detection.

The two kernels used, one for the horizontal gradient G_x and another for the vertical gradient G_y, are defined as follows:

$$G_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}, G_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}. \tag{2.9}$$

To apply the Sobel operator, G_x is convolved with the image in the horizontal direction, and G_y is convolved with the image in the vertical direction. The resulting convolutions produce two gradient images representing the horizontal and vertical gradients.

The gradient magnitude $G(x, y)$ at each pixel is then calculated as:

$$G(x, y) = \sqrt{G_x^2 + G_y^2}. \tag{2.10}$$

This represents the strength of the edges in the image. The angle of the gradient is discovered the same way as in the roberts operator 2.3.1 equation 2.8.

The Sobel operator is more robust to noise compared to the Roberts operator because of its larger kernels and weighted averaging scheme.

2.3.3 Canny edge detector

The canny edge detector Burnham, Hardy, and Meadors 1995; Canny 1986 is known for its effectiveness in detecting edges while minimizing noise and accurately localizing edges in an image.

This algorithm consists of the following steps:

1. Noise Reduction - A Gaussian filter is applied to the image to reduce noise. The filter smooths the image by convolving it with a Gaussian kernel.

2. Gradient Calculation - This next step is to calculate the gradient magnitude and orientation at each pixel in the smoothed image. This is achieved by convolving the image with an edge detection operator in the horizontal and vertical directions, just as it was mentioned in subsections 2.3.1 and 2.3.2.

3. Non-Maximum Suppression - After getting the gradient magnitude image, it is applied a non-maximum suppression to thin out the edges. It involves iterating over each pixel and suppressing non-maximum values in the local neighbourhood along the gradient direction.

4. Double Threshold - This step identifies which edge pixels are going to be identified as strong, weak, or suppressed based on gradient magnitude values. Pixels with gradient magnitudes above a high threshold are considered strong edges, those below a low threshold are suppressed, and those with values between the high and the low thresholds are labeled as weak edges.

5. Edge Tracking by Hysteresis - In this final step, it is verified if weak edges are actually part of edges. This is done by making sure that a weak edge is connected to high edges. If this is not the case and the weak edge is isolated, it is assumed to be just noise.

The Canny edge detector is deemed highly for the way it provides accurate and essential edges, without being affected by the noise.

2.4 Forest fire prevention techniques

Due to climate change, wildfires are an important topic in the scientific community and in every other field, as larger and larger fires seem to happen every year. As a result, governments have taken it upon themselves to pass laws Interior n.d.; Commission n.d. that attempt to prevent wildfires. Some of these preventative measures use rudimentary techniques and others use more advanced and technological techniques.

This section explains how some of these techniques work.

2.4.1 Rudimentary techniques

When talking about rudimentary wildfire prevention techniques, I am referring to techniques that do not necessarily require technology to implement, such as firebreaks and prescribed burns Ascoli et al. 2018; Agee and Skinner 2005; Green 1977; Agee et al. 2000; Fernandes and Botelho 2003. Most of these types of techniques were used long before technology appeared and are still used today, which is an indication of their usefulness and effectiveness.

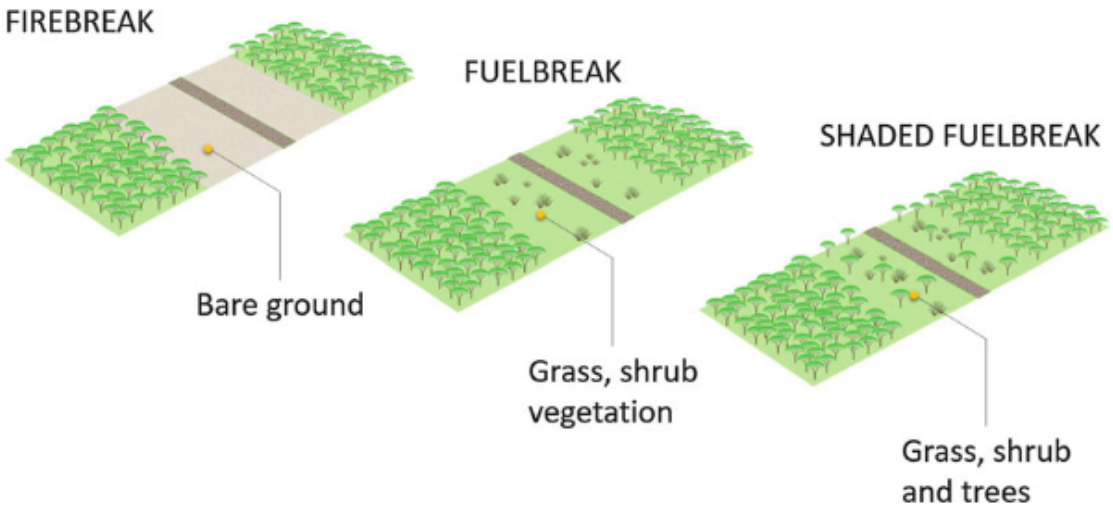


Figure 2.1: Image demonstration of firebreak, fuelbreak and shaded fuelbreak Ascoli et al. 2018.

As it can be seen in the figure 2.1, there are three types of fuel-managed areas:

- shaded fuelbreaks, where there are still trees and bushes, but in very small numbers and maintained at a low density;
- fuelbreaks, where there is only grass and shrub vegetation, but in small numbers;
- firebreaks, where there is nothing to fuel a fire, not even grass;

Shaded fuelbreaks and fuelbreaks are used to reduce the size and intensity of a fire, making it easier for firefighters to extinguish it. Firebreaks are used to prevent fires from spreading, thus reducing their size.

Although controversial, many experts believe that prescribed burns are one of the ways to contain and even prevent large, uncontrolled wildfires of ever starting Agee and Skinner 2005; Fernandes and Botelho 2003. The reason being that these burns clean the prescribed areas of any fuel for future wildfires, such as vegetation and trees, and prevent them from spreading and growing even further, just like a fuelbreak. These prescribed burns can also become a permanent fuelbreak as it is easier to maintain and get rid of any vegetation that tries to grow.

In addition to firebreaks and controlled burns, it is also important in fire-prone areas to keep vegetation as close to the ground as possible, to keep tree canopies away from the ground by cutting low hanging branches, and to clear the land of any rubbish that that could cause or exacerbate fires, such as glass and chemicals Agee and Skinner 2005.

2.4.2 Advanced techniques

As wildfires become a greater concern for the safety of the general public, more studies using technological advancements are being conducted, resulting in the creation

of techniques and equipment that help and improve the prevention of wildfires Fernández-Álvarez, Armesto, and Picos 2019; Dandois and Ellis 2010; Yuan, Liu, and Zhang 2017, 2016; Koetz et al. 2008; Abdollahi and Yebra 2023; Apriani, Oktaviani, and Sofian 2022.

One of the ways that computer vision is being used to prevent forest fires from spreading is by quickly identifying a fire, preferably in its early stages, and alerting the authorities.

This section shows some of the studies and techniques that are being used to prevent forest fires.

2.4.2.1 LiDAR-Based Wildfire Prevention in WUI:The Automatic Detection, Measurement and Evaluation of Forest Fuels

One of these studies Fernández-Álvarez, Armesto, and Picos 2019 presents a method for identifying trees and vegetation using Laser imaging, Detection, and Ranging (LiDAR) technology with a UAV platform. This methodology makes it possible to gather information on tree height, pruning height, and spacing, as well as shrub height and cover. It also verifies if vegetation is in compliance with the legislation of wildfire prevention.

This method takes a lot of computational power and a lot of time, as it models each tree and shrub, found with the algorithm based on Kraus and Pfeifer linear prediction, separately, and then applies three models to each.

2.4.2.2 Remote Sensing of Vegetation Structure Using Computer Vision

Another study Dandois and Ellis 2010 compares the viability of a computer vision program, that they called "Ecosynth", with a more expensive and sophisticated LiDAR-based program in the production of Digital Terrain Models (DTM), Canopy Height Models (CHM) and Above Ground Biomass Models (AGBM).

For the Ecosynth program, aerial photographs of test sites with sufficient trees and vegetation for the work were acquired and processed into a dataset of 3D point clouds using Bundler software. Once the Blunder point clouds had been geocorrected using the Helmert transformation model, there were some points that had to be filtered out due to their incorrect location using a two-step statistical outlier filter.

In order to generate the DTMs, the Kriging method was used on the ground points, after the remaining points were passed through a progressive morphological filter that separated them into ground and non-ground points. The non-ground points were then used to produce the CHMs, by finding the differences between the elevation values of the non-ground points and the ground points used for the DTMs. AGBMs were produced using standard LiDAR forestry methods.

Although the paper concluded that LiDAR models were ever so slightly better than the Ecosynth ones, it also concluded that the values gathered by the Ecosynth were very close to the LiDAR ones, especially in generating CHMs, and the technology used is way cheaper.

2.4.2.3 Fire Detection Using Infrared Images for UAV-based Forest Fire Surveillance

This paper Yuan, Liu, and Zhang 2017 explains how they managed to create a fast fire detection algorithm to conduct surveillance and, with the use of an infrared camera on a UAV, detect forest fires.

As it can be seen on figure 2.2, the proposed method starts with the capture of infrared images, then using the Otsu method it segments the hot objects, to then detect moving objects with the use of optical flow. It is then verified if it detected fire using motion vector analysis. If it did the fire is then tracked with the use of a blob counter. For this algorithm, the brightness values and the motion features of fire in infrared cameras were crucial in order to detect it with accuracy.

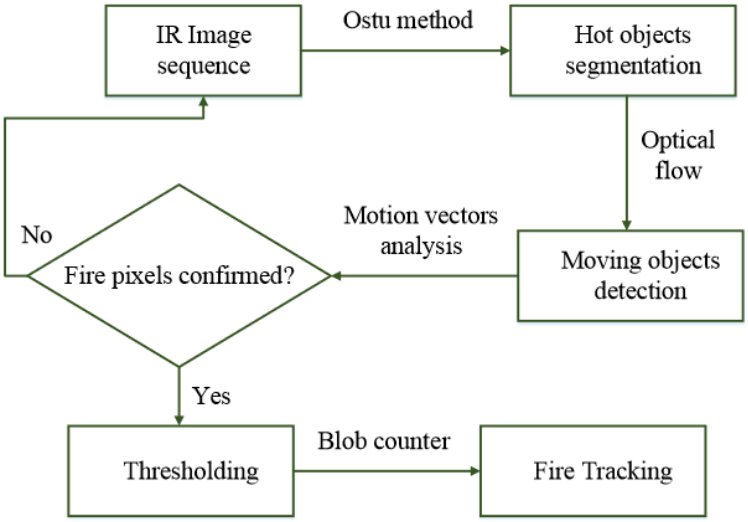


Figure 2.2: Illustration of the proposed fire detection algorithm of paper Yuan, Liu, and Zhang 2017.

Because infrared cameras show hotter things in brighter colours, the first thing it is done is the separation of the pixels that could be showing fire from the rest. So using the Otsu method, the threshold of the image is found, and then a binary image is created using the threshold found, so that the pixels that might contain valuable information become white and the rest become black. Using the binary image, a third image is created that only has the original pixel values of what might be fire and the rest remains black. This way, it is easier to analyse the image for motion details.

It is then used the Lucas-Kanade method, explained in the subsection 2.2.5.1, to find the movement of the pixels.

Since fire have a random movement pattern, and to avoid confusing it with hot moving objects such as vehicles, the orientation variation was used and checked to see if the movement previously found outweighs the orientation variation of the pixel. If this happens, it means that the pixel in question is a fire pixel, giving it a value of 1, while those that are not are set to 0. A blob counter method is then used to track and identify

the fire areas.

2.4.2.4 Vision-based Forest Fire Detection in Aerial Images for Firefighting Using UAVs

Although this paper Yuan, Liu, and Zhang 2016 introduces a different method that uses a normal RGB camera on a UAV, it is very similar to the prior paper Yuan, Liu, and Zhang 2017, as this one was also written by the same researchers and has a similar method of identifying fire, just as it can be seen by comparing the flowchart of this paper, figure 2.3, with the flowchart on figure 2.2. This method uses both color and motion characteristics of fire.

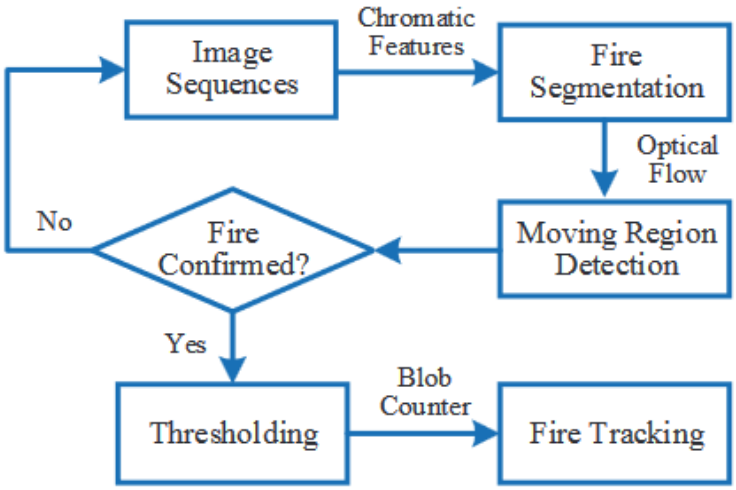


Figure 2.3: Flowchart of fire detection algorithms of paper Yuan, Liu, and Zhang 2016.

Using the L*a*b* colour model, where 'L*' stands for lightness, 'a*' for the red to green colour range and 'b*' for the yellow to blue colour range, makes it easier to extract fire colour features from the image. Optical flow, specifically the Horn-Schunck method, which was explained in the subsection 2.2.5.2, is then used for motion feature analysis to distinguish fire movement from other moving objects. Since fire has a random motion, this distinction can be made by using the variation of the optical flow velocity field and checking whether the motion feature belongs to fire. The combination of all these features makes the detection of forest fires more reliable.

In order to track the fire regions found, the paper uses the blob counter method, which tracks the number and position of fire blobs. This method involves converting the collected image into a binary image, identifying objects based on pixel connectivity, and obtaining the dimension and position of the tracked objects.

3

Methodology

The objective of this chapter is to explain the methodology employed in developing a robust and efficient automated system solution capable of identifying trunks in various environmental conditions and tracking their movement across sequential video frames. In order to achieve this objective, the findings and preparation of the data are explained in Section 3.1, separately from the algorithm design presented in Section 3.2. Section 3.2 provides a more detailed account of the image processing techniques employed and the steps taken in order to identify and track the trunks between videos.

3.1 Data Collection and Preparation

The data used for this study was sourced from a publicly available video on YouTube, specifically captured by a UAV flying through fields of trees. However, not all parts of the video were suitable for analysis. To ensure consistent and reliable detection of tree trunks, only segments where the UAV was leveled with the ground and moving parallel to the rows of trees were selected. This meant avoiding any portions of the video where the UAV tilted, turned, or changed altitude drastically, as these movements could distort the appearance of the tree trunks and negatively impact the accuracy of detection.

In addition to selecting steady footage, the environment in the video was carefully considered. Only segments showing fields of trees that were not overgrown were used. This was crucial, as the objective is to compare a clean field with the exact same but in a state of overgrowth. It is more straightforward to have a clean field and then edit it to appear as if it is overgrown.

By standardizing the video footage and applying consistent selective techniques, it was possible to achieve more accurate detection and tracking of tree trunks.

3.2 Algorithm Design

In this section it is outlined the steps and considerations involved in the algorithm's design to identify and track tree trunks in images. The algorithm aims to accurately

identify tree trunks and assess if the ground around it is cleaned. As it can be assessed in Figure 3.1, the core of the methodology used is an algorithm that includes frame processing, trunk tracking, new trunk detection and frame detection. In addition, certain restrictions were taken into account.

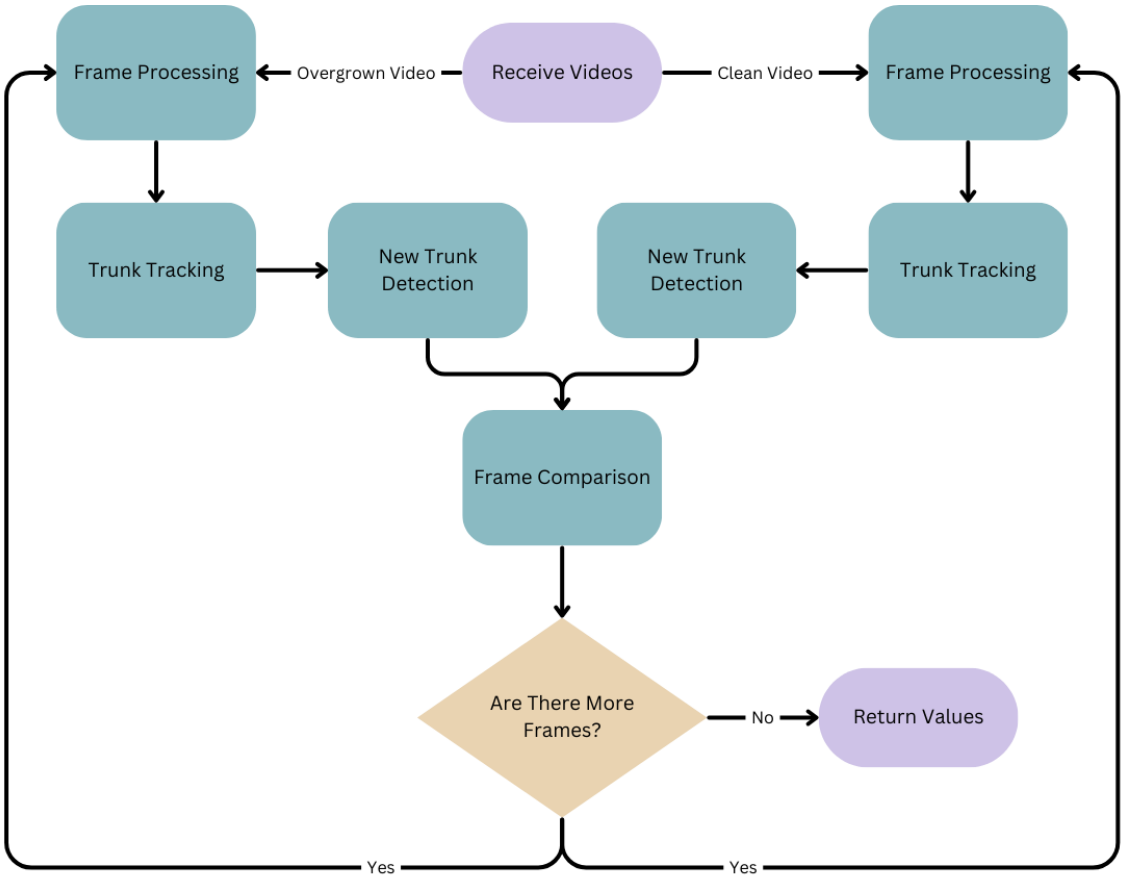


Figure 3.1: Flowchart of the Algorithm Design.

3.2.1 Frame Processing

Once the video segments that can be used are identified, the footage is then preprocessed by the program in a frame-by-frame manner, and as it can be seen in Figure 3.2 each frame goes through three processes. First and foremost, the frame is resized to a standard resolution of 854 by 480 pixels. This resolution was chosen to balance the trade-off between processing speed and detail preservation, as higher resolutions could capture more detail but would slow processing significantly, and lower resolutions could miss useful details for accurate detection.

Figure 3.2: Flowchart of the Frame Processing process.

Following resizing, each frame goes through a series of preprocessing steps aimed at enhancing the features necessary for an accurate tree trunk detection. The first step in this process is edge detection, performed with the use of the Canny edge detection algorithm with threshold values set to 150 and 200. These thresholds were selected based on preliminary tests to effectively highlight the edges of tree trunks while minimizing the detection of irrelevant edges. The output of the Canny edge detection is then further processed by applying dilation three times, using the following 3x3 kernel, consisting entirely of ones:

$$Kernel = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}. \tag{3.1}$$

The objective of dilation is to thicken the edges, thereby eliminating gaps and noise, and thus highlighting the trunks.

3.2.2 Trunk Tracking

With the frame preprocessed, it is easier to track the trunks which were first found on the previous frame or were already being tracked. This way, the zone where the trunk is can be ignored, making it easier to find new trunks on the rest of the image.

In order to perform the action mentioned above it is performed a series of processes shown in Figure 3.3. It is first checked if there are trunks previously discovered, and if so, the area to search for the trunk (this search is explained in the Subsection 3.2.3) is reduced to the bounding box of the previous frame plus a bit more to the sides, to account for the movement between frames, making the search much more accurate and faster. If, by any chance, there is no detection in the new frame, the bounding box is enlarged and moved a few pixels to the right or to the left, depending on which side of the image it is.

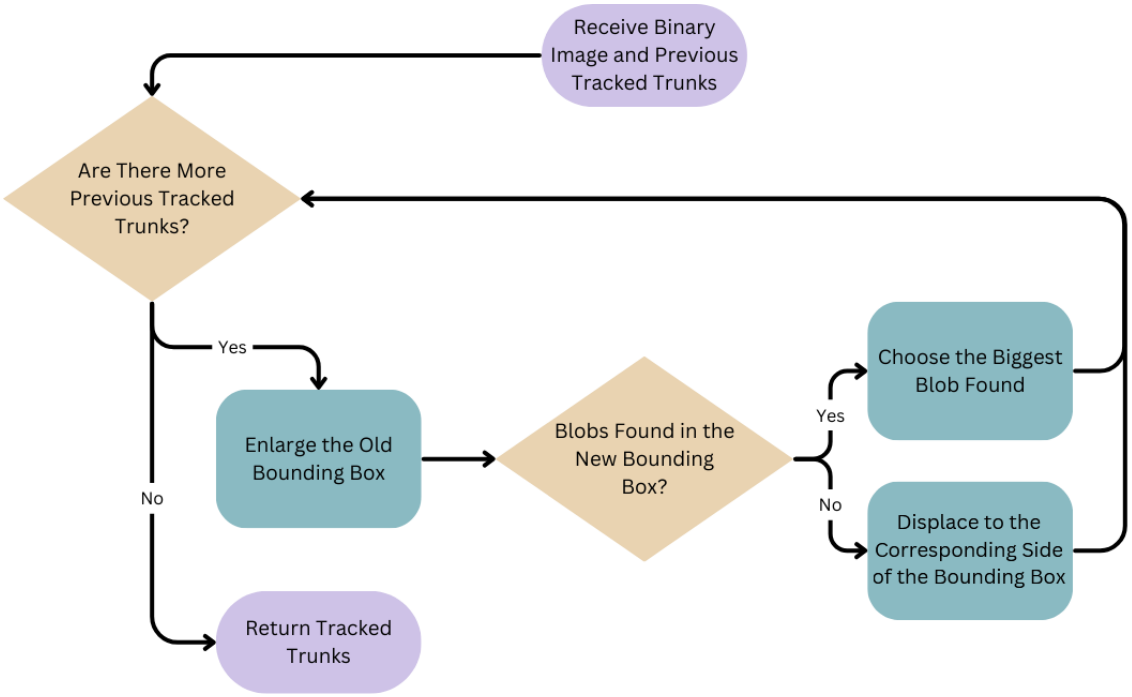


Figure 3.3: Flowchart of the Trunk Tracking process.

3.2.3 New Trunk Detection

In recognition of the possibility that trunks may already have been identified, and with a view to avoiding wasting time and resources in repeating this process, the pixels representing the areas that have been successfully detected in the processed image are all assigned the colour white. This approach ensures that only new trunks are considered for detection.

Initially, as shown on Figure 3.4, the vertical sides of the trunks are identified by detecting significant white-to-black discrepancies between columns. Each pair of discrepancies represents a trunk. To locate the horizontal sides, the largest continuous group of black pixels between the vertical pair is identified, ensuring the absence of a horizontal line of solely white pixels.

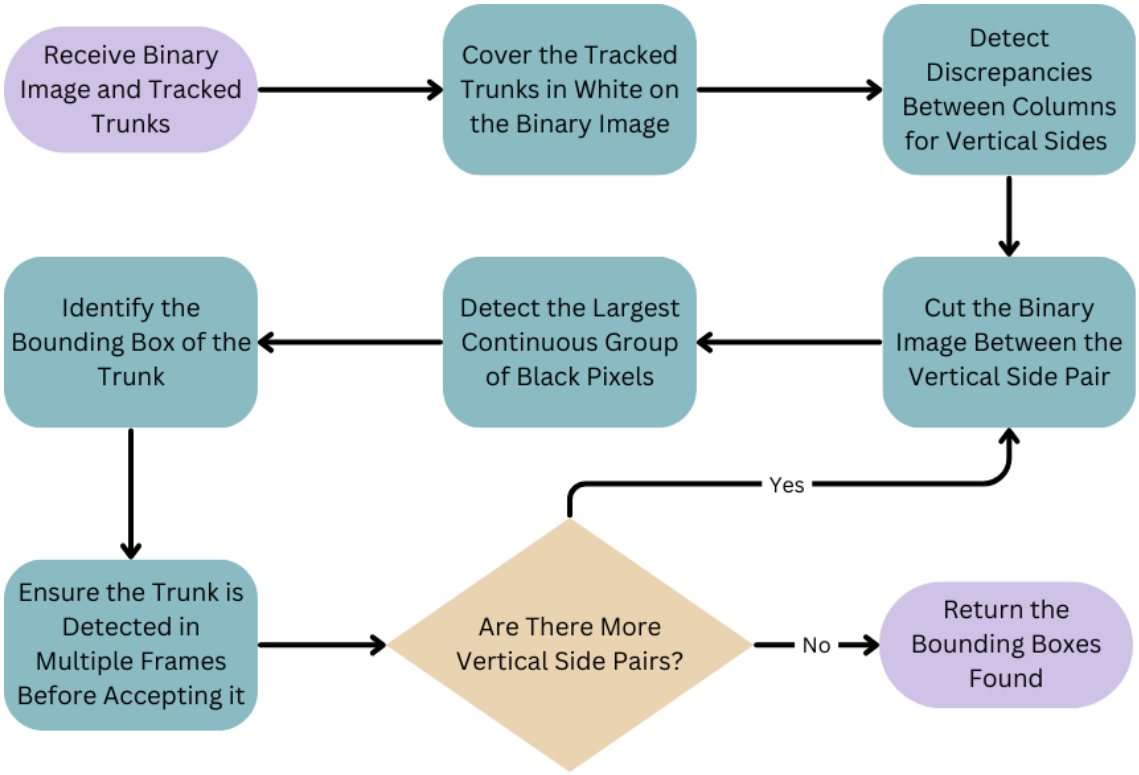


Figure 3.4: Flowchart of the New Trunk Detection process.

Subsequently, each trunk is subjected to a series of constraints, delineated in Section 3.3, to ensure it is a trunk and to exclude the possibility of it being garbage. Thereafter, an identifier is assigned to each trunk, based on its location and the side of the image on which it appears.

Before proceeding to the subsequent trunk or, in the absence of remaining trunks, to the comparison of frames (see Subsection 3.2.4), it is essential to confirm whether the current tree trunk has been consistently detected across multiple frames. Otherwise, it is held in a buffer until it can be reliably tracked across several frames.

3.2.4 Frame Comparison

Once all the trunks have been tracked and detected in both the overgrown and clean video frames, a comparison is made between the height of the trunks present in the clean frame and the height of those visible in the overgrown frame, with the process displayed in the Figure 3.5.

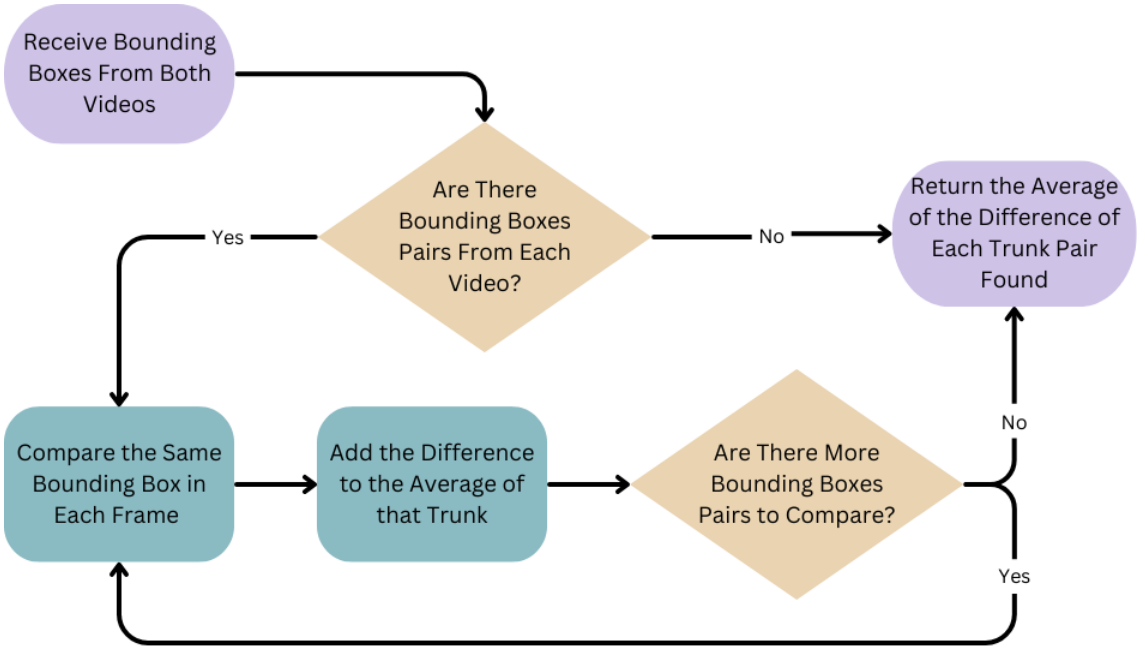


Figure 3.5: Flowchart of the Frame Comparison process.

It is important to note that only the trunks that were identified on both frames are compared. To identify these pairs, the position of the bounding box of a trunk on the clean frame is compared with the position of the bounding boxes of the trunks on the other video frame, and the closest is selected. A comparison of the heights of the bounding boxes allows for the assessment of the extent to which the trunk is covered by overgrowth, if any. Given the potential for variability in these values, the median of the difference in heights between the two trunks during the period that the trunk is visible in the frame is calculated. This value is the coverage percentage for the trunk.

For the trunks that were identified solely on the clean video, the value of 100% was attributed to them with respect to the coverage percentage.

Finally, following the processing of both videos, the operator is alerted to all trunk instances exceeding 15% coverage and is provided with the respective time stamp and side of the image. In the event that the number of trunks on the overgrown frame exceeds that of the clean frame, an operator must investigate the matter further.

3.3 Restrictions

Given that the videos in question are from tree plantations, it is normal to assume some restrictions to ensure the efficacy of the tracking and detection part of the process. The restrictions that have been implemented are as follows:

1. **Middle Section Analysis:** Only detections with their middle section within a specified horizontal stripe are considered trunks.

2. **Median Width Evaluation:** The width of a trunk is compared to the median width of previously found trunks, with the median being updated accordingly.
3. **Trunk Height Determination:** The height of potential trunks is analyzed, ensuring that it is at least twice the width for validity.
4. **Edge Touch Elimination:** Trunks that extend beyond the image boundaries are discarded and no longer tracked.
5. **Bottom of Frame Requirement:** The footage recorded has to have the bottom of the trunks in frame.
6. **Video Synchronization Necessity:** The videos that are going to be analyzed have to start at the same location and should have the same length.
7. **Consistent UAV Speed:** The speed of the UAV that records the footage has to be the same in both videos and should be around 2 meters per second.

4

Implementation

The objective of this section is intended to provide a comprehensive and detailed explanation of each function created and utilized in the script, by explaining the intended purpose, and the rationale behind the design. This is an example of how the methodology can be implemented, as any individual can decide to expand or minimize the number of functions used in their script.

4.1 Copy List of Lists Function

It creates a new and independent variable by replicating a list of lists.

4.2 Clean Two Arrays Function

The clean_two_arrays function is designed to filter out elements from an array based on their overlap with elements in another array. The elements are coordinates of rectangles, and the ones that overlap are discarded. The first array is returned without the discarded elements.

4.3 Touch Image Sides Function

This function determines whether the coordinates of the rectangles in the list intersect or exceed the edges (left or right) of the frame. In such cases, it indicates that the trunk is going to extend beyond the frame, and thus the rectangles are removed from the list, as the trunk can no longer be followed.

4.4 Median Width Trunk Function

Since the program is intended to be used in orchards or tree plantations, the width of the trees is expected to remain relatively consistent, while the heights from tree to tree have a more significant variation, as it can be seen in Figure 4.1. This function was thus

created for the purpose of comparing each detected trunk with the median width and if the trunk is within a certain interval of the median width, it is added to a list, and the remainder is discarded. The addition of these trunks to the list affects the median width, being therefore an ever-changing variable.

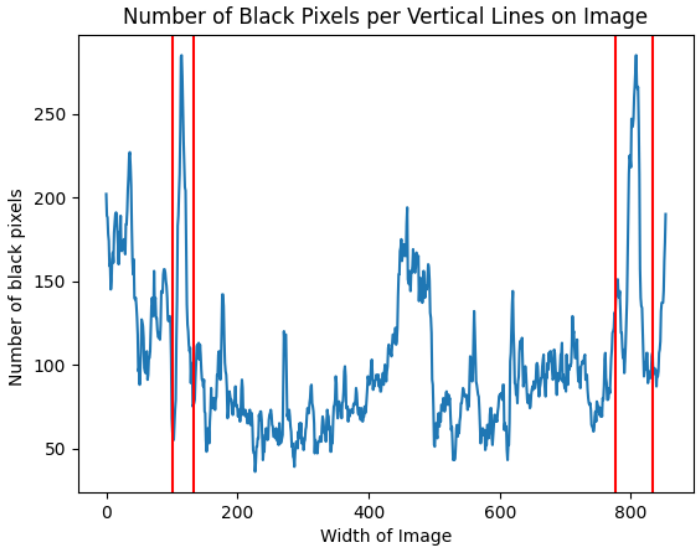
Figure 4.1: Trunks, which exhibit disparate heights but comparable widths, being identified with red rectangles.

4.5 Find Potential Trunks Function

With the input of a filtered binary image, the function creates a color inversion of it in order to count, per column, the pixels that could be potential trunk pixels. For this calculation, it uses cv2.reduce(), a function from the OpenCV library that condenses a two-dimensional matrix into a one-dimensional row or column vector. In this case, it condenses a matrix into an array that depicts the vertical distribution of zeros. With this array, it is identified spikes in the column sums, which represent sudden increases or decreases in pixel values, as it can be seen in Figure 4.2c. These spikes could be the center of potential trunks, so the positions of these spikes are stored. To reduce noise and save only significant changes, a threshold of 20% of the highest value in the array is used and if the difference between three consecutive positions is greater than the threshold, it is considered a spike. Since each spike up must be followed by a spike down, the function pairs the them to define potential trunk regions, and if the number of spikes found is odd, it ignores the last one.

(a) Binary image with vertical lines.

(b) Original image with vertical lines.



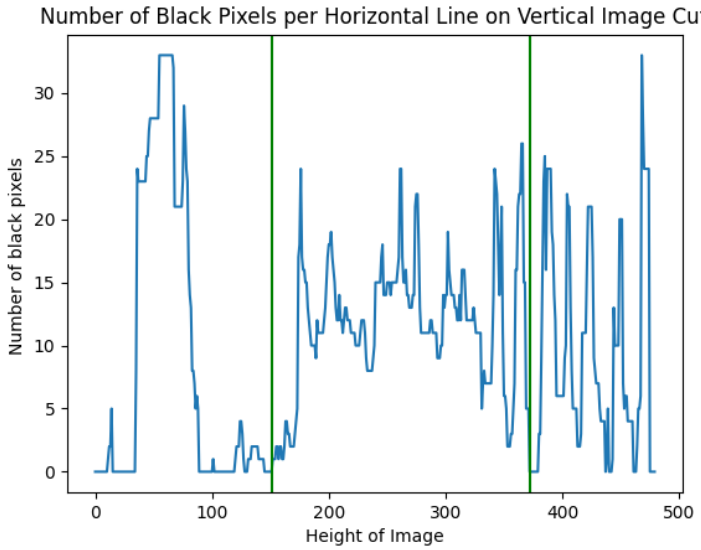
(c) Plot representing the vertical distribution of zeros.

Figure 4.2: Example of the process to find the vertical sides of potential trunks.

After finding the vertical sides of the potential trunks (Figure 4.2a and Figure 4.2), i.e. the spike pairs, the inverted binary image is separated for each pair (Figure 4.3a and Figure 4.4a). For each separation, cv2.reduce() is used again to condense the two-dimensional image into a one-dimensional array, but this time representing the horizontal distribution of zeros (Figure 4.3c and Figure 4.4c). This array is used to determine the top and bottom boundaries, but instead of spikes, it measures the furthest distance between two consecutive positions that have a value of zero (Figure 4.3b and Figure 4.4b).

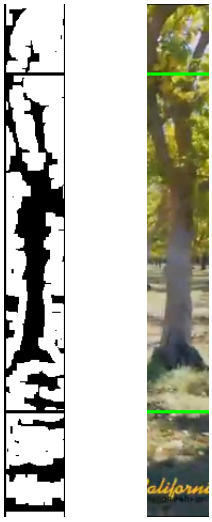
(a) Vertical section of the dilated image with horizontal lines.

(b) Vertical section of the original image with horizontal lines.



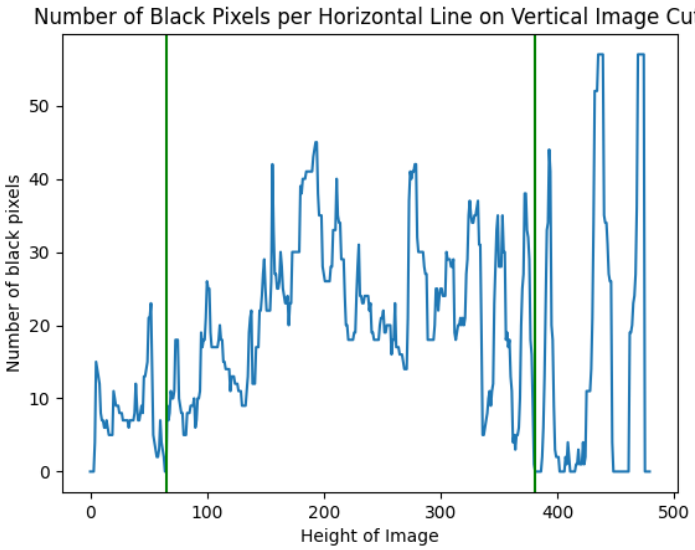
(c) Plot representing the horizontal distribution of zeros of the trunk.

Figure 4.3: First trunk identified.



(a) Vertical section of the dilated image with horizontal lines.

(b) Vertical section of the original image with horizontal lines.



(c) Plot representing the horizontal distribution of zeros of the trunk.

Figure 4.4: Second trunk identified.

Once the boundaries for each potential trunk have been established (Figure 4.5), each one is checked to ensure whether its height is at least two times greater than its width. If this condition is met, the region is considered as a potential trunk and its coordinates are appended to the list that the function returns.

Figure 4.5: Rectangles identifying trunks.

4.6 Clean Trees Function

This function is used to modify an image by removing the areas of the already detected trees. This is done by increasing the size of the bounding rectangles around the trunks and then setting the pixel values in these areas to a specific value (in this case, white as it can be seen in Figure 4.6 and Figure 4.7).

Figure 4.6: Example with the dilated image.

Figure 4.7: Example with the original image.

4.7 Increase Rectangle Size Function

As the function name suggests, this function is used to increase the dimension of the rectangle that identifies each trunk by 2% in height, 1% at the bottom and 1% at the top border, and by 60% in width, 30% on each side. The discrepancy in percentage increases between height and width can be attributed to the fact that the trunk has a greater tendency to shift laterally than vertically.

In the event that the boundary of the rectangle comes into contact or exceeds one of the image boundaries, it will be locked there. For example, if the top boundary touches or exceeds the top edge of the image, the y-coordinate of this boundary will remain at 0. The adjusted bounding box coordinates are stored in a new list, which is returned at the end of the function.

4.8 Set Trunk ID Function

This function returns the ID of a newly found trunk and the corresponding trunk coordinates by adding the ID to the beginning of the corresponding trunk coordinates list.

The ID is composed of an L or an R at the beginning, indicating whether the tree was on the left or right side of the image, respectively. This is followed by the frame number where it was initially found. Figure 4.1 illustrates two examples of trunks identified with a rectangle and their respective IDs. In this case, the one with the ID "L0__449" is on the left and was first detected on the frame 449, while the one with the ID "R0__402" is on the right and was first detected on the frame 402.

4.9 Follow Function

The follow function is designed to track and identify tree trunks across frames. This is achieved in four parts. The first is the procession of the frame with the implementation of specific algorithms that allow for easier tracking and identification. The second is tracking previously detected trunks, by attempting to follow and update the locations of trunks that were detected in previous frames, The third part is detecting new trunks, by identifying any new trunks that may appear in the current frame. The fourth is the final processing and output.

Processing the Frame

The function begins by processing the input frame to prepare it for trunk detection and tracking. This processing is crucial for enhancing the features in the image that correspond to tree trunks, making them possible to detect and follow across frames in a video. First the frame is passed through the Canny edge detection algorithm (cv2.Canny()), which detects edges by identifying areas with a significant intensity gradient, with a threshold1 value of 150 and a threshold2 value of 200. This step highlights the contours of the trunks

and foliage in their vicinity, while simultaneously obscuring the details on the surface of the trunks, making them more distinct against the background, resulting in a binary image like the one shown in the Figure 4.8. Subsequently, the function dilates the binary image through the use of the cv2.dilate() function, iterating it three times with a 3x3 kernel comprising solely of ones. Dilation is a morphological operation that expands the white regions in the image, which helps in connecting any gaps in the edges and clean any noise detected by the Canny algorithm (Figure 4.9).

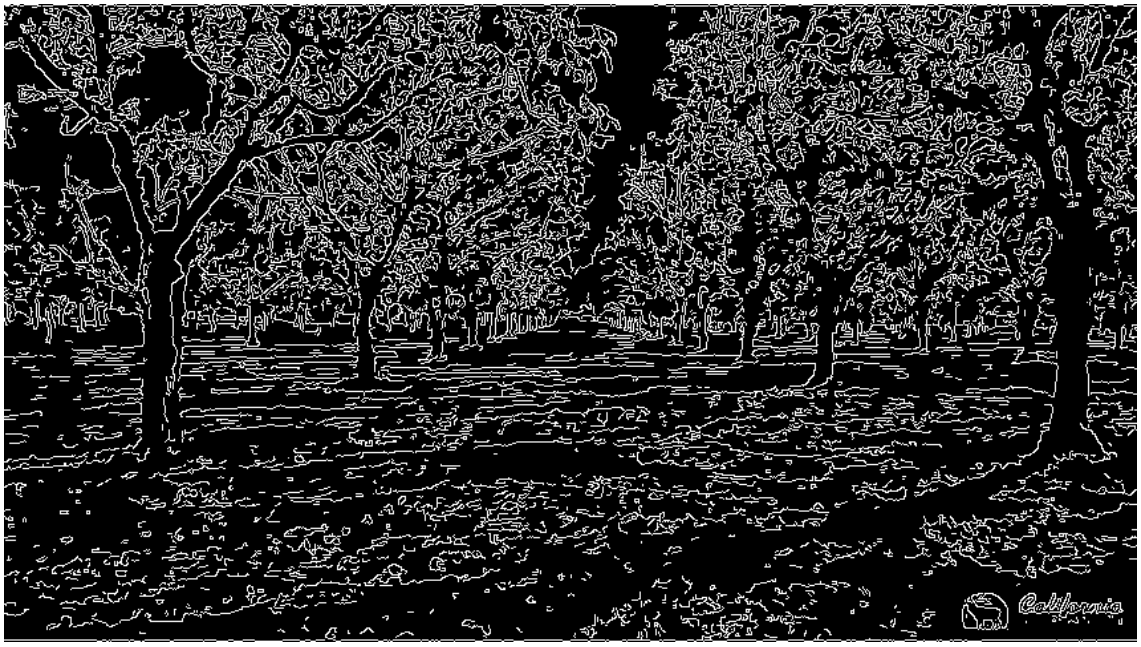


Figure 4.8: Example of Canny edge detection algorithm application.

Figure 4.9: Example of dilation function result after Canny function.

Tracking Previously Detected Trunks

Once the frame has been processed, any previously detected trunks are then used to determine their new location within the current frame. This is achieved by enlarging the bounding boxes around the previously detected trunks, with the use of the function explained in Section 4.7, in order to account for movement. The resulting enlarged area is then searched for the most prominent trunk within it (see Section 4.5).

In the event that a trunk is detected within this area, its coordinates are updated, and it is considered a continuation of the trunk from the previous frame.

In the event that no trunk is detected, the function makes slight adjustments to the original bounding box coordinates, assuming that the trunk might have moved slightly out of the previous area, and prepares to track it in subsequent frames. To this end, the position of the trunk within the frame is verified, determining whether it is on the left or right side. This information is used to determine the direction of movement, which is then applied to the original bounding box coordinates. The position of the trunk is adjusted by adding one pixel in the calculated direction and subtracting one pixel in the opposite direction. Additionally, one pixel is added to the top and bottom of the original bounding box, resulting in the final updated trunk bounding box.

Detecting New Trunks

Following the updating of the coordinates of previously detected trunks and the removal of the areas covered by these trunks from the image (see Section 4.6), the function then checks the rest of the image for any new trunks that may have become visible enough to be detected. These new trunks are identified using the function explained in Section 4.5, and it is ensured that the center of each detection falls within the correct interval, as previously explained in Section 4.11. Subsequently, the newly detected trunks are verified for overlap with the previously tracked detections using the function outlined in Section 4.11. Once verified, the new detections are assigned unique identifiers using the function described in Section 4.8.

Final Processing and Output

Finally, the function combines the list of updated trunks and newly detected trunks, processes them to ensure that they are not in contact with or extending beyond the boundaries of the image, and then returns a final list of trunks. This list is largely prepared to be used in the subsequent frame of the sequence, ensuring that any new trunks that enter the frame are tracked in the following frame along with those that have been followed from previous frames.

4.10 Buffering Begin Function

The buffering_begin function is designed to verify and certify that the detections in question are, in fact, genuine trunk detections and not false positives. In order to

achieve this objective and initiate the tracking process, it is necessary that ten subsequent detections be made, with a maximum buffer of three frames between each detection. This buffer is used to ensure that, even when there are changes in the environment or quick occlusions, it is given a chance for the subsequent frames to continue with the detection. In the event that no detections are made in the three buffer frames, the information for that detection is discarded.

Besides verifying if the position in the frame is the same and the frame count does not exceed by three, the requirement for a subsequent detection to be deemed valid is that the discrepancy along the x-axis between the current detection and the previous one does not exceed 2.5% of the frame width.

The function operates by first comparing the buffer list with the list of currently detected trunks. In the event that a trunk is identified solely within the buffer list, the buffer count associated with that detection is incremented by one. Upon reaching a value of three, the trunk is then removed from the list. Conversely, if a trunk is identified in both the buffer list and the current detection set, the buffer count is reset to zero and the number of detections left to be considered a trunk and commence being tracked, is reduced by one. Finally, in the event that any of the current detections fall outside the scope of the buffer list, they are added to it.

4.11 Middle Trunks Function

By preparing and ensuring that the UAV is programmed to fly at about half the height of the tree trunk, it is possible to discard some false detections, such as high branches, not detecting the entire trunk, or even misdetecting part of the foliage or sky. Since all of these examples would not have their vertical middle around the vertical middle of the trunks, this function was created to ignore them. Since not every tree trunk has the same vertical middle, it defines a movable vertical interval, that in this case is around the middle of the image, and then checks and returns the detections that have their vertical middle between the defined interval, which has been set to one ninth of the entire image. If the user does not specify a value for the interval, its position will be assumed to be in the center of the image.

In the Figure 4.10 it is possible to see the function in practice. The white lines inside the rectangles are the middle of each rectangle and those that do not have them between the blue horizontal lines are the ones that are ignored. In the three examples bellow the rectangles that are brown are the ones that are ignored.

(a) Example 1

(b) Example 2

(c) Example 3

Figure 4.10: Examples of the function process.

4.12 Compare Trunks Function

The compare_trunks function is used to compare two sets of detected tree trunks. One set where the ground around the trunks is cleaned and one, a newer set, where trees might be overgrown or obscured. The purpose of this function is to identify which trees have become obscured by overgrowth by analyzing changes in the size between the two images. The results are stored in a list that contains the percentage of the trunk that is obscured and the location for each tree.

The function begins by verifying if there are any new trunks detected in the original video frame, adding them to a list that tracks the overgrowth percentage for each trunk. Every new trunk starts with a value of minus one in the percentage slot, which will help to identify the trunks that are so obscured with vegetative growth that they cannot be detected in the new video.

To be able to compare the same trunk from different frames, it is essential to ensure that the frames in question have been successfully detected. To this end, it is necessary to ensure that the discrepancy between the x-axis and y-axis of the trunks from different frames is either equal or less than 5% of the frame width and height, respectively.

Once it has been established that the detections have been successful, the value of the coverage percentage is calculated between the two detections and the coverage percentage slot is inspected to verify the presence of a negative one. In the event that such a value is present, this indicates that it is the first time that the trunk is being compared, and thus,

the initial value of the coverage percentage is recorded.

If the coverage percentage slot does not present a value of a negative one, the equation 4.1 is applied to gather the median of the last 30 values.

$$median\_coverage\_perc = \frac{median\_coverage\_perc * 29 + coverage\_perc}{30} \tag{4.1}$$

This way the coverage percentage is updated in a manner that mitigates fluctuations over time.

4.13 Main Function

The provided function begins by opening two video files: the reference video, which represents a tree plantation with a clean field, and the overgrown video, which represents a state of overgrowth of the tree plantation field. These videos are then read frame by frame to detect and compare the trunks between each frame.

Initially, the frames are resized to a consistent size of 854x480 pixels to ensure uniform processing. Subsequently, the script employs the functions outlined in Section 4.9, Section 4.10 and Section 4.4 to detect and track tree trunks from each frame.

Once the trunks have been successfully detected and tracked, the script employs the compare_trunks function (see Section 4.12) to match trunks from the reference and overgrown videos. This allows the determination of whether the trunks are obscured by vegetation growth around the trunk, thereby indicating the necessity for cleaning.

Finally, the script checks if there were any trunks with a negative one representing the percentage of coverage. This value means that the respective trunk on the overgrown video was never detected, meaning that the vegetative growth covered the majority of the trunk, thereby necessitating the replacement of the negative one with 100. Subsequently, the trunks that have a coverage percentage greater than 15% are returned with the respective time stamp and side of the video where they were initially detected (for an illustration of this, see Figure 5.2).

5

Discussion

In this chapter, the results obtained from the script developed based on the methodology outlined in Chapter 3 are presented. As previously discussed in Section 3.1, the data utilized for the results was a video obtained from the online platform YouTube that was prepared for the test. This footage was employed as the reference video and used for the video exhibiting overgrowth covering the trunks. To mimic the overgrowth in the second video, the initial video was edited with the introduction of green blobs in front of multiple trees, partially obscuring their trunks.

5.1 Interpretation of Results

In order to evaluate the efficacy of the developed script, it was anticipated that 14 trunks would be discernible in the reference video, 4 unobstructed trunks in the overgrown video, and 10 partially obstructed trunks in the same overgrown video.

The results of the developed script, shown in Figures 5.1 and 5.2, demonstrate a promising ability to identify and track tree trunks in video footage, with a high degree of accuracy under controlled conditions. In the reference video, where the land around the trunks is cleared, the algorithm successfully identified thirteen out of fourteen trunks, indicating a strong capability for recognizing distinct trunks in a clear and less complex environment. This achievement suggests that the system is reliable when dealing with clearly defined trees, where trunks are not occluded partially or entirely by surrounding vegetation.

In the video exhibiting excessive vegetative growth, the system also accurately identified all four unobstructed trunks, thereby demonstrating once more its capacity to detect trunks that remain unobstructed by vegetation. As expected, since it is the same video, the result of the comparison between the trunks of the reference video and the trunks of the overgrowth video yielded a near zero percent coverage, with the biggest discrepancy being three percent.

However, the system exhibited a notable limitation in its ability to detect trunks with overgrowth obstructing their view, correctly identifying only three out of ten instances.

```
['R0__31', 100]
['L0__43', 2]
['L0__220', 47]
['R0__265', 100]
['R0__402', 1]
['L0__449', 100]
['R0__612', 100]
['L0__669', 100]
['L0__887', 39]
['R0__897', 3]
['R0__1041', 20]
['L0__1060', 2]
['R0__1122', 100]
```

Figure 5.1: ID and percentage of coverage of each trunk. The ID indicates the frame and the side of the frame at which the trunk was initially detected.

```
The tree that was first found at 0:01 on the right needs to be checked
The tree that was first found at 0:07 on the left needs to be checked
The tree that was first found at 0:08 on the right needs to be checked
The tree that was first found at 0:14 on the left needs to be checked
The tree that was first found at 0:20 on the right needs to be checked
The tree that was first found at 0:22 on the left needs to be checked
The tree that was first found at 0:29 on the left needs to be checked
The tree that was first found at 0:34 on the right needs to be checked
The tree that was first found at 0:37 on the right needs to be checked
```

Figure 5.2: Return of the script that specifies the time stamp and the side of the image in which each trunk that requires to be checked was found.

All of the trunks in question exhibited a degree of obstruction exceeding 15%, which was the threshold used to generate a warning indicating that the trunk needed to be checked. This indicates that the low detection rate is not a significant issue and does not significantly impact the overall results, although it still indicates a potential area for improvement.

Regardless, as shown on Figure 5.1, the six trunks not identified are represented with a 100, indicating that 100% of their trunk is covered, as no portion of the trunk could be discerned. In contrast, the three trunks detected showed the normal percentage of coverage, which was estimated to be 47%, 39% and 20%. The latter trunk can be clearly

identified and compared in Figures 5.3 and 5.4 with the ID of "R0__1041".

Figure 5.3: Frame from reference video, which shows three identified trunks.

Figure 5.4: Frame from overgrown video, which shows three identified trunks.

Furthermore, the system identified that nine out of the ten trunks required cleaning, correctly flagging almost every tree with significant overgrowth. However, since one tree was not identified in the reference video, it was not possible to verify whether its trunk was obstructed by overgrowth.

Despite the aforementioned mishap, the trunk detection algorithm was able to successfully identify and follow almost all tree trunks across multiple frames, thereby providing

reliable insights into which trees required maintenance. This demonstrates that the chosen methodology is effective for the specific task of trunk detection and tracking in near ground footage, thus achieving the core objective of the project. However, further refinements are necessary to improve performance in more complex environments, ensuring the system's robustness across a wider range of real-world conditions.

5.2 Limitations of the Study

While the developed methodology offers promising results in tree trunk detection, several limitations restrict its broader applicability. One of the primary limitations stems from the constraints placed on the system for accurate trunk tracking and comparison. The methodology requires that the tree trunks in the footage be relatively straight to be successfully detected and tracked. Curved or irregularly shaped trunks, which are common in natural forest environments, may not be effectively processed, reducing the system's ability to detect all of them accurately.

Another significant limitation is the requirement for both videos being compared to start at the same time. The methodology assumes that the UAV footage being analyzed has perfectly synchronized start times between the two video feeds. Any discrepancies in timing can lead to misalignment in the frames, resulting in errors in trunk tracking and comparisons. This makes the system less flexible and more difficult to deploy in real-world scenarios where exact synchronization of video recording may not always be possible.

Additionally, the speed at which the UAV records the video must be carefully controlled for optimal results. The system assumes a UAV speed of around 2 meters per second with 30 frames per second (to be determined based on the specific setup), ensuring that the frames captured have minimal motion blur and that the tree trunks are consistently detectable from frame to frame. If the UAV moves too quickly or too slowly, the quality of the footage may degrade, affecting the accuracy of trunk detection. This speed requirement limits the usability of the system in different flight conditions or for UAVs with varying operational speeds.

Finally, the most significant limitation of this study was the lack of data available for testing the script developed. Despite the fact that the footage is not homogeneous, with variations throughout, and the script developed was as general as possible, the data used was still limited. This could have resulted in overfitting and skewed results, indicating that while the script may demonstrate efficacy when applied to the specific data set utilized, it may encounter challenges when deployed in environments or conditions not represented in the footage employed.

6

Conclusions and Future Work

6.1 Conclusion

As part of a broader effort to improve forest fire prevention techniques through the application of advanced computer vision and image processing techniques, this document presents a comprehensive approach to the detection and tracking of tree trunks from tree plantations in video images, with the objective of evaluating the vegetation density below.

The study begins by understanding the state of the study area (Chapter 2), showing different methodologies employed to forest fire prevention (Section 2.4). Although there is research on the detection of vegetation density, the majority of it utilizes data collected above the tree canopy, which presents challenges in accurately identifying critical zones that require cleaning, particularly in areas near the trunks, where the need is most critical. Additionally, the majority of the techniques implemented utilize expensive advanced hardware, such as LiDAR technology, while the objective of this paper is to use normal RGB data, making the implementation of it a lot cheaper.

The methodology implemented (Chapter 3) was designed to handle the dynamic nuances of nature and the constant movement of UAVs, thereby ensuring that the system could robustly handle variations in lighting, distances, and trunk visibility. To achieve this, certain restrictions had to be imposed (see Section 3.3), which might result in the omission of some detections. However, as evidenced in Section 5.1, the majority of unobstructed trunks are correctly identified, as are some of the obstructed ones. This indicates that the imposed restrictions had a minimal negative impact on the overall outcome.

The implementation of the script revealed the crucial role of certain functions in ensuring the accuracy of trunk tracking over time, such as the "Follow" and "Buffering Begin" functions (see Sections 4.9 and 4.10). These functions allowed the system to maintain a reliable and continuous tracking of trunks, thereby improving the reliability of the detection process. The implementation of these functions, along with the others detailed in Chapter 4, demonstrated the feasibility and efficacy of the proposed methodology in real-world scenarios.

While the algorithms developed have shown promising results, the study also highlighted certain limitations (see Section 5.2), such as sensitivity to extreme environmental conditions and the lack of data available for further tests. Some of these optimizations are addressed in the subsequent section (Section 6.2), in which the potential future work with the methodology created in this paper is discussed.

In conclusion, this research has has made a notable contribution to the field of forest fire prevention by advancing the state of tree trunk detection and tracking in video data.

6.2 Future Work

This research opens up several avenues for future exploration and improvement, both in terms of enhancing the current system and expanding its applications. One potential enhancement is improving the system's ability to align video frames based on the geographical location of the trunks, rather than relying on both videos to start at the same time. By using GPS data or other location-based technologies, the system could compare frames from different videos based on the relative position of the trunks, even if the videos are not perfectly synchronized. This would increase flexibility and reduce the constraints on how the footage is collected.

Another important step forward would be enabling the system to work in real-time, requiring the embedding of the entire process within the UAV itself, without the need to rely on post-flight analysis. Real-time processing would significantly improve the system's practicality in forest fire prevention, as it could provide immediate feedback on changes in vegetation, allowing for quicker responses to overgrowth or other risks.

The methodology developed in this thesis also holds potential for broader applications beyond fire prevention. For instance, it could be employed in long-term ecological research to monitor the growth of trees over extended periods. By analyzing differences in trunk size and appearances, researchers could track the health and development of tree plantations. Additionally, this system could help identify dead or decaying trees, which are often a fire hazard. With more powerful processing capabilities and higher-resolution data, the system could even be adapted to detect signs of disease or fungal infections on trunks, providing critical insights for forest management.

Finally, further improvements could be made to enhance the robustness and efficiency of the system. Future work could involve integrating this trunk detection method with other forest monitoring technologies, such as environmental sensors or fire detection methodology. By combining different data sources, the system could provide a more complete view of forest conditions, improving its ability to detect risks and contribute to forest conservation efforts.

Bibliography

MacCarthy, J. et al. (2024-08). "The latest data confirms: Forest fires are getting worse". In: *World Resources Institute*. URL: https://www.wri.org/insights/global-trends-forest-fires (cit. on p. 1).

Westerling, A. L. (2016-06). "Increasing western US forest wildfire activity: Sensitivity to changes in the timing of spring". In: *Philosophical Transactions of the Royal Society B: Biological Sciences* 371, p. 20150178. DOI: 10.1098/rstb.2015.0178 (cit. on p. 1).

Moritz, M. A. et al. (2014-11). "Learning to coexist with wildfire". In: *Nature* 515 (7525). DOI: 10.1038/nature13946 (cit. on p. 1).

Jolly, W. (2015-07). "Climate-induced variations in global wildfire danger from 1979 to 2013". In: *Nature Communications* 6, p. 7537. DOI: 10.1038/ncomms8537 (cit. on p. 1).

Barbero, R. et al. (2015). "Climate change presents increased potential for very large fires in the contiguous United States". In: *International Journal of Wildland Fire* 24.7, p. 892. ISSN: 1049-8001. DOI: 10.1071/wf15083. URL: http://dx.doi.org/10.1071/WF15083 (cit. on p. 1).

Stephens, S. L. et al. (2013). "Managing Forests and Fire in Changing Climates". In: *Science* 342.6154, pp. 41–42. DOI: 10.1126/science.1240294 (cit. on p. 1).

Ascoli, D. et al. (2018). "Firebreak and Fuelbreak". In: *Encyclopedia of Wildfires and Wildland-Urban Interface (WUI) Fires*. Ed. by S. L. Manzello. Cham: Springer International Publishing, pp. 1–9. ISBN: 978-3-319-51727-8. DOI: 10.1007/978-3-319-51727-8_70-1. URL: https://doi.org/10.1007/978-3-319-51727-8_70-1 (cit. on pp. 1, 9, 10).

Agee, J. K. and C. N. Skinner (2005). "Basic principles of forest fuel reduction treatments". In: *Forest Ecology and Management* 211.1, pp. 83–96. ISSN: 0378-1127. DOI: https://doi.org/10.1016/j.foreco.2005.01.034 (cit. on pp. 1, 9, 10).

Green, L. R. (1977). *Fuelbreaks and other fuel modification for wildland fire control*. 499. US Government Printing Office (cit. on pp. 1, 9).

Agee, J. K. et al. (2000). "The use of shaded fuelbreaks in landscape fire management". In: *Forest Ecology and Management* 127.1, pp. 55–66. ISSN: 0378-1127. DOI: https://doi.org/10.1016/S0378-1127(99)00116-4. URL: https://www.sciencedirect.com/science/article/pii/S0378112799001164 (cit. on pp. 1, 9).

Fernandes, P. and H. Botelho (2003-07). "A review of prescribed burning effectiveness in fire hazard reduction". In: *International Journal of Wildland Fire* 12, pp. 127–128. DOI: 10.1071/WF02042 (cit. on pp. 1, 9, 10).

Lu, Y. et al. (2018). "A survey on vision-based uav navigation". In: *Geo-Spatial Information Science* 21 (1), pp. 21–32. DOI: 10.1080/10095020.2017.1420509 (cit. on p. 2).

Kanellakis, C. and G. Nikolakopoulos (2017). "Survey on computer vision for uavs: current developments and trends". In: *Journal of Intelligent & Robotic Systems* 87 (1), pp. 141–168. DOI: 10.1007/s10846-017-0483-z (cit. on p. 2).

Shakhatreh, H. et al. (2019). "Unmanned aerial vehicles (uavs): a survey on civil applications and key research challenges". In: *IEEE Access* 7, pp. 48572–48634. DOI: 10.1109/access.2019.2909530 (cit. on p. 2).

McCarthy, J. et al. (1955-01). "A proposal for the Dartmouth summer research project on artificial intelligence". In: *Artificial Intelligence: Critical Concepts* 2, pp. 44–53. URL: http://jmc.stanford.edu/articles/dartmouth/dartmouth.pdf (cit. on p. 3).

Moor, J. (2006-12). "The Dartmouth College Artificial Intelligence Conference: The Next Fifty Years". In: *AI Magazine* 27.4, p. 87. DOI: 10.1609/aimag.v27i4.1911. URL: https://ojs.aaai.org/aimagazine/index.php/aimagazine/article/view/1911 (cit. on p. 3).

Szeliski, R. (2011). *Computer Vision: Algorithms and Applications*. 1st ed. Texts in Computer Science. Springer (cit. on p. 3).

David A. Forsyth, J. P. (2002). *Computer Vision: A Modern Approach*. US ed. Prentice Hall (cit. on p. 3).

Bovik, A. C. (2000). *Handbook of Image and Video Processing*. 1st ed. Academic Press series in communications, networking and multimedia. Academic Press (cit. on p. 3).

— (2009). *The Essential Guide to Image Processing*. Academic Press (cit. on pp. 3, 4).

OpenCV (2023-07). *Optical Flow*. URL: https://docs.opencv.org/3.4/d4/dee/tutorial_optical_flow.html (visited on 2023-07-16) (cit. on p. 5).

Lucas, B. and T. Kanade (1981-04). "An Iterative Image Registration Technique with an Application to Stereo Vision (IJCAI)". In: vol. 81 (cit. on p. 5).

Horn, B. and B. Schunck (1981-08). "Determining Optical Flow". In: *Artificial Intelligence* 17, pp. 185–203. DOI: 10.1016/0004-3702(81)90024-2 (cit. on p. 6).

Illingworth, J. and J. Kittler (1988). "A survey of the hough transform". In: *Computer Vision, Graphics, and Image Processing* 44.1, pp. 87–116. DOI: https://doi.org/10.1016/S0734-189X(88)80033-1. URL: https://www.sciencedirect.com/science/article/pii/S0734189X88800331 (cit. on p. 6).

Burnham, J., J. Hardy, and K. Meadors (1995). "Comparison of the Roberts, Sobel, Robinson, Canny, and Hough Image Detection Algorithms". In: (cit. on pp. 6–8).

Ziou, D. and S. Tabbone (1998-01). "'Edge detection techniques: An overview'". In: *International Journal of Patttern Recognition and Image Analysis* 4, pp. 537–559 (cit. on p. 7).

Shrivakshan, G. and C. Chandrasekar (2012). "A comparison of various edge detection techniques used in image processing". In: *International Journal of Computer Science Issues (IJCSI)* 9.5, p. 269. DOI: 685e8ecc440343d59f157ca1add377b956b2199d (cit. on pp. 7, 8).

Canny, J. (1986). "A Computational Approach to Edge Detection". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* PAMI-8.6, pp. 679–698. DOI: 10.1109/TPAMI.1986.4767851 (cit. on p. 8).

Interior, U. D. of the (n.d.). *Recent Laws & Policies: The Office of Wildland Fire works to ensure consistent interpretation and implementation of new laws and policies across the Department of the Interior.* URL: https://www.doi.gov/wildlandfire/recent-laws-policies (visited on 2023-07-16) (cit. on p. 9).

Commission, E. (n.d.). *Forest fires*. URL: https://environment.ec.europa.eu/topics/forests/forest-fires_en (visited on 2023-07-16) (cit. on p. 9).

Fernández-Álvarez, M., J. Armesto, and J. Picos (2019-02). "LiDAR-Based Wildfire Prevention in WUI: The Automatic Detection, Measurement and Evaluation of Forest Fuels". In: *Forests* 10, p. 148. DOI: 10.3390/f10020148 (cit. on p. 11).

Dandois, J. P. and E. C. Ellis (2010). "Remote Sensing of Vegetation Structure Using Computer Vision". In: *Remote Sensing* 2.4, pp. 1157–1176. ISSN: 2072-4292. DOI: 10.3390/rs2041157. URL: https://www.mdpi.com/2072-4292/2/4/1157 (cit. on p. 11).

Yuan, C., Z. Liu, and Y. Zhang (2017). "Fire detection using infrared images for UAV-based forest fire surveillance". In: *2017 International Conference on Unmanned Aircraft Systems (ICUAS)*, pp. 567–572. DOI: 10.1109/ICUAS.2017.7991306 (cit. on pp. 11–13).

— (2016). "Vision-based forest fire detection in aerial images for firefighting using UAVs". In: *2016 International Conference on Unmanned Aircraft Systems (ICUAS)*, pp. 1200–1205. DOI: 10.1109/ICUAS.2016.7502546 (cit. on pp. 11, 13).

Koetz, B. et al. (2008). "Multi-source land cover classification for forest fire management based on imaging spectrometry and LiDAR data". In: *Forest Ecology and Management* 256.3, pp. 263–271. ISSN: 0378-1127. DOI: https://doi.org/10.1016/j.foreco.2008.04.025. URL: https://www.sciencedirect.com/science/article/pii/S0378112708003241 (cit. on p. 11).

Abdollahi, A. and M. Yebra (2023). "Forest fuel type classification: Review of remote sensing techniques, constraints and future trends". In: *Journal of Environmental Management* 342, p. 118315. ISSN: 0301-4797. DOI: https://doi.org/10.1016/j.jenvman.2023.118315. URL: https://www.sciencedirect.com/science/article/pii/S0301479723011039 (cit. on p. 11).

Apriani, Y., W. A. Oktaviani, and I. M. Sofian (2022). "Design and implementation of lora-based forest fire monitoring system". In: *Journal of Robotics and Control (JRC)* 3 (3), pp. 236–243. DOI: 10.18196/jrc.v3i3.14128 (cit. on p. 11).