João Pedro Leal Abalada de Matos Carvalho

Mestre em Engenharia Electrotécnica e de Computadores

Improved terrain type classification using UAV downwash dynamic texture effect

Dissertação para obtenção do Grau de Doutor em

Engenharia Electrotécnica e de Computadores

Orientador:	José Manuel Matos Ribeiro da Fonseca, Professor Associado com Agregação, FCT-UNL Universidade Nova de Lisboa
Co-orientador:	André Teixeira Bento Damas Mora, Professor Auxiliar, FCT-UNL Universidade Nova de Lisboa

FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

Outubro, 2020

Improved terrain type classification using UAV downwash dynamic texture effect

To my beloved family...

Acknowledgements

I would like to thank Professor José Fonseca for the supervision and assistance provided. I would also like to thank Professor André Mora for his support, commitment, advice and supervision, motivation and encouragement throughout the development of this thesis. Especially for all his help and patience when helping me sort out problems, even when sometimes things appeared to be stuck.

I would also like to thank to all my close friends and colleagues for many interesting discussions, their support and friendship.

Last, but not least, I thank my mother, father and sister for all their care, support and stimulus to do more and well, not only during this thesis, but also throughout the entire academic course. And to Beatriz, who was always present for the good and bad moments, for all her love, support, and especially, for the encouragement and never having stopped believing that I could make it through the end.

This thesis was supported by the IPSTERS project (DSAIPA/AI/0100/2018) and has also received funding from the ECSEL Joint Undertaking (JU) under grant agreement No 783119.

This thesis was developed in the CA3 group (CA3 - Computational Intelligence Research Group) of CTS, UNINOVA in cooperation with Beyond Vision from Lisbon.

Abstract

The ability to autonomously navigate in an unknown, dynamic environment, while at the same time classifying various terrain types, are significant challenges still faced by the computer vision research community. Addressing these problems is of great interest for the development of collaborative autonomous navigation robots. For example, an Unmanned Aerial Vehicle (UAV) can be used to determine a path, while an Unmanned Surface Vehicle (USV) follows that path to reach the target destination. For the UAV to be able to determine if a path is valid or not, it must be able to identify the type of terrain it is flying over. With the help of its rotor air flow (known as downwash effect), it becomes possible to extract advanced texture features, used for terrain type classification.

This dissertation presents a complete analysis on the extraction of static and dynamic texture features, proposing various algorithms and analyzing their pros and cons. A UAV equipped with a single RGB camera was used to capture images and a Multilayer Neural Network was used for the automatic classification of water and non-water-type terrains by means of the downwash effect created by the UAV rotors. The terrain type classification results are then merged into a georeferenced dynamic map, where it is possible to distinguish between water and non-water areas in real time.

To improve the algorithms' processing time, several sequential processes were converted into parallel processes and executed in the UAV onboard GPU with the CUDA framework achieving speedups up to 10x. A comparison between the processing time of these two processing modes, sequential in the CPU and parallel in the GPU, is also presented in this dissertation.

All the algorithms were developed using open-source libraries, and were analyzed and validated both via simulation and real environments. To evaluate the robustness of the proposed algorithms, the studied terrains were tested with and without the presence of the downwash effect. It was concluded that the classifier could be improved by performing combinations between static and dynamic features, achieving an accuracy higher than 99% in the classification of water and non-water terrain.

Keywords: Terrain Classification, Textures, Downwash, UAV, Feature Extraction, GPU, Neural Network, Autonomous Navigation

Resumo

Dotar equipamentos moveis da funcionalidade de navegação autónoma em ambientes desconhecidos e dinâmicos, ao mesmo tempo que, classificam terrenos do tipo água e não água, são desafios que se colocam atualmente a investigadores na área da visão computacional. As soluções para estes problemas são de grande interesse para a navegação autónoma e a colaboração entre robôs. Por exemplo, um veículo aéreo não tripulado (UAV) pode ser usado para determinar o caminho que um veículo terrestre não tripulado (USV) deve percorrer para alcançar o destino pretendido. Para o *UAV* conseguir determinar se o caminho é válido ou não, tem de ser capaz de identificar qual o tipo de terreno que está a sobrevoar. Com a ajuda do fluxo de ar gerado pelos motores (conhecido como efeito *downwash*), é possível extrair características de textura avançadas, que serão usadas para a classificação do tipo de terreno.

Esta dissertação apresenta uma análise completa sobre extração de texturas estáticas e dinâmicas, propondo diversos algoritmos e analisando os seus prós e contras. Um *UAV* equipado com uma única câmera RGB foi usado para capturar as imagens. Para classificar automaticamente terrenos do tipo água e não água foi usada uma rede neuronal multicamada e recorreu-se ao efeito de *downwash* criado pelos motores do *UAV*. Os resultados da classificação do tipo de terreno são depois colocados num mapa dinâmico georreferenciado, onde é possível distinguir, em tempo real, terrenos do tipo água e não água.

De forma a melhorar o tempo de processamento dos algoritmos desenvolvidos, vários processos sequenciais foram convertidos em processos paralelos e executados na GPU a bordo do *UAV*, com a ajuda da framework *CUDA*, tornando o algoritmo até 10x mais rápido. Também são apresentadas nesta dissertação comparações entre o tempo de processamento destes dois modos de processamento, sequencial na *CPU* e paralelo na *GPU*.

Todos os algoritmos foram desenvolvidos através de bibliotecas *open-source*, e foram analisados e validados, tanto através de ambientes de simulação como em ambientes reais. Para avaliar a robustez dos algoritmos propostos, os terrenos estudados foram testados com e sem a presença do efeito *downwash*. Concluiu-se que o classificador pode ser melhorado realizando combinações entre as características de textura estáticas e dinâmicas, alcançando uma precisão superior a 99% na classificação de terrenos do tipo água e não água.

Palavras-chave: Classificação de Terrenos, Texturas, Downwash, UAV, Extracção de Características, GPU, Redes Neuronais, Navegação Autónoma

Contents

List of Figures

List of Tables

Listings

Acronyms

AFarCloud	Aggregate Farming in the Cloud.
AI	Artifical Intelligence.
ALU	Arithmetic Logic Unit.
BEV	Beyond Vision.
CM	Circular Motion.
CNNs	Convolutional Neural Networks.
CPU	Central Processing Unit.
CU	Control Unit.
CUDA	Compute Unified Device Architecture.
DC	Density Calculation.
DCM	Dynamic Cortex Memory networks.
DL	Deep Learning.
EMD	Empirical Mode Decomposition.
FO	Flows Orientation.
FOV	Field Of View.
FPS	Frames Per Second.
GLCM	Gray-Level Co-Occurrence Matrix.
GLRLM	Gray-Level Run Length Matrix.
GPU	Graphics Processing Unit.
HEIFU	Hexa Exterior Intelligent Flying Unit.

HGM	Histogram of Gradient Magnitudes.
HOG	Histogram of Oriented Gradients.
HSV	Hue Saturation Value.
IMF	Intrinsic Mode Functions.
IPSTERS	IPSentinel Terrestrial Enhanced Recognition System.
LATP	Local Adaptive Ternary Patterns.
LBP	Local binary patterns.
LDA	Linear Discriminant Analysis.
LiDAR	Light Detection and Ranging.
LoG	Laplacian of Gaussian.
LTP	Local Ternary Patterns.
M5G	Mobilizadores 5G.
ML	Machine Learning.
MLP	Multilayer Perceptron.
NN	Neural Network.
NOF	Number of Features.
OF	Optical Flow.
PCA	Principal Components Analysis.
PDMFC	Projeto Desenvolvimento Manutenção Formação e Consultadoria.
PSO	Particle Swarm Optimization.
ReLU	Rectified Linear Unit.
RNN	Recurrent Neural Networks.
ROS	Robot Operating System.
SVM	Support vector machine.
TC	Terrain Classification.

TD	Travel Distance.
UAS	Unmanned Aircraft System.
UAV	Unmanned Aerial Vehicle.
URDF	Unified Robot Description File.
USV	Unmanned Surface Vehicle.
VPU	Visual Processing Unit.
W-K Filter	Wiener-Khinchin Filter.

Chapter 1

Introduction

1.1 The need for an Unmanned Aerial Vehicle

A Unmanned Aerial Vehicle (UAV) is simply defined as an aircraft without a human pilot aboard, whereas the concept of Unmanned Aircraft System (UAS), with which it is often confused, relates to the use of a ground control station (e.g. for performing heavy processing) and is able to communicate with the vehicle.

Nowadays, civil applications for UAVs are broadening (Figure 1.1) and, justified by the improvement of inherent technology (e.g. batteries and sensors), and a fall in the price of that technology, substantial resources are now being invested in research and development for UAVs' novel applications and technologies.

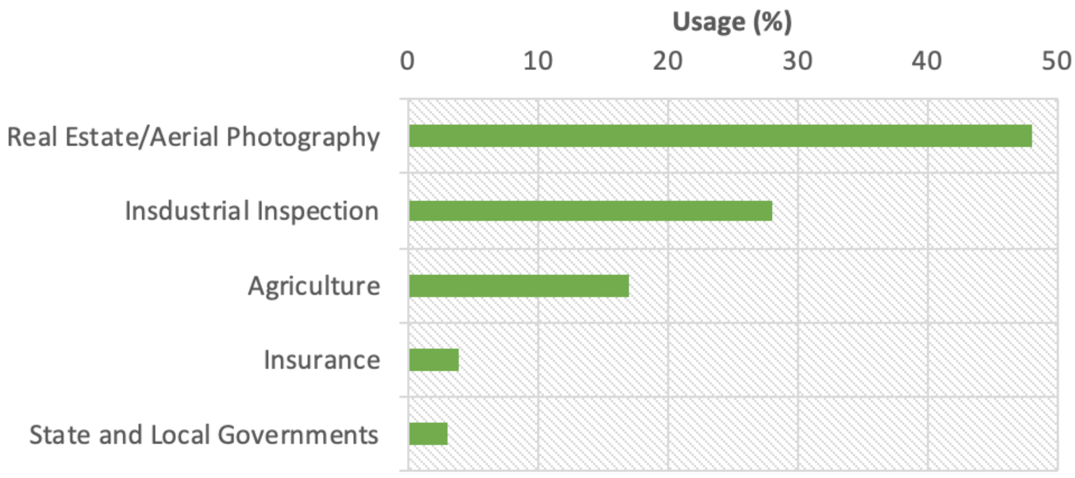


Figure 1.1: Usage of Small commercial UAV in different application areas (FAA, 2018).

As Figure 1.1 suggests, the Federal Aviation Administration (FAA) predicts that by

2021 there will be more than 6 million registered drones on their database, from which 21% are meant for construction, industrial and utility inspection, as explained in more detail in (Glaser, 2018).

1.2 Problem Statement and Motivation

For any robotic system to be autonomous, it needs to perceive the surrounding environment and distinguish terrains types and traversable from obstructed areas. As the title of this thesis suggests, a Terrain Classification (TC) will be implemented, or, in other words, different sensing modalities will be fused to make the UAV autonomous.

This is an important task given that in large unknown areas, the time, and the physical and monetary requirements necessary for a human to monitor all of the terrains, are considerable. Yet, UAVs can perform these same functions in a shorter time and are less prone to errors.

To achieve autonomy in a collaborative environment, an UAV can be useful to characterize the environment by identifying different types of terrains (water and non-water-types terrain) and allow other autonomous robots to avoid water (Unmanned Ground Vehicle (UGV)) or non-water terrains (USV). The environments where it can be flying over can be difficult to characterize and identify, being this terrain classification a challenging task. As an example, Figure 1.2 show two water-type terrains, being the lake easier to detected than open sea, that has more complex features (waves, foam, etc.).

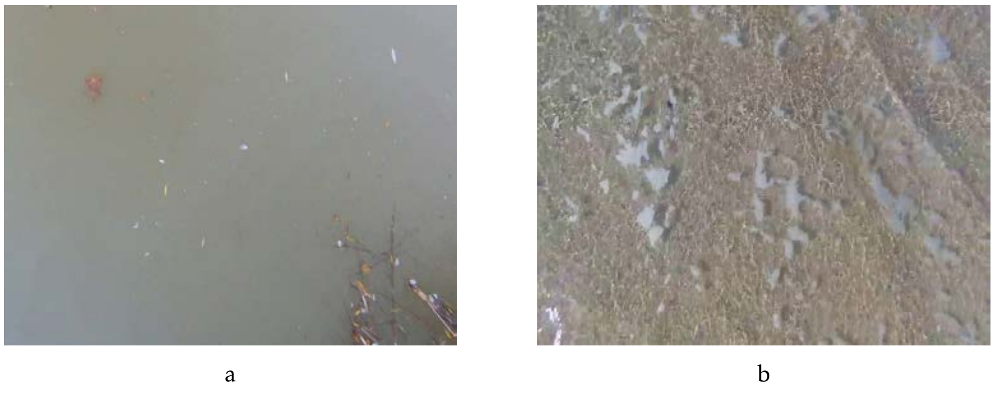

a b

Figure 1.2: Water terrains types (two of the testing environments): a) Parque da Paz-Water terrain without downwash; b) Costa da Caparica- Sea terrain type.

The terrain classification algorithms have to be fast, given the real-time nature of the tasks and the consequences arising from late decisions. For this reason, and with regard to the sensors being used, a passive (RGB camera) sensor is proposed to acquire 2D information from the environment where the UAV is flying over. Also, to process information in real-time on the UAV's on-board computer during navigation, a GPU co-processing is proposed to speed up calculations. The resulting data will be sent to a ground station to be visualized and shared among other robots. Figure 1.3 displays the process diagram.

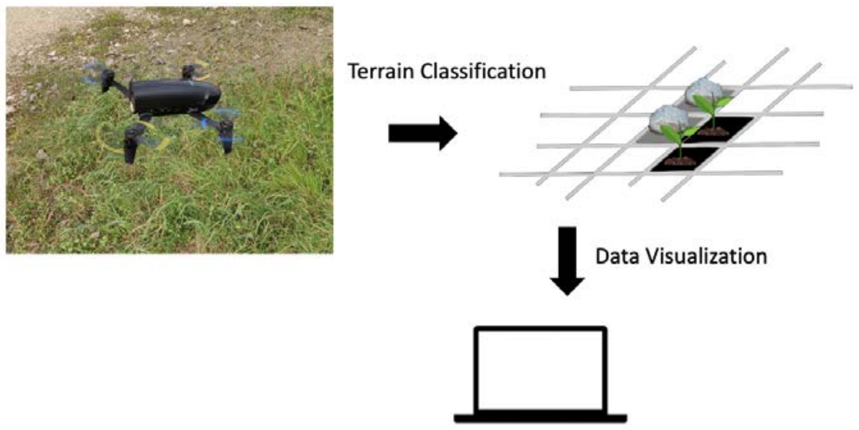


Figure 1.3: Diagram of the proposed algorithm.

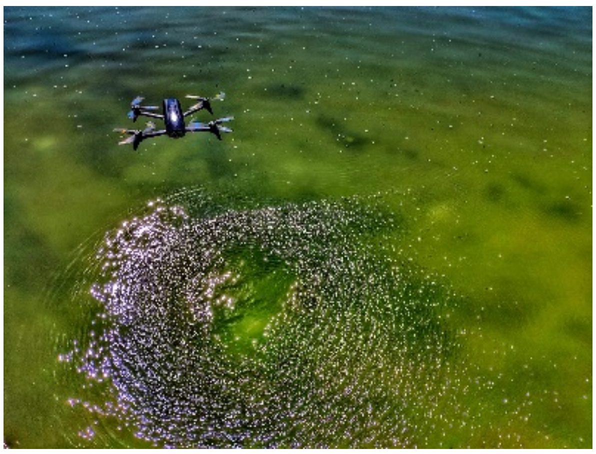
1.3 Research Question and Hypothesis

The ability to classify types of terrain is important for unmanned aerial vehicles (UAVs). There is a range of different areas where UAVs can benefit from this capability such as emergency landing, aerial mapping, decision making and cooperation between robots for autonomous navigation. This dissertation will only focus on identification of water versus non-water terrain types.

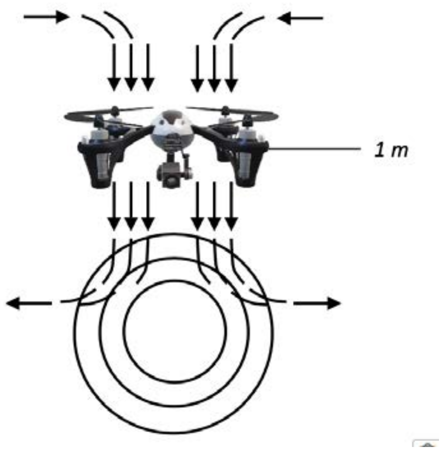
Previous works on terrain-type classification from RGB images taken onboard UAVs analyzed and classified terrain types from static images and, the few who extracted dynamic feature frame sequences of images do not robustly classify the types of terrains. Also, in water-type terrains, when there is no motion, i.e, no color variation, they are wrongly classified by algorithms designed to evaluate only static textures, as presented in Chapter 2. This happens in water-type terrains because it is not possible to observe any texture under these conditions. Lowering the UAV altitude to induce the downwash effect adds motion in the environment, and circular textures can be observed in water-type terrains.

Regarding terrain-type classification, only one article was found using the UAV downwash effect (Pombeiro et al., 2015) that appears at low altitudes (maximum 2 meters) to improve the classification results. The downwash effect is induced by the UAV rotors as seen in Figure 1.4. When the UAV approaches the terrain (especially in water), several types of object movements induced by the air flow are observed, such as linear and circular movements.

Having these challenges in mind, the proposed research question is as follows:



a



b

Figure 1.4: Downwash Effect: a) in water terrain; b) the concept.

Research Question

How can a UAV take advantage of the downwash effect and extract static and dynamic features to classify the type of the underlying terrain and improve robots cooperation?

To better analyse and interpret the main research question, five research sub questions are proposed:

1. How does the UAV downwash effect improve the classification of the type of terrain?
2. What is the effect of the downwash on different type of terrains?
3. Is it possible to induce dynamic textures with the downwash effect that are characteristic of some terrains (water terrain for example)?
4. With the downwash effect is it possible to classify water versus non-water terrain?
5. Can GPU computing improve algorithm-execution time?

In view of the aforesaid research question, the following hypothesis is proposed:

Hypothesis

If different types of terrain behave differently when exposed to the downwash effect of UAV rotors, then it should be possible to obtain unique information to identify them.

Guided by the research questions, a step-by-step process was designed dissertation. The first step is to analyse the different static and dynamic features that are observed in different terrain types in the presence of the downwash effect. Next, it is important to ascertain if it is possible to take some advantage of the textures to classify terrain type. Then, it is necessary to translate some CPU developed algorithms into GPU in order to improve the classification execution time. Lastly, it is important that the pros and cons of different feature selection methods for terrain classification are investigated and the most suitable method be adapted to the problem at hand.

1.4 Research Method

The proposed work has the goal of performing research in terrain-type classification to aid autonomous navigation and cooperation between robots. To achieve such a result, this dissertation followed the classical research method that consists of seven main phases, as can be seen in Figure 1.5.

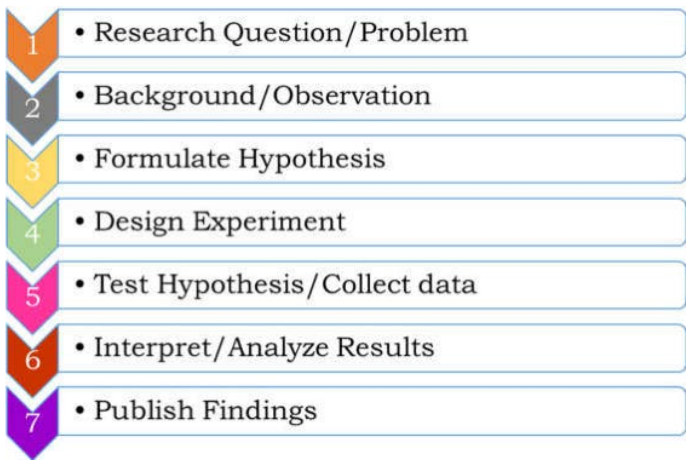


Figure 1.5: Classical research method adapted from (Camarinha-Matos, 2000).

Following this method, the research work was planned and scheduled according to the seven main phases:

1. **Research Question / Problem:** Working context identification and motivation in order to formulate the research question;

2. **Background / Observation:** State of the art analysis in research and practice. In these two topics, some main subjects are addressed, namely: related background in natural and artificial texture, texture extraction and selection methods;

3. **Formulate Hypothesis:** Formulate a hypothesis by conducting a preliminary analysis of some of the main problems, and the current state of the art;

4. **Design Experiment:** The hypothesis of this dissertation, will define different testing scenarios. Some may be tested in the lab using simulation or offline acquired data, but others certainly need live field experiments;

5. **Test Hypothesis / Collect Data:** At different levels, the implementation of all intermediate steps has to be tested and validates, and this it must be acertained whether the inclusion of those steps enhance the expected/hypothesized results. This includes ascertaining, via exhaustive and broad tests, whether all of the established variables (texture in this case) can classify a terrain (for example water, rock and vegetation), in order to validate the hypothesis and answer the research question;

6. **Interpret / Analyse Results:** After obtaining the results they must be analyzed. In order to decide if the results are good/acceptable, they need to be compared to the those from the real environment (If the program detects water, for example, this will be compared to the real environment and a person will check if there is actually water);

7. **Publish findings:** Whenever sufficient results with worthwhile contributory value to thei field of application are found, publication should follow the significance of new discoveries, to decide whether to submit the work to a conference or as a journal article.

Although the described phases might give the impression of a sequence, there are some iterations among them. As an example, after implementing, testing and interpreting some results, there was the need to make some reformulation in the hypothesis and corresponding model design to achieve results that were more accurate.

1.5 Integration with other Research Activities

This thesis was developed at CA3 - Computational Intelligence Group of CTS/UNINOVA and at PDM group Company. Consequently, it is important to mention some P2020 and H2020 projects that have already provided support for this work, such as Mobilizadores 5G (M5G), AFarCloud, PEST and IPSentinel Terrestrial Enhanced Recognition System (IPSTERS):

- **Mobilizadores 5G:** European Regional Development Fund (ERDF), through the Regional Operational Programme of Lisbon (POR LISBOA 2020) and the Competitiveness and Internationalization Operational Programme (COMPETE 2020) of the Portugal 2020 framework [Project 5G No. 024539 (POCI-01-0247-FEDER-024539)] - The implementation of the M5G project will allow the design and integrated validation of a new set of products, which will be able to form part of and provide services within future 5G networks. In the context of the M5G project, the UAV is know as a Patrol Drone and its function is to navigate to an event trigger (such as an explosion), requiring special capabilities to navigate in an unknown, unstructured and Beyond Line of Sight (BLOS) environment. The contribution of this dissertation to this project is in emergency landing, that is, when the UAV needs to land it is imperative that it land on a non-water type terrain;

- **AFarCloud:** The thesis is also related to the European Union's Horizon 2020 research and innovation programme (AFarCloud project) in particular the AFarCloud, that include as country partners: Austria, Belgium, Czech Republic, Finland, Germany, Greece, Italy, Latvia, Norway, Poland, Portugal, Spain and Sweden. The AFarCloud project will promote novel precision farming solutions by providing Cyber Physical Systems (CPS), as well as a monitoring and sensing framework able to utilize new autonomous robotics platforms and incorporating the legacy systems already deployed in the farms. One of the important tasks in this project is terrain-type classification to identify vegetation, sand and water terrains. Figure 1.6 shows a mission test in Valladolid, Spain;

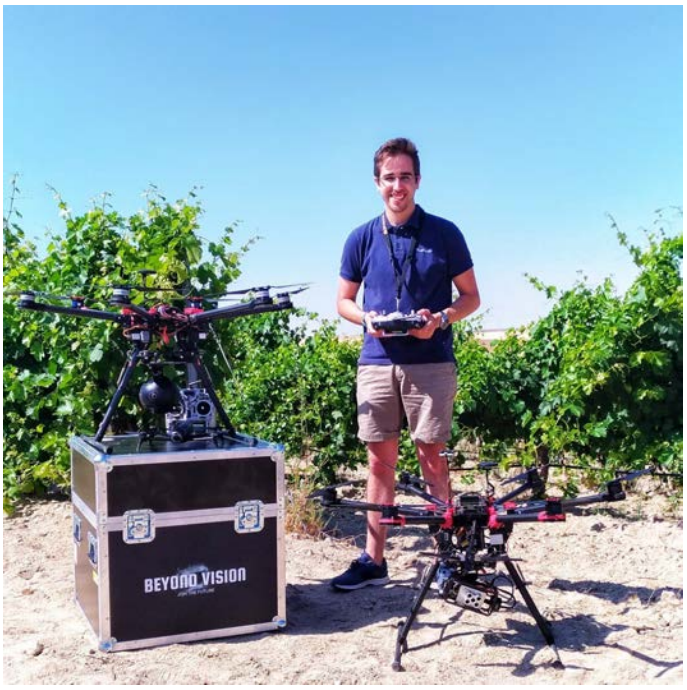


Figure 1.6: AFarCloud project. Tests in Valladolid, Spain.

- **PEST:** This work was also supported by the Portuguese Fundação para a Ciência e a Tecnologia (Science and Technology Foundation), within the framework of the PEST UID/EEA/00066/2019 project. Its purpose is to develop software for validating methodologies, reference architectures, components and suitable integration, as well as verification approaches for automated systems in different domains. These combine high security and privacy protection while preserving functional-safety and operational performance;

- **IPSTERS:** The aim of this project is to produce level-3 products for land applications using Sentinel-1 and Sentinel-2 datasets with Artifical Intelligence (AI) and dedicated hardware to accelerate data processing.

1.6 Dissertation Structure

The following chapters of this document present a study of static and dynamic textures extraction to classify different terrain types, in particular distinguishing water from non-water terrain to improve autonomous navigation between robots - starting from the state-of-the-art review through to the proposed approach and statement of the final conclusions. The chapters are structured as shown in the following summary:

- **Chapter 1: Introduction** presents the research question, its hypothesis, the gap regarding terrain classification and introduces an implementation approach. The motivations are outlined;

- **Chapter 2: State of the Art** shows the history behind the technology. Several interesting considerations are explored, in order to establish the background of existing terrain classification approaches. The search for new ideas and potential income;

- **Chapter 3: Methodology** shows the terrains under study in this dissertation, the chosen UAVs and how the RGB camera is calibrated; This chapter also analyses and build algorithms to classify terrains using textural features extracted from the downwash effect captured by an RGB camera;

- **Chapter 4: Experimental Results** presents all the results acquired from the algorithms in study and provides a deep analyses and comparison between the solutions presented;

- **Chapter 5: Conclusion and Future Work** summarizes the study and its achievements. Further comments, criticisms and improvements are taken into consideration, so the project can evolve and progress;

- **Apendix A: Dissemination** shows the published journal and conference papers that validate the algorithms proposed in this dissertation;

- **Apendix B: Supporting Concepts** introduces the different UAVs' types; The concept of downwash effect; the Robot Operating System (ROS) framework - How the UAV will comunicate with the ground station.

Chapter 2

State of the Art

Nowadays, due to UAVs' greater availability and capabilities, there is a research trend to explore innovative applications of UAVs useful to society. UAVs are having a major impact on search and rescue missions, in logistics, in precision agriculture, among other applications. Key issues are the provision of safe and reliable operation and offer a clear perception of the surrounding area.

The challenge of determining what is around a robot and distinguishing traversable passages from obstructions is a topic of research in itself, given its importance for achieving autonomous behaviour. The aim of this chapter is to present and review the literature related to this dissertation's topics of interest, namely terrain classification.

Figure 2.1 expresses the motivation behind this dissertation: cooperation between robots (in this case a UAV with a USV). The task of this UAV is to cooperate with the USV so that it reaches its intended destination. Since the LiDAR of the USV has a range between 250 and 300 meters, when the intended destination is further than this range the LiDAR cannot detect obstacles and thus it is impossible to ascertain the best path to reach the destination. Thus, instead of the USV walking along unknown paths and consuming resources, the UAV is launched with the purpose of detecting terrain types and mapping them, and then that information is delivered to the USV in order to supplement the trajectory planning of the USV to the required destination. As such, terrain classification is a crucial functionality for a wide range of autonomous vehicles (Matos-Carvalho et al., 2018):

- Ground vehicles to avoid water bodies;
- Aerial vehicles to determine suitable landing areas;
- Surface vehicles to detect safe passageways.

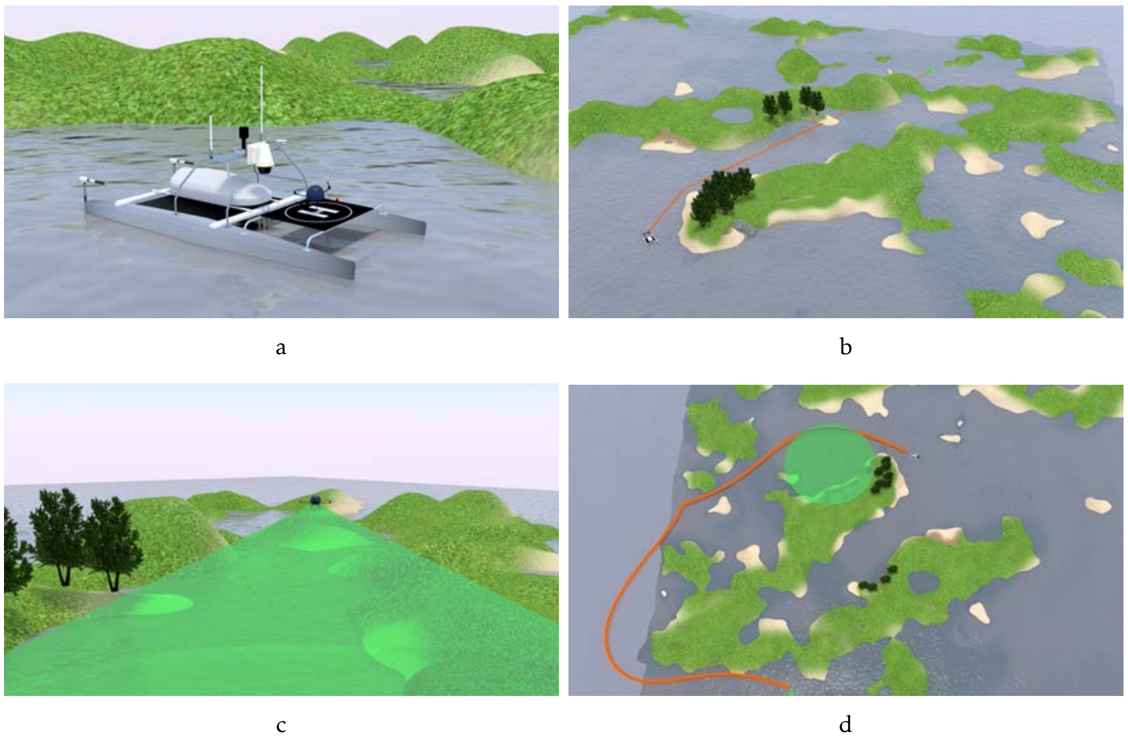


Figure 2.1: a) Cooperation between an Unmanned Aerial Vehicle (UAV) and Unmanned Surface Vehicle (USV) to improve autonomous navigation. b) shows a route followed by a USV alone. c) and d) show a UAV cooperating in order to improve the route taken by the USV (Matos-Carvalho et al., 2018).

Thus, to extract the terrain features to be used to classify the terrain types, the following sections will present several methods to be used in this context using 2D image processing.

2.1 Texture

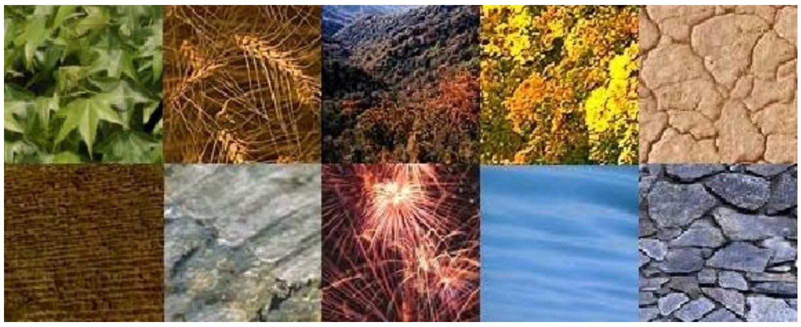
"Texture is one of the important characteristics used in identifying objects or regions of interest in an image, whether the image be a photomicrograph, an aerial photograph, or a satellite image" – Robert M. Haralick (Haralick et al., 1973).

Texture is the appearance of a surface, i.e. the "skin" of a shape, which allows one to identify it and distinguish it in other ways. When we touch or look at an object or surface it is possible to feel whether the texture/skin is smooth or soft - a small difference between high and low features per area; rough or wavy - a large difference between high and low features per area. The texture is therefore a visual or tactile sensation.

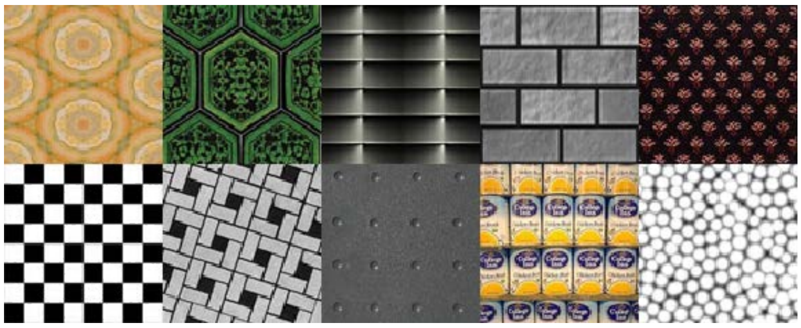
As for the visual aspect it is possible to group the textures as follows:

- **Natural Textures** → Those that result from the natural intervention of the environment or that characterize the external appearance of the forms and things that exist in nature, i.e. rocks, vegetation and wood (Figure 2.2 a));

- **Artificial Textures** → Those that result from human intervention through the use of manipulated materials and instruments(Figure 2.2 b)).



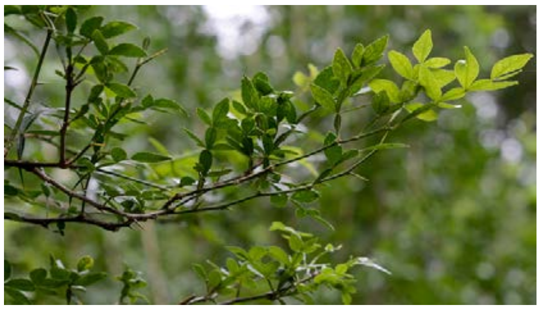
a



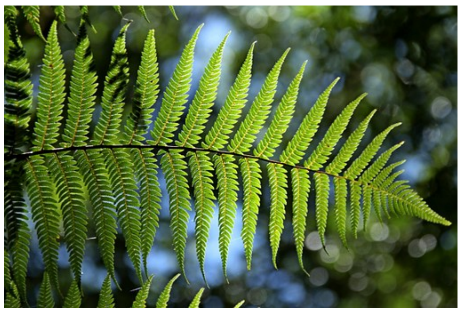
b

Figure 2.2: Natural Textures a) vs Artifical Textures b).

As shown in Figure 2.2 a) and b), textures are created in natural or man-made settings by a vast variety of objects, and can be seen in images. However, even in natural settings, objects of the same type (as observed in Figure 2.2 a)) exhibit differences from one another. Thus, textures cannot be described solely by the object type but instead require more detailed evaluation.



a



b

Figure 2.3: Different leaf types a) and b).

As seen in Figure 2.3, the same object classification (in this case two types of leaf) shows different sizes, shapes and arrangements. Whereas the first is small in size with an enormous amount of leaves, the second, despite having larger leaves, contains a smaller amount of leaves. Consequently, it is necessary to find a method that can identify any leaf type, regardless of its texture, color and size. However, it is very hard to explain the spatial arrangements of the leaves because of irregular arrangement. However, by analysing each figure (Figure 2.3 a) and b)), it is possible to identify the way in which the leaves are arranged. This is simply one of the problems of identifying and describing what texture is. Thus, there are two main approaches to identifying and describing what a texture is:

- **Structured Approach:** Texture is a set of primitive *texels* with a regular or repeated pattern (relationship). An example of the structured approach is the Voronoi tesselation. Using this method, it is possible to obtain the characterization of the spatial relationships. Section 2.2 will present this topic in detail and how these spatial relationships can be described;

- **Statistical Approach:** This approach sees an image texture as a quantitative measurement of the spatial distribution of intensities in a certain region. This approach is, in general, easier to use in computer vision as will be shown below.

To summarize, and in order to understand the various texture characteristics, this chapter will describe how texture can be represented, computed, and how it can be used in image analysis (computationally).

2.2 Texel-Based Texture Descriptions

As mentioned in section 2.1, texture can be a set of primitive *texels* with spatial relationships, as shown in Figures 2.4 a) and b), and a structured approach can describe these spatial relationships. One well-known method that follows this approach was developed by Tuceryan and Jain (Tüceryan and Jain, 1990). They used the Voronoi tesselation properties to extract the textural information from a given image. This method was also proposed to provide an understanding of the relationship between the local spatial distributions of sites and the Voronoi polygon forms (Voronoi, 1908). These sites are extracted by a given image and represent the texture extraction by using, for example, the local maximum intensity or another simple/complex threshold method.

The first Voronoi tesselation proposed in computer vision, was developed by Ahuja (Ahuja, 1982). Let ζ be three or more sites in an Euclidean plane, which to simplify, are not collinear. Consider also a random pair of points P and Q that belong to the ζ plane. For any pair of points P and Q, it is possible to create the perpendicular bisector of the line joining these points (in this case, P and Q are called Voronoi neighbors because they share the same bisector line). The points of these lines are the equidistant points between P and Q that divide the plane into two half planes: the points that are closer to P- H^Q in the

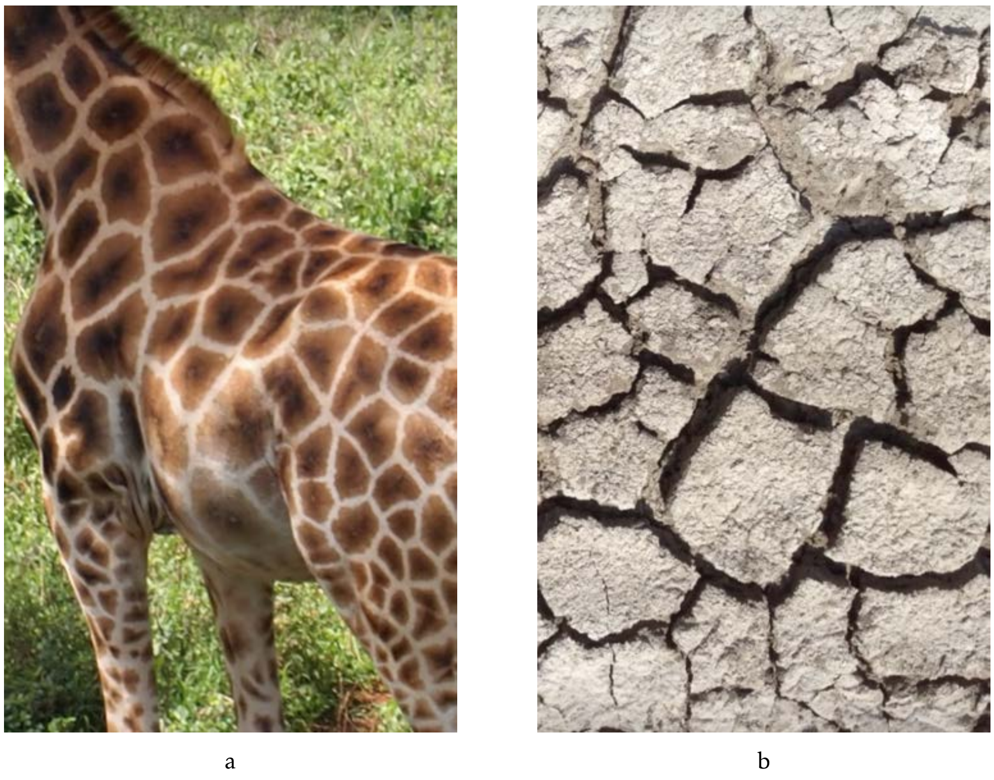


Figure 2.4: Patterns based on the natural world: From the spots on a giraffe a) to the spots on dried mud b).

Euclidean plane; and the points that are close to Q- H^P (in relation to the bisector line between P and Q).

$$V(P) = \bigcap_{Q \in \zeta, Q \neq P} H^Q \tag{2.1}$$

Equation 2.1 denominated by polygonal region (Voronoi, 1908), is the calculation of all points that are closer to P. The calculation of all polygonal region in ζ is called the *Voronoi diagram* of ζ (Shamos and Hoey, 1975) and can be measured using Euclidean or Manhattan distances:

$$Euclidean\ distance = d\Big[Point\ A\ , Point\ B)\Big] = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2} \tag{2.2}$$

or

$$Manhattan\ distance = d\Big[Point\ A\ , Point\ B\Big] = |x_1 - x_2| + |y_1 - y_2| \tag{2.3}$$

where

$$Point\ A = (x_1, y_1)\ and\ Point\ B = (x_2, y_2) \tag{2.4}$$

Finally, Voronoi diagrams with incomplete polygonal regions are called *Voronoi tessellation* of the Euclidean plane. *Voronoi tessellation* can be seen in Figure 2.5.

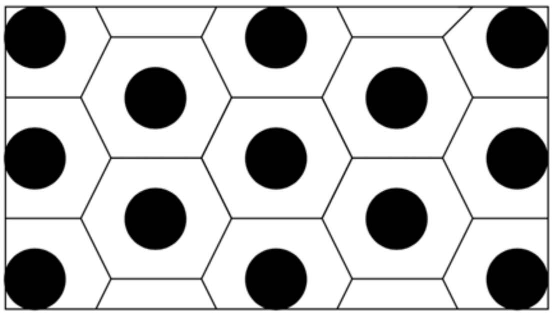


Figure 2.5: The Voronoi tesselation of a set of sites texels.

After the extraction of all polygons and the creation of *Voronoi tessellation* as shown in Figure 2.5, it is possible to merge the polygons into clusters with similar shape feature values in order to identify the borders separating such uniform regions. The following equation is used for this purpose (Hu, 1962).

$$m_{pq} = \sum_x \sum_y (x - x_0)^p \, (y - y_0)^q \, I(x,y) \tag{2.5}$$

Equation 2.5 represents the moments of the Voronoi polygon area and can provide information about two important matters:

- Polygon shapes in the entire image;
- Spatial distribution in the textured image.

Detailed information about the use of Equation 2.5 in polygon regions is explained in (B.WilsonJr. and S.Farrior, 1976). The lower of p and q values (order moments) will provide more accuration information in geometric interpretations. For example, momentum m_{00} is responsible for providing information about the polygon area because it will be the sum of all information about the *Voronoi tessellation*. The information about m_{10}/m_{00} and m_{01}/m_{00} is related to the displacement between the site (polygon centroid) in relation to (x,y) points in the x and y directions. The m_{11}, m_{02} and m_{20} give information about the major axis orientation. With this information, it is possible to obtain some good features to build clusters over the image as shown in Table 2.1. Where f_1 provides information about the polygon area; f_2 details the distance between point (x,y) and the centroid coordinates; f_3 provides the polygon direction; f_4 is related to the polygon elongation; f_5 provides information about the polygon orientation.

2.3 Quantitative Texture Measurements

As explained in section 2.2, the *Voronoi tessellation* algorithm has good accuracy when applied in environments with artificial textures, due to their well defined spatial distributions, as shown in Figure 2.2 b). However, in real environments, i.e. with natural textures,

Table 2.1: The computation of texture features from moments.

Feature	**Computation**
f_1	m_{00}
f_2	$\sqrt{\bar{x}^2+\bar{y}^2}$
f_3	$\tan^{-1}(\bar{y}/\bar{x})$
f_4	$\left[\frac{\left[(m_{20}-m_{02})^2+4m_{11}^2\right]^{\frac{1}{2}}}{\left[(m_{20}-m_{02})^2+4m_{11}^2\right]^{\frac{1}{2}}+m_{20}+m_{02}}\right]^{\frac{1}{2}}$
f_5	$\tan^{-1}(2m_{11}/(m_{20}-m_{02}))$

it is more difficult to use the *Voronoi tessellation* method to extract and evaluate textures. Thus, it is necessary to use a statistical approach, as mentioned in section 2.1 with the goal of evaluating quantitatively the textures that are under study. This approach is less intuitive, but in terms of computational efficiency it is faster and with good precision in the extraction and classification of textures.

In the following sections several algorithms of quantitative texture extraction will be examined and explained.

2.3.1 Edge Density and Detection

There are several ways to analyse textures, as previously mentioned. One of the well-known methods, which is simple to implement, is known as Edge detection. With the ability to check the busyness of a particular region using a given image, it is possible to assist in texture identification analysis. For this, it is necessary to know what gradient and direction means:

- **Gradient:** Created by the original image, gradient is the directional change in the image's intensity value;
- **Direction:** Is where the gradient is pointing, i.e. in which direction the edge is to be found (for instance- if the angle is zero it means that the change of intensity values will flow from the left (low values) to the right (high values))).

The following equation (equation 2.6) demonstrates how to calculate the image gradient, know as gradient magnitude, Mag(p):

$$\nabla f = \begin{bmatrix} g_x \\ g_y \end{bmatrix} = \begin{bmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{bmatrix} \tag{2.6}$$

Where $\frac{\partial f}{\partial x}$ and $\frac{\partial f}{\partial y}$ are the gradient x and y directions, respectively.

However, to simplify the calculations, the image derivative can be replaced by finite differences. To produce better results, the central difference is used instead of forward and backward differences, as shown in Figure 2.6.

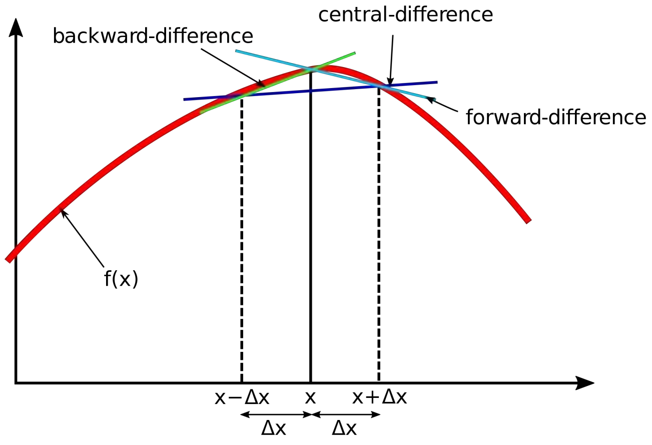


Figure 2.6: The three finite differences: forward, backward and central differences. The central difference gives better results to approximate a derivative.

Thus, to perform the central difference, a convolution between the mask and the image, T, is used:

$$G_x = \frac{\partial f}{\partial x} = \begin{bmatrix} -1 & 0 & +1 \end{bmatrix} * T \qquad and \qquad G_y = \frac{\partial f}{\partial y} = \begin{bmatrix} -1 \\ 0 \\ +1 \end{bmatrix} * T \tag{2.7}$$

Note that equation 2.7 uses an odd mask (e.g. 3x3 matrix) to avoid pixel shifting and non-symmetric filter response.

The next step is to calculate the two outputs from each pixel P of the entire image T:

$$Gradient\ magnitude = Mag(P) = \sqrt{G_x^2 + G_y^2} \tag{2.8}$$

and,

$$Gradient\ direction = Dir(P) = \tan^{-1}\left(\frac{G_y}{G_x}\right) \tag{2.9}$$

With this information (equations 2.8 and 2.9), it is possible to use histograms in order to suppress the image noise. Taking Figure 2.7 as an example, a mask to detect edges will be used in the next step. The measurement of edgeness is usually computed by simple edge masks such as the Sobel, Prewitt, Roberts, Differentiation, Canny and Laplacian:

- **Sobel:** Being a filter that is mainly applied in contour detection algorithms, the Sobel filter calculates the gradient of the image intensity at each pixel and its direction.

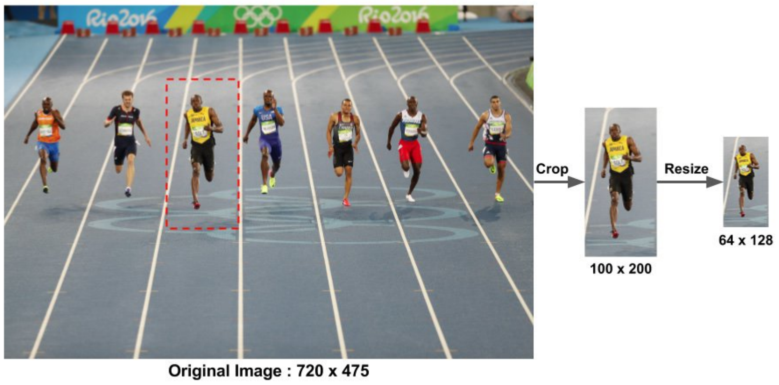


Figure 2.7: Example of an image to explain the edge detection method as distinct from the magnitude and direction gradient to histogram calculation (Mallick, 2016).

With this, and as mentioned before, it is possible to estimate the presence of a high transition and the orientation of this transition. As those high variations correspond to well-defined boundaries between objects, it is possible to detect contours. Instead of using the kernel in equation 2.7, Sobel uses the following:

$$G_x = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix} * T \quad and \quad G_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ +1 & +2 & +1 \end{bmatrix} * T \tag{2.10}$$

- **Prewitt:** This filter, although similar to the Sobel filter as mentioned above, manages to suppress the high frequency noise due to the mask shown in equation 2.11.

$$G_x = \begin{bmatrix} -1 & 0 & +1 \\ -1 & 0 & +1 \\ -1 & 0 & +1 \end{bmatrix} * T \quad and \quad G_y = \begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ +1 & +1 & +1 \end{bmatrix} * T \tag{2.11}$$

Thus, the edge results are not as pronounced in Figure 2.8, which shows the difference between the Sobel and Prewitt filters (Jose et al., 2014).

- **Roberts:** According to this filter, the edge detector should produce the corners captured and shown with good definition. The edge intensity values should be near what a human can perceive and the background should only contribute without noise values. Knowing this, the masks that Roberts proposed to find the edges are describe in equation 2.12:

$$G_x = \begin{bmatrix} +1 & 0 \\ 0 & -1 \end{bmatrix} * T \quad and \quad G_y = \begin{bmatrix} 0 & +1 \\ -1 & 0 \end{bmatrix} * T \tag{2.12}$$

By making a convolution between these masks and the original image T, it is possible to calculate the gradient magnitude and direction as shown in equations 2.8 and 2.9, respectively.

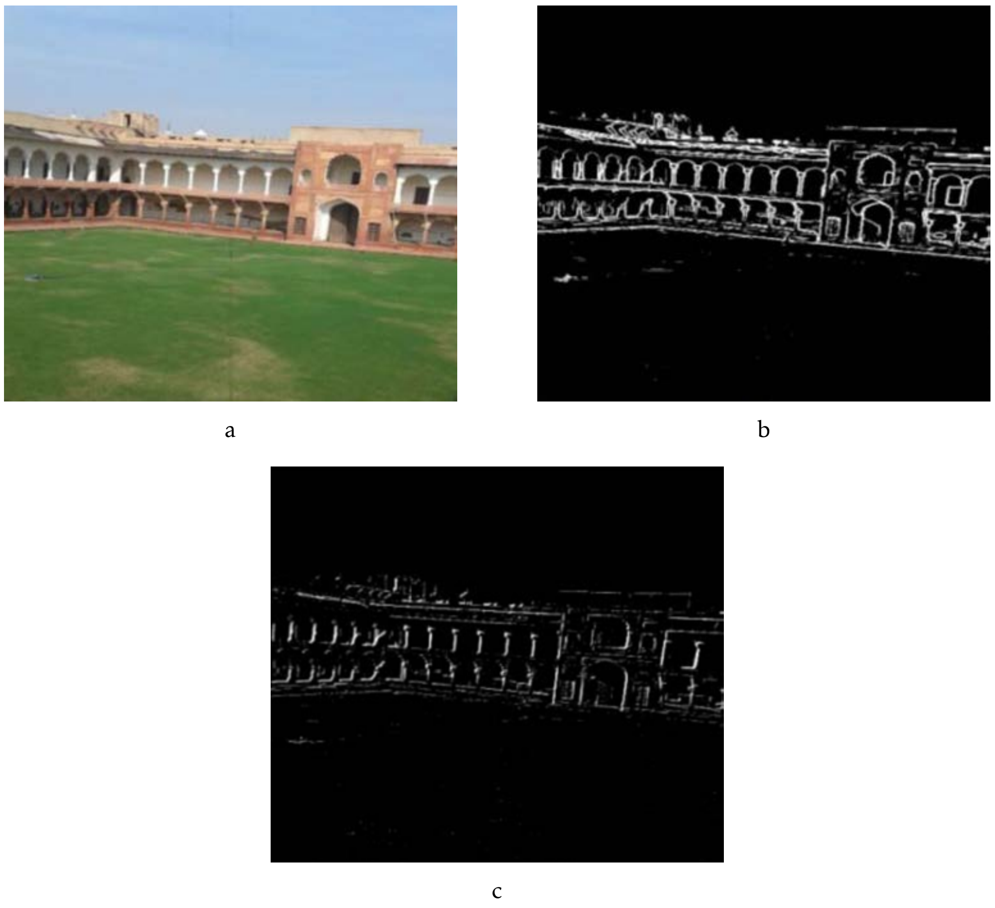


Figure 2.8: Comparison of edge detectors: input image (a); Sobel result (b); and Prewitt result (c) (Jose et al., 2014).

It should be noted that the Roberts filter, although it is simple mathematically and thus computationally, has a huge disadvantage when it comes to noise. If there is noise in the background, the Roberts filter will not be able to suppress it and will produce false edges.

- **Differentiation:** Being a two-dimensional filter also, the differentiation filter closely resembles Roberts. However, instead of being the sum of the differences between the diagonals, it is the difference in both the horizontal and vertical relative to the point where the calculation is being carried out:

$$G_x = \begin{bmatrix} +1 & -1 \\ 0 & 0 \end{bmatrix} * T \qquad and \qquad G_y = \begin{bmatrix} +1 & 0 \\ -1 & 0 \end{bmatrix} * T \tag{2.13}$$

With the x and y convolution masks (equations 2.13, 2.8 and 2.9) the magnitude of the gradient and its direction can be calculated. However, for the magnitude, instead of using the square root as described in equation 2.8, the absolute value is used to calculate the gradient magnitude, as show in equation 2.14.

$$Gradient\ magnitude\ alternative = Mag(P) = |G_x^2 + G_y^2| \tag{2.14}$$

The biggest advantage of using the absolute value, is to decrease the processing time. However, it becomes a disadvantage if the goal is to get more reliable results.

- **Canny:** Developed in (Canny, 1986), this algorithm is based on the Sobel filter previously presented but improved in order to respect three conditions:

  1. The edges have to be detected with good accuracy. In addition, this algorithm should be able to detect as many edges as possible;
  2. The edges detected by Canny algorithm should be located as close as possible to the real edges, i.e., it is necessary to minimize this distance;
  3. The noise, in the object under study and in the background, should be minimized in order to not create false edges.

  With these ideas in mind, and as mentioned before, to suppress the noise, a Gaussian filter is used in a Canny algorithm as shown in equation 2.15:

$$Gaussian\ Filter_{i,j} = \frac{1}{2\pi\sigma^2}\exp\left(-\frac{(i-(k+1))^2+(j-(k+1))^2}{2\sigma^2}\right) \tag{2.15}$$

  Where k is the Gaussian kernel size, i and j are the Gaussian kernel row and column, respectively. Rows and columns also need to be in interval between 1 and $(2k+1)$ (Canny, 1986).

  Usually, the Gaussian kernel size is a 5x5 filter due to two fundamental reasons: The larger the kernel size, the more noise will be suppressed (because the image will be smoother). However, the larger the size of this kernel, the fewer edges are detected by the Canny algorithm for the reason already mentioned: smoothness. Due to these two factors, it is necessary to find a middle ground.

  An example of a 5x5 filter with $\sigma = 1.4$ is shown in Figure 2.9 and in equation 2.16.

$$Gaussian\ Filter = \begin{bmatrix} +2 & +4 & +5 & +4 & +2 \\ +4 & +9 & +12 & +9 & +4 \\ +5 & +12 & +15 & +12 & +5 \\ +4 & +9 & +12 & +9 & +4 \\ +2 & +4 & +5 & +4 & +2 \end{bmatrix} \cdot \frac{1}{159} \tag{2.16}$$

  Equation 2.16 will now convolute with the image T. Next, in order to calculate the gradient magnitude and its direction, the Sobel filter (equation 2.10) is used as mentioned before.

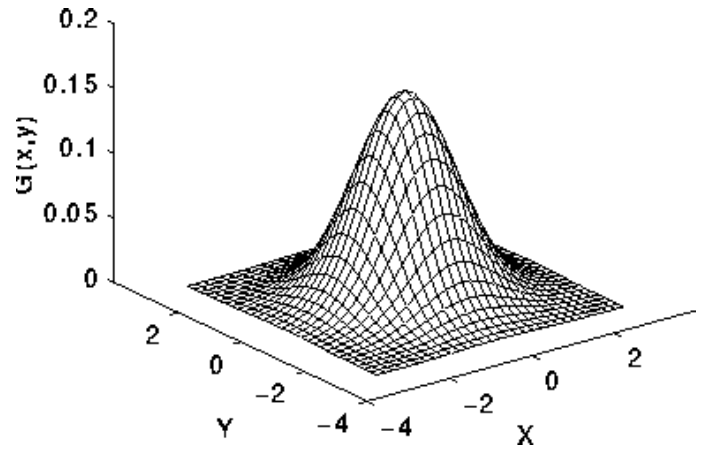


Figure 2.9: 2-D Gaussian function with $\sigma = 1.4$.

After performing the equation in 2.10 it is necessary to remove the pixels that are not considered edges. This is done by non-maximum suppression: at each center pixel of a 3x3 mask (for example) a gradient magnitude comparison will be made with its neighbors, i.e., if the gradient direction of the center pixel is pointing in a north-south direction (0 or 270 degrees), the gradient magnitude comparison will be made with the east-west (90 and 180 degrees) direction pixels. If the center pixel has a higher gradient magnitude than the magnitude gradient at pixels in the east-west directions of a 3x3 mask, the value will be preserved. However, if the opposite is ascertained, the center pixel must be eliminated, i.e. its gradient magnitude value is set to zero.

After the non-maximum suppression is computed, the pixels with values other than zero (in a gradient magnitude matrix) are the candidates to be the edges of a given image. However, it is necessary to be sure which candidates are the real ones (edges). Thus, a hysteresis (two empirical thresholds) is used for this purpose. If the current gradient pixel is higher than the upper threshold, it means that the pixel is an edge (set white value). If the current gradient pixel is lower than the lower threshold, it means that the pixel is not an edge (set black pixel). However, if the current gradient pixel is between these thresholds, will be set as an edge if one of its neighbours is an edge.

- **Laplacian:** While the kernels described above (Sobel, Prewitt, Roberts, Differentiation and Canny) are an approximation of the first derivative, the Laplacian is a filter to obtain the second derivative as shown in equation 2.17.

$$L(x,y) = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2} \tag{2.17}$$

Where f is the pixel intensity value in x and y coordinates.

To simplify the calculations, the second derivative is approximated by a convolution between a mask and the input image, as shown in the following equation (equation 2.18).

$$G = \begin{bmatrix} 0 & -1 & 0 \\ -1 & +4 & -1 \\ 0 & -1 & 0 \end{bmatrix} * T \qquad or \qquad G = \begin{bmatrix} -1 & -1 & -1 \\ -1 & +8 & -1 \\ -1 & -1 & -1 \end{bmatrix} * T \tag{2.18}$$

Where G is the gradient magnitude. As shown in equation 2.18, there are two common Laplacian masks to detect edges. However the output can give false edges because of its noise sensitivity. To suppress the high frequency noise, a combination is made between the Gaussian filter (low-pass filter) and the Laplacian. This combination is known as LoG, and can be represented by an equation that makes the convolution between image T and LoG, as shown in 2.19.

$$LoG_{i,j} = -\frac{1}{\pi\sigma^4}\left[1 - \frac{(i-(k+1))^2 + (j-(k+1))^2}{2\sigma^2}\right]\exp\left(-\frac{(i-(k+1))^2 + (j-(k+1))^2}{2\sigma^2}\right) \tag{2.19}$$

Where k is the LoG kernel size, i and j are the LoG kernel row and column, respectively. Rows and columns also need to be in interval between 1 and $(2k+1)$, as mentioned before.

The following Figure 2.10 and equation 2.20 represent the result of Equation 2.19 and its kernel, respectively. The kernel size is equal to 9 and its sigma equal to 1.4. Thus, with the kernel shown in equation 2.20 it is possible to suppress the high frequency noise and avoid false edges.

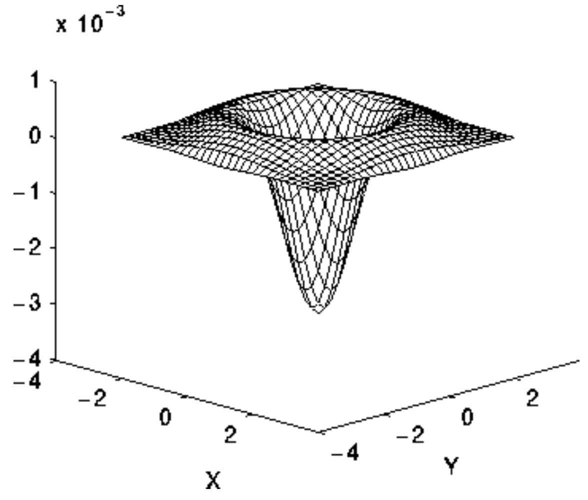


Figure 2.10: The 2-D LoG function with $\sigma = 1.4$ and $k = 9$.

$$LoG\ Filter = \begin{bmatrix} 0 & +1 & +1 & +2 & +2 & +2 & +1 & +1 & 0 \\ +1 & +2 & +4 & +5 & +5 & +5 & +2 & +2 & +1 \\ +1 & +4 & +5 & +3 & 0 & +3 & +5 & +4 & +1 \\ +2 & +5 & +3 & -12 & -24 & -12 & +3 & +5 & +2 \\ +2 & +5 & 0 & -24 & -40 & -24 & 0 & +5 & +2 \\ +2 & +5 & +3 & -12 & -24 & -12 & +3 & +5 & +2 \\ +1 & +4 & +5 & +3 & 0 & +3 & +5 & +4 & +1 \\ +1 & +2 & +4 & +5 & +5 & +5 & +2 & +2 & +1 \\ 0 & +1 & +1 & +2 & +2 & +2 & +1 & +1 & 0 \end{bmatrix} \tag{2.20}$$

2.3.1.1 Histogram of Oriented Gradients (HOG)

Now that the difference between some of these edge detection filters (Sobel, Prewitt, Roberts, Differentiation, Canny and Laplacian) as been explained, it is important to understand how to use these filters to extract features in order to classify the terrains. A well-known algorithm is called HOG and will be explained in this section (to exemplify the edge detection filters, Figure 2.11 shows the Sobel calculation for detecting the image edges).

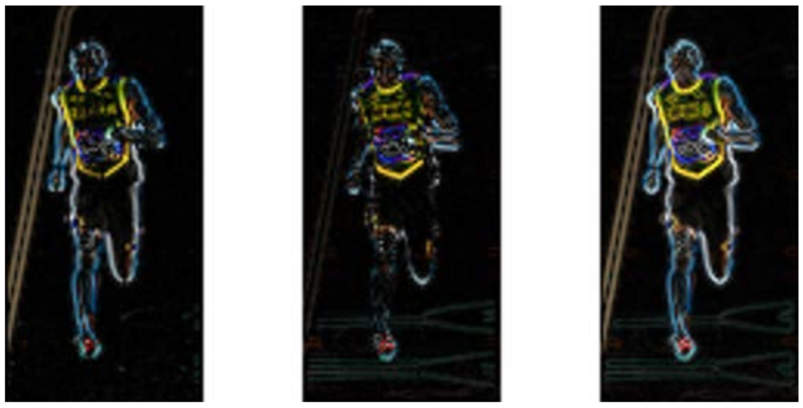


Figure 2.11: Sobel gradient results from Figure 2.7- a) X-gradient result from G_x(equation 2.10); b) Y-gradient result from G_y(equation 2.10); c) Gradient Magnitude from a) and b) (Mallick, 2016).

Once the gradient magnitude as shown in Figure 2.11 c) has been calculated, gradient direction also needs to be used as mentioned above in Equation 2.9. As described at the beginning of this section (Section 2.3.1), a histogram of bins is used in order to suppress the noise. Thus, to reach this goal, the image was divided into squares with a dimension of 8 by 8, as shown in Figure 2.12, in order to produce a gradient of magnitude and direction mean and therefore exclude possible outliers.

Figure 2.13 shows detailed information in a block of 8 by 8 wherein each coordinate of the matrix's magnitude and direction gradients is calculated. In the center of Figure 2.13, the gradient magnitude and direction with arrows is shown:

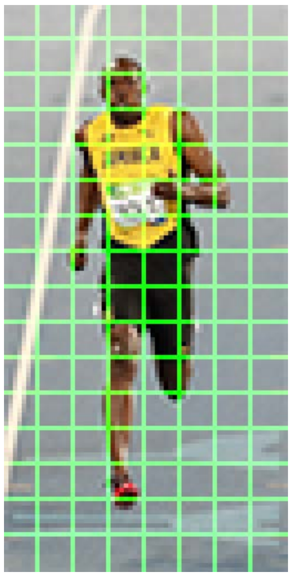
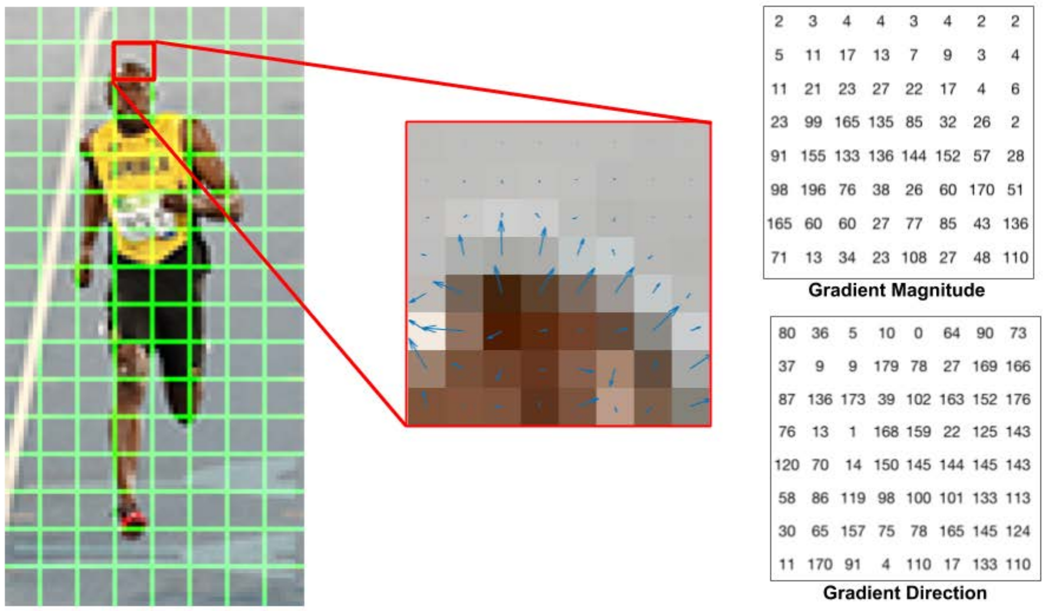


Figure 2.12: Original image divided into 8x8 squares (Mallick, 2016).



2	3	4	4	3	4	2	2
5	11	17	13	7	9	3	4
11	21	23	27	22	17	4	6
23	99	165	135	85	32	26	2
91	155	133	136	144	152	57	28
98	196	76	38	26	60	170	51
165	60	60	27	77	85	43	136
71	13	34	23	108	27	48	110

Gradient Magnitude

80	36	5	10	0	64	90	73
37	9	9	179	78	27	169	166
87	136	173	39	102	163	152	176
76	13	1	168	159	22	125	143
120	70	14	150	145	144	145	143
58	86	119	98	100	101	133	113
30	65	157	75	78	165	145	124
11	170	91	4	110	17	133	110

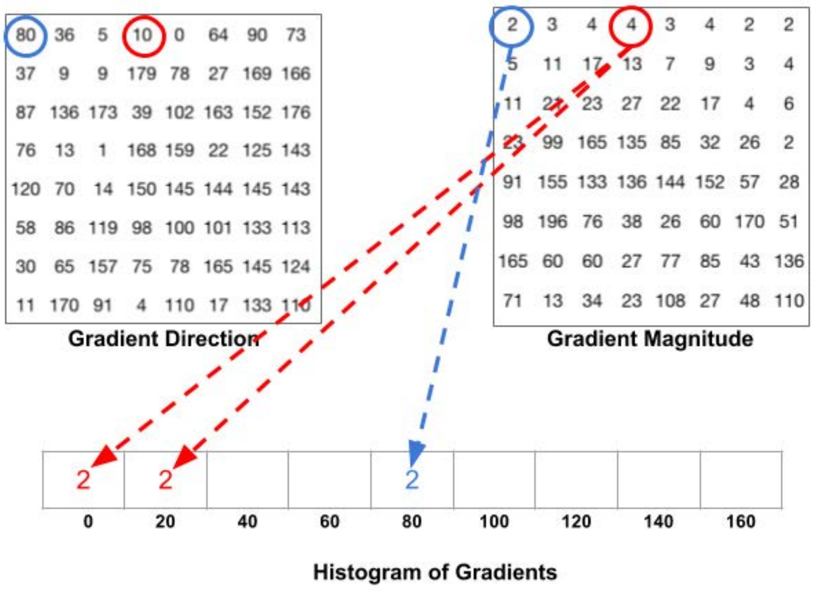
Gradient Direction

Figure 2.13: The magnitude and direction gradients represented using arrows in an 8 by 8 block. Adapted from (Mallick, 2016).

- The larger the arrow length, the greater the magnitude gradient of the pixel will be (Equation 2.8);
- The arrow direction is represented by the result of the gradient direction as shown in Equation 2.9.

The right side of Figure 2.13 provides detail information about the magnitude and the direction gradient of each pixel (in an 8 by 8 matrix). Regarding the gradient direction results, a difference should be noted: the angles only vary between 0 and 180 degrees (in a clockwise direction) and do not follow the standard values from 0 to 360 degrees. This difference is called "unsigned" gradients, because the symmetric of each gradient is given by the same direction gradient value. The reason for using this model and not following the standard ones has to do with the fact that, empirically, the results obtained are better using the values between 0 and 180 degrees than values between 0 and 360 degrees.

After calculating the values of the gradient magnitude and direction, as observed in Figure 2.13, the final step is to build the nine-bin histogram of gradients to omit the outlier values and intensify the most common values. A bin is selected based on the direction (gradient direction), and the veto (value that goes to the bin histogram).


Gradient Direction

80	36	5	10	0	64	90	73
37	9	9	179	78	27	169	166
87	136	173	39	102	163	152	176
76	13	1	168	159	22	125	143
120	70	14	150	145	144	145	143
58	86	119	98	100	101	133	113
30	65	157	75	78	165	145	124
11	170	91	4	110	17	133	110

Gradient Magnitude

2	3	4	4	3	4	2	2
5	11	17	13	7	9	3	4
11	21	23	27	22	17	4	6
23	99	165	135	85	32	26	2
91	155	133	136	144	152	57	28
98	196	76	38	26	60	170	51
165	60	60	27	77	85	43	136
71	13	34	23	108	27	48	110

Histogram of Gradients

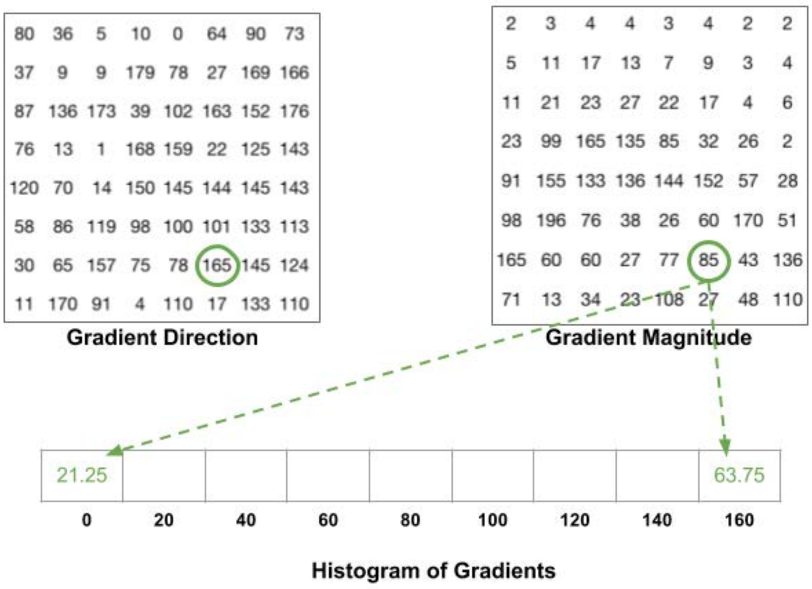
0	20	40	60	80	100	120	140	160
2	2			2				

Figure 2.14: The nine-bin histogram of gradients. Adapted from (Mallick, 2016).

As shown in Figure 2.14, and giving the example of the first value in the gradient direction (the first blue circle: 80) it is noted that its gradient magnitude value is two (blue circle in the gradient magnitude table). Thus this value (two) will be add to the fifth bin.

As regards the red circle (value 10) in the gradient direction, its corresponding gradient magnitude is four. Since 10 (red circle in the gradient direction) is at an equal distance between the first bin and the second bin, its corresponding gradient magnitude value will also be divided by an equal value (in this case two for each bin).

It is also possible that the value of the gradient direction is higher than 160 degrees. In that case it is necessary to take into account the first bin and the last bin. Figure 2.15 is used as an example.



Gradient Direction

80	36	5	10	0	64	90	73
37	9	9	179	78	27	169	166
87	136	173	39	102	163	152	176
76	13	1	168	159	22	125	143
120	70	14	150	145	144	145	143
58	86	119	98	100	101	133	113
30	65	157	75	78	165	145	124
11	170	91	4	110	17	133	110

Gradient Magnitude

2	3	4	4	3	4	2	2
5	11	17	13	7	9	3	4
11	21	23	27	22	17	4	6
23	99	165	135	85	32	26	2
91	155	133	136	144	152	57	28
98	196	76	38	26	60	170	51
165	60	60	27	77	85	43	136
71	13	34	23	108	27	48	110

Histogram of Gradients

0	20	40	60	80	100	120	140	160
21.25								63.75

Figure 2.15: An example of a nine-bin gradient histogram when the angle of gradient direction is greater than 160 degrees. Adapted from (Mallick, 2016).

As observed in Figure 2.15, it is not easy or direct to know the value that will go to the first and last bin. Thus, Equation 2.21 returns the result values from the relationship between the limiting angles (in this case 160 and 180 degrees) and the value of the gradient direction under study (in this case 165 degrees).

$$A_{bin} = \left(1 - \frac{\left|B_{bin} - Gradient_{Direction_{value}}\right|}{\left|A_{bin} - B_{bin}\right|}\right) \cdot Gradient_{Manitude_{value}} \tag{2.21}$$

Where A_{bin} and B_{bin} are the gradient magnitude limit angles (in this case, when A_{bin} is 180, the B_{bin} limit is 160 and vice-versa).

When all pixels from this 8x8 matrix are calculated, it is possible to build the final nine-bin gradient histogram as shown in Figure 2.16.

As can be seen in Figure 2.16, the histogram has a greater weight at 0 and 180 degrees. This means that the gradient direction is pointing either up or down (north or south, respectively).

Some works studied in this dissertation (Ghosh and Sharma, 2015; Li et al., 2016) demonstrated the high efficiency of the HOG for classifying different types of terrain. However, the author in (Ghosh and Sharma, 2015) developed an improved HOG known as Histogram of Gradient Magnitudes (HGM) . The big difference between HGM and HOG is that HGM is invariant of rotation, as can be seen in Figure 2.17.

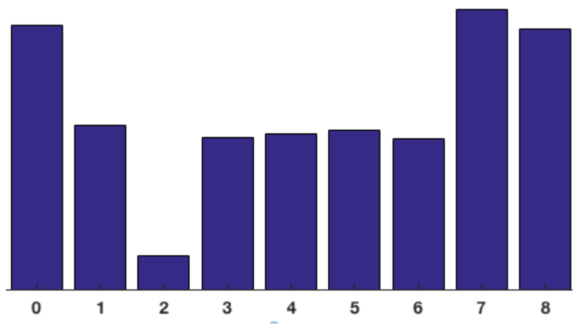


Figure 2.16: The nine-bin histogram of gradients- Final result in an 8x8 matrix. Adapted from (Mallick, 2016).

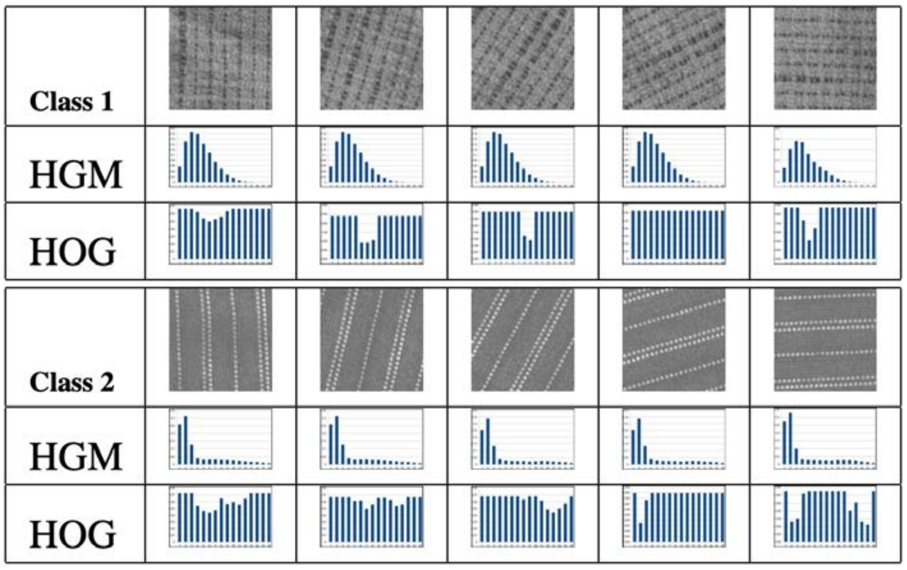


Figure 2.17: HOG vs HGM in different rotations (Ghosh and Sharma, 2015).

In this way, the algorithm proposed in (Ghosh and Sharma, 2015) becomes more robust to image rotations, either from satellites or UAVs. Also, the HGM was able to classify, based on different datasets, seven distinct classes: water, buildings, trees, sky, people, cars and dogs. Figure 2.18 shows the results.

In (Li et al., 2016) a histogram of oriented gradients based gist (HOG-gist) was proposed for building recognition. This approach computes the normalized histograms of multi-orientation gradients for the same image with four different scales.

2.3.2 Gabor Filter

This section presents another method for extracting terrain's static textures- the Gabor method. This is able to choose multiple texture directions. The Gabor filter is also the impulse response formed by a multiplication of a sinusoidal signal with a Gaussian envelope function and can be computed using the following complex Equation 2.22:

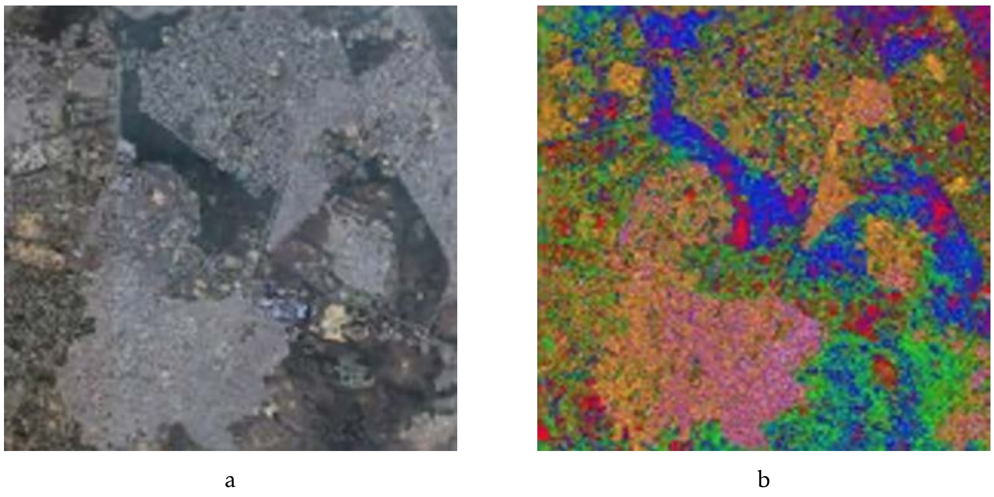


Figure 2.18: HGM algorithm proposed in (Ghosh and Sharma, 2015): a) input raw image; b) segmentation results by HGM.

$$G(x,y,\lambda,\theta,\psi,\sigma,\gamma) = e^{\left(-\frac{x'^2+\gamma^2 y'^2}{2\sigma^2}\right)} e^{\left(i\left(2\pi\frac{x'}{\lambda}+\psi\right)\right)} \tag{2.22}$$

Its real and imaginary components can be obtained by equations 2.23 and 2.24, respectively:

$$G(x,y,\lambda,\theta,\psi,\sigma,\gamma) = e^{\left(-\frac{x'^2+\gamma^2 y'^2}{2\sigma^2}\right)} \cos\left(2\pi\frac{x'}{\lambda}+\psi\right) \tag{2.23}$$

$$G(x,y,\lambda,\theta,\psi,\sigma,\gamma) = e^{\left(-\frac{x'^2+\gamma^2 y'^2}{2\sigma^2}\right)} \sin\left(2\pi\frac{x'}{\lambda}+\psi\right) \tag{2.24}$$

where:

$$x' = x\cos(\theta) + y\sin(\theta) \tag{2.25}$$

$$y' = -x\sin(\theta) + y\cos(\theta) \tag{2.26}$$

These equations (2.22, 2.23 and 2.24) require as input parameters:

- **x and y:** Filter coordinates, where x represents the columns and y the rows;
- **Lambda** (λ): Represents the sinusoid's wavelength;
- **Theta (θ):** Defines the Gaussian envelope orientation;
- **Psi (ψ):** Symbolizes the phase offset;
- **Sigma (σ):** Describes the Gaussian envelope size;

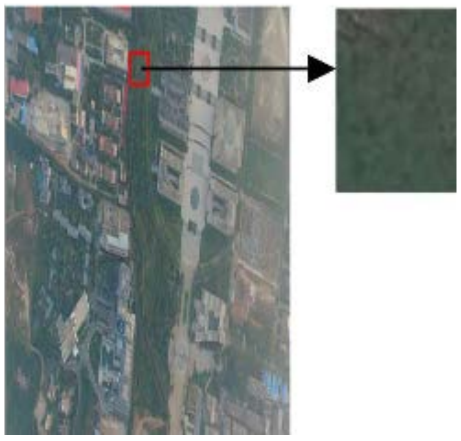
- **Gamma (γ):** Reflects the shape of the ellipse in the gabor filter space.

To simplify and increase the system speed, normally only the real component of the Gabor function (equation 2.23) is used. After obtaining the multiplication of a Gaussian with a sinusoidal function, i.e. the kernel of the filter, it will be convolved with the original image (equation 2.27). The result of the Gabor filter applied over a water surface is presented in Figure 3.9.

$$f[x,y]*g[x,y] = \sum_{-n1}^{n1}\sum_{-n2}^{n2} f[n1,n2]\cdot g[x-n1,y-n2] \tag{2.27}$$

Some of the works studied in this dissertation relating to terrain classification (Acharya et al., 2016; Hofmann et al., 1998; Ma et al., 2017) proved the usefulness of the Gabor filter to extract texture characteristics. In (Ma et al., 2017) the combination of the Gabor filter and a sparse-representation-based classification using a DJI Phantom 3 Advanced UAV (Figure 2.19) was studied. The author of (Ma et al., 2017) was able to classify four different terrain types: grass, soil, water and trees.

a



b

Figure 2.19: Gabor algorithm used in (Ma et al., 2017): a) DJI Phantom 3 Advanced UAV; b) Aerial sample images with different terrain types.

Gabor filtering has also been used in the biomedical field (Acharya et al., 2016), in unsupervised texture segmentation in a deterministic annealing framework (Hofmann et al., 1998), showing its great potential.

2.3.3 Local Binary Patterns

Another visual descriptor to extract the image textures is the Local Binary Patterns (LBP) method. LBPs are very simple to design and compute, however they are very powerful texture descriptors. Contrary to many descriptors that evaluate the image textures as a whole (as explained in Sections 3.2.5 and 3.2.6), LBPs compute and evaluate the texture locally (Ojala et al., 2002), i.e., in each pixel neighborhood.

The following steps explain how an LBP filter is formed and how it can be used in terrain classification (within the scope of this dissertation):

1. **Gray scale conversion:** The first step is to convert an RGB image into a gray image in order to eliminate some high frequencies in these three channels (red, green and blue);
2. **Build a mask:** Before calculating the texture value of each pixel, it is important to choose the size, r, of the mask to be used. In this section, a 3x3 window is used in relation to the center pixel;
3. **Thresholds:** The center pixel is then compared with its neighbors. The ones with a value higher than the center pixel's value are returned as zero, while those that have a smaller-or-equal value relative to the center pixel are returned as one. These results are shown in Figure 2.20;

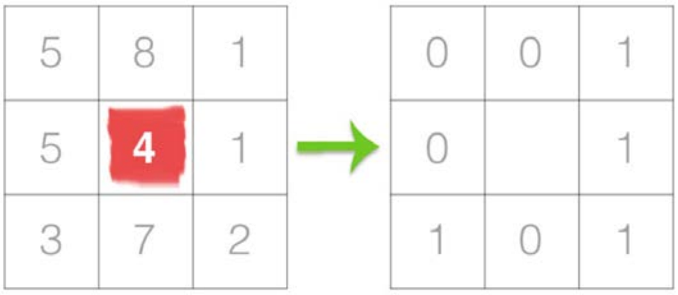


Figure 2.20: Comparison between the center pixels and its neighbors (Rosebrock, 2015).

4. **Binary Code:** After calculating the outputs for the neighbors, the next step is to calculate the center pixel value. It is possible to start from any neighbor and work clockwise or counter-clockwise. However, it is imperative to be consistent throughout the whole image. In this example, the start point is at the top-center and it is rotating counter-clockwise. Figure 2.21 describes the 8-bit binary code converted into a decimal value. The value obtained (in this case 23) will be the new value of the center pixel;

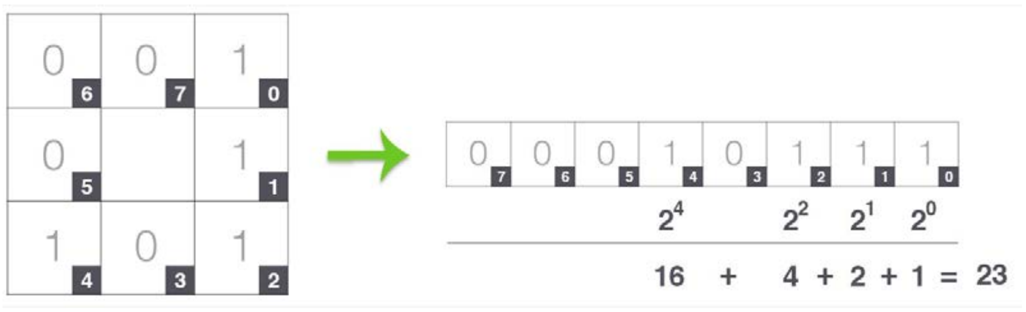


Figure 2.21: Center pixel value. Conversion of 8-bit binary neighbourhood into a decimal value (Rosebrock, 2015).

5. **New 2-D array:** The new values are saved in a new 2-D array that will represent the LBP of a given input image as shown in Figure 2.22;

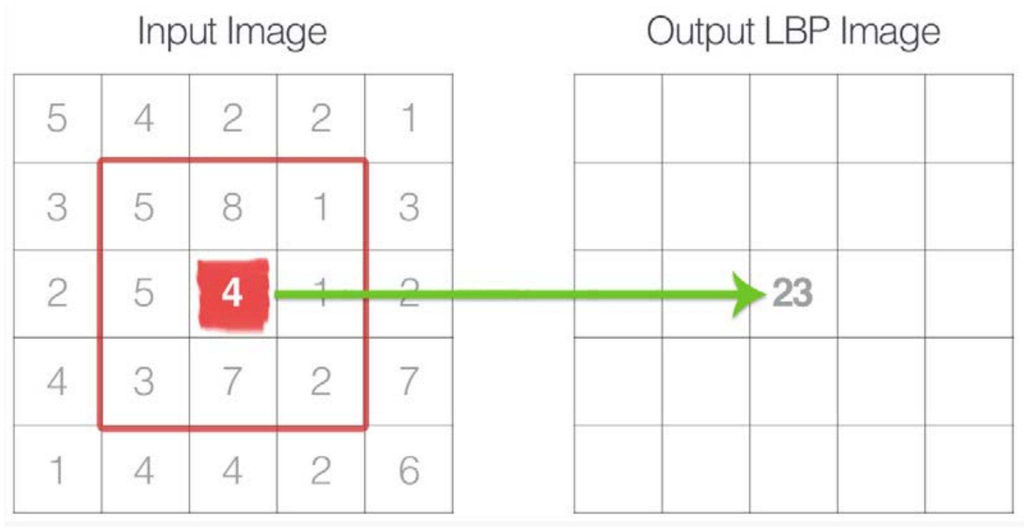


Figure 2.22: LBP result. Left image is the input image and the right image is the new 2-D image (Rosebrock, 2015).

6. **Histogram:** Being a powerful tool, and as explained in section 2.2, the histogram is very useful for image processing and in this case it is not an exception. In this dissertation, where the goal is terrain classification, it is necessary to classify the terrain type. Thus, the histogram is able to give us concrete values that can differentiate between different terrains.

   A histogram of all the values returned by LBP located in the new 2-D array is then performed. Since the 3x3 mask has $2^8 = 256$ different values, the limits of this histogram are between 0 and $2^8 - 1 = 255$. Figure 2.23 shows an example of a LBP histogram. Looking at Figure 2.23 it is important to realize that the values on the ordinate axes are a normalization so that the posterior empirical thresholds are general and independent of the images' sizes.

Now that the LBP filter implementation has been explained, it is important to understand what are the main advantages and disadvantages of this algorithm. With regard to its advantages, as mentioned before, LBPs are easy-to-compute and fast algorithms. These are good advantages because rapid processing is required to be able to classify terrains in an emergency landing. However, having a mask with a 3x3 dimension is a disadvantage due to the fact that it picks up a lot of noise. Thus, in (Ojala et al., 2002) a way of avoiding this problem was proposed.

As can be seen in Figure 2.24, two parameters were added in order to be able to build a dynamic mask:

- Numbers of symmetrical p points in order to avoid square masks;

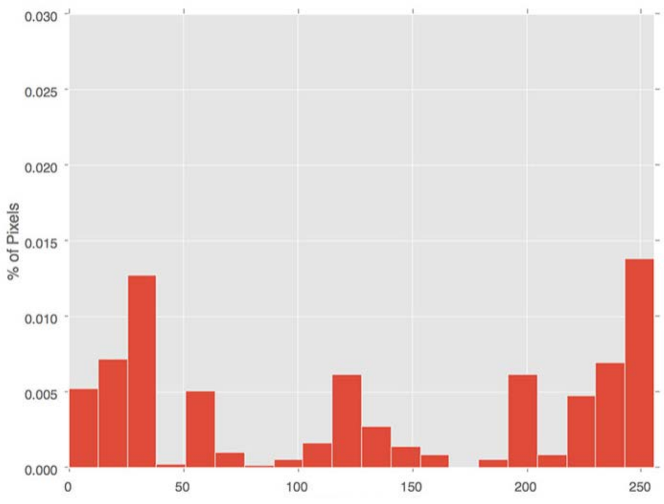


Figure 2.23: LBP histogram (Rosebrock, 2015).

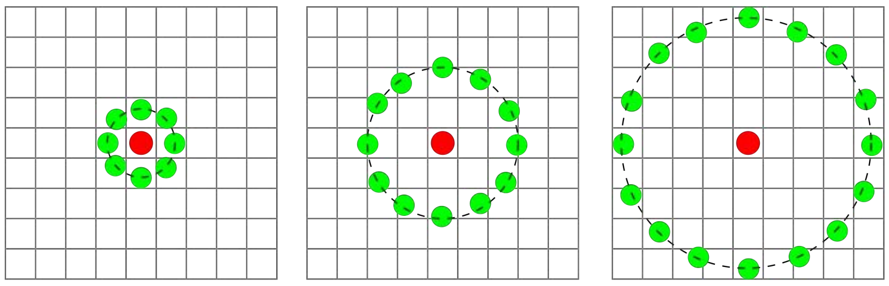


Figure 2.24: LBP histogram (Rosebrock, 2015).

- The radius of the circle r so that it is possible to build masks at different scales.

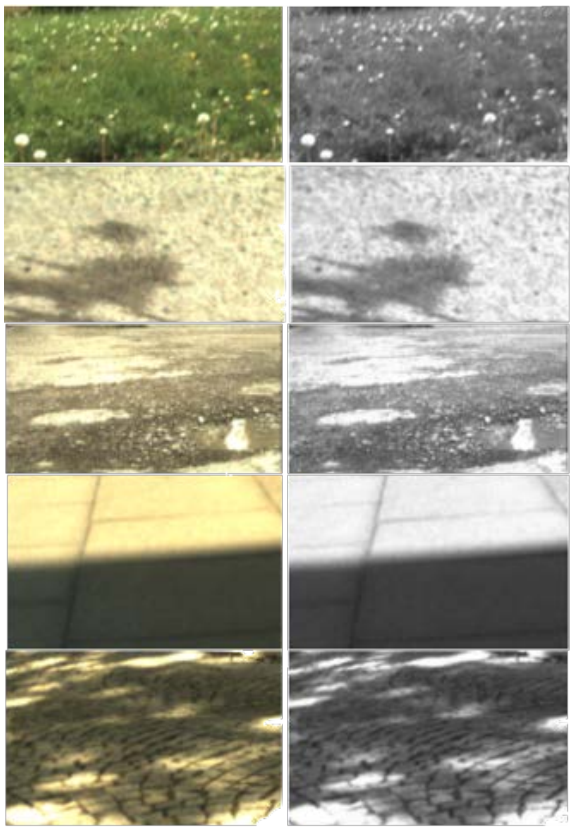
Some works studied in this dissertation (Khan et al., 2011; Mboga et al., 2017; Ojala et al., 2002; Tong et al., 2018) showed the good efficiency of the LBPs for the classification of different terrain types. In (Khan et al., 2011) a USV was used (Figure 2.25 a)) to classify five different terrain types as shown in Figure 2.25 b). The author used an outdoor robot to extract the terrain textures, and using the LBP algorithm combined with a classifier, it was possible to classify these five different terrain types.

Besides the terrain classification topic, LBPs have also been used in medical image analysis (Nanni et al., 2010), face recognition problems (Ahonen et al., 2007; Huang et al., 2011), whereas in (Satpathy et al., 2014) they were used in object recognition tasks.

2.3.3.1 Local Ternary Patterns

Being developed from the LBP, Local Ternary Patterns (LTP) use a threshold k around the center pixel value c (Tan and Triggs, 2010). Instead of creating a binary code using just the center pixel c as reference (as explained in Section 2.3.3), the binary code is constructed from a comparison between the neighbors and the center pixel with a relation of k and c, as follows:

a



b

Figure 2.25: LBP algorithm used in (Khan et al., 2011): a) Outdoor robot; b) Sample images of different terrain types.

$$T = \begin{cases} 1 & T \geq (c+k) \\ 0 & T < (c+k) \text{ } and \text{ } T > (c-k) \\ -1 & T \leq (c-k) \end{cases} \tag{2.28}$$

where T represents the neighbors of the center pixel, c is the center pixel and k is the threshold value. One can observe in Equation 2.28 that it is possible to return three possible values: T will be one if neighbors pixels are greater than or equal to $(c+k)$; T will be zero if neighbors pixels are between $(c-k)$ and $(c+k)$; Finally, T will be -1 if neighbors pixels are smaller than or equal to $(c-k)$.

It should be noted that LTP will create two different matrix types. The first will be a matrix that only contains positive values. The second matrix will only obtain the negative values generated by Equation 2.28.

In the same way as in Section 2.3.3, the last step of using this algorithm is the use of the LBP code binary to return an output to be included in a histogram. Thus, two separate histograms are used in order to return values that differentiate terrain types (in this case because it is within the scope of this thesis).

94	38	54
23	**50**	78
47	66	12

1	-1	0
-1		1
0	1	-1

Figure 2.26: Local Ternary Patterns: Left: 3x3 matrix by input image; Right: LTP output values.

Figure 2.26 shows an example of an input 3x3 matrix image and its respective LTP results. In this example a threshold of $k = 5$ was used. As explained above, two binary patterns are built by the positive and negative values. Thus, the LTP's result is (-1)1(-1)01(-1)10 (following clock wise direction with $k = 5$) and the positive and negative code binaries are:

- **Positive code binary:** 01001010;
- **Negative code binary:** 10100100.

From the literature review (Khan et al., 2011; Tan and Triggs, 2010), it was shown that LTPs are capable of extracting better characteristics than LBPs due to the binary code that is created by the relationship between the central pixel and its neighbors (Khan et al., 2011) as can be seen in Table 2.3.

2.3.3.2 Local Adaptive Ternary Patterns

Based on LTP, Local Adaptive Ternary Patterns (LATP) is an improvement aiming to be less sensitive to changes in luminance and therefore to noise (Akhloufi and Bendada, 2010). Being the same as the LTP, LATPs also compare other parameters with their neighborhood, T. However, for LATPs to be less sensitive to noise and light intensity, the 3x3 mask pixels cannot be compared only by a threshold, k, as in LTP. The following equation (Equation 2.29) shows the other important parameters:

$$T = \begin{cases} 1 \;\; T \geq (\mu + k\sigma) \\ 0 \;\; T < (\mu + k\sigma) \; and \; T > (\mu - k\sigma) \\ -1 \;\; T \leq (\mu - k\sigma) \end{cases} \tag{2.29}$$

Where μ is the local region mean, σ is the local region standard deviation and k is the given threshold. It is therefore important to point out that when compared to the average and with the standard deviation, the values are compared all around zero, which makes the algorithm generic to any terrain under study (thesis dissertation).

In the same way as for LTP, in this algorithm two matrices are also generated because, as shown in Equation 2.29, positive and negative values are generated.

94	38	54
23	**50**	78
47	66	12

1	0	0
-1		1
0	0	-1

Figure 2.27: Local Adaptive Ternary Patterns: Left: 3x3 matrix by input image; Right: LATP output values.

Figure 2.27 shows an example of an input 3x3 matrix image and its respective LATP results. In this example a threshold k was equal to one, with the mean μ equal to 51.33 and the standard deviation σ equal to 25.74. As explained in Section 2.3.3.1, two binary patterns are built by the positive and negative values. From this LATP example (rotating clockwise, as was the case in Section 2.3.3.1) with $k = 1$ the output code is 01(-1)00(-1)10 and the positive and negative code binaries are:

- **Positive code binary:** 0100010;
- **Negative code binary:** 00100100.

The works studied in this dissertation (Akhloufi and Bendada, 2010; Khan et al., 2011) show that LATPs are capable of taking better characteristics than LBPs due to the binary code that is created by the relationship between the central pixel and its neighbors (Khan et al., 2011) as can be seen in Table 2.3.

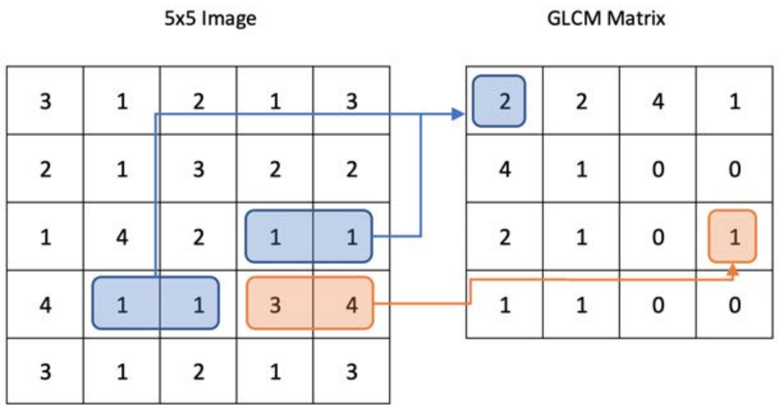
2.3.4 Gray-Level Co-Occurrence Matrix

Sections 2.3.3 presented algorithms that extract and evaluate the local texture information. In Sections 3.2.5 and 3.2.6 the evaluation is all-embracing: the Gray-Level Co-Occurrence Matrix known as GLCM. This algorithm is nothing less than a statistical texture features extraction that represents textures based on the relationship and distribution between pixels of a given static image. The algorithms that evaluate the texture of a frame can be classified as first, second or higher statistical texture orders. The difference between the different orders is that, in the first order it only calculates properties for individual pixels, such as mean and variance from the original image, neglecting the spatial relationship between pixels. However, the second and higher texture orders calculate properties of an image using a spatial relationship between two or more pixels, such as GLCM (Haralick et al., 1973).

To build the GLCM mask, it is critical to understand these three parameters:

1. The distance, d, between i and j pixels;
2. The angular orientation θ chosen;
3. Symmetrical matrix decision.

After understanding the parameters described above, it is possible to create the GLCM matrix. The first step is to create the GLCM $N \times N$ matrix. Consider as an example this dissertation, where the matrix dimension is 256×256, because the input images are defined between 0 and 255 levels (256 different gray levels). Figure 2.28 is an example of a 5×5 GLCM matrix. To explain this in a more simplified and intuitive way, Figure 2.28 will be related to a GLCM matrix with a 5×5 dimension with four gray levels.



3	1	2	1	3
2	1	3	2	2
1	4	2	1	1
4	1	1	3	4
3	1	2	1	3

2	2	4	1
4	1	0	0
2	1	0	1
1	1	0	0

a

Normalized GLCM

0.1	0.1	0.2	0.05
0.2	0.05	0	0
0.1	0.05	0	0.05
0.05	0.05	0	0

b

Figure 2.28: Design of the GLCM matrix from a 5×5 image with four gray levels. (a) GLCM matrix with $d = 1$ and $\theta = 1$; (b) The GLCM normalized matrix.

As can be seen in Figure 2.28 a), the GLCM matrix was designed to have the distance d between pixels equal to 1, with an angular orientation θ equal to 0. The GLCM matrix can be normalized as shown in Figure 2.28 b), by dividing each element by the sum of all image pixels. The co-occurrence matrix can have up to eight different angular orientations (0, 45, 135, 180, 225, 270 and 315 degrees).

Now that the GLCM matrix is defined, the next step is to determine if its transpose is added to the matrix in order to become a GLCM symmetrical matrix (in this example $d = 100$, $\theta = 0$ and a symmetric matrix were chosen):

$$M = M + M^T \tag{2.30}$$

With a GLCM matrix defined, the next step is to understand the texture features proposed by Haralick (Haralick et al., 1973).Shown below are the most commonly used texture features in texture classification:

$$Contrast = \sum_{i=0}^{N-1}\sum_{j=0}^{N-1} |i-j|^2 \cdot p(i,j) \tag{2.31}$$

$$Correlation = \sum_{i=0}^{N-1}\sum_{j=0}^{N-1} \frac{(i-\mu_x)(j-\mu_y)\cdot p(i,j)}{\sigma_x \cdot \sigma_y} \tag{2.32}$$

$$Energy = \sum_{i=0}^{N-1}\sum_{j=0}^{N-1} p(i,j)^2 \tag{2.33}$$

$$Homogeneity = \sum_{i=0}^{N-1}\sum_{j=0}^{N-1} \frac{p(i,j)}{1+|i-j|} \tag{2.34}$$

$$Entropy = -\sum_{i=0}^{N-1}\sum_{j=0}^{N-1} p(i,j)\cdot \log_{10}(p(i,j)) \tag{2.35}$$

$$Variance = \sum_{i=0}^{N-1}\sum_{j=0}^{N-1} (i-\mu)^2 \cdot p(i,j) \tag{2.36}$$

where:

$p(i,j) = (i,j)^{th}$ coordinates in a GLCM normalized matrix as shown in Figure 2.28 b), and:

$$\mu = \sum_{i=0}^{N-1}\sum_{j=0}^{N-1} p(i,j) \tag{2.37}$$

$$\mu_x = \sum_{i=0}^{N-1}\sum_{j=0}^{N-1} i \cdot p(i,j) \tag{2.38}$$

$$\mu_y = \sum_{j=0}^{N-1}\sum_{i=0}^{N-1} j \cdot p(i,j) \tag{2.39}$$

$$\sigma_x^2 = \sum_{i=0}^{N-1}(\sum_{j=0}^{N-1} p(i,j) - \mu_x)^2 \tag{2.40}$$

$$\sigma_y^2 = \sum_{j=0}^{N-1}(\sum_{i=0}^{N-1} p(i,j) - \mu_y)^2 \tag{2.41}$$

Equations (2.31), (2.32), (2.33), (2.34), (2.35) and (2.36), that were used in this dissertation, have different meanings:

- **Contrast:** Used to return the intensity contrast between a pixel and its neighbor throughout the entire image;
- **Correlation:** This method is important to tell how a pixel is correlated with its neighbor throughout the entire image;
- **Energy:** Also known as angular second moment, the goal is to evaluate how uniform an image is;
- **Homogeneity:** Known as Inverse Difference Moment, this equations returns 1 when the GLCM is uniform (diagonal matrix);
- **Entropy:** This feature measures the randomness of intensity distribution. The greater the information's heterogeneity in an image, the greater the entropy's value is. However, the greater the homogeneity, the more the entropy tens towards zero;
- **Variance:** Represents the degree of dispersion of the values around the mean.

Some of the works studied in this dissertation relating to terrain classification, (Caridade et al., 2008; Fraczek et al., 2018; Mboga et al., 2017; Tong et al., 2018) made great use of the GLCM algorithm to extract characteristics from Table 3.3 in Section 3.2.5. According to (Tong et al., 2018) the use of features from the GLCM matrix was studied, in combination with an SVM classifier.

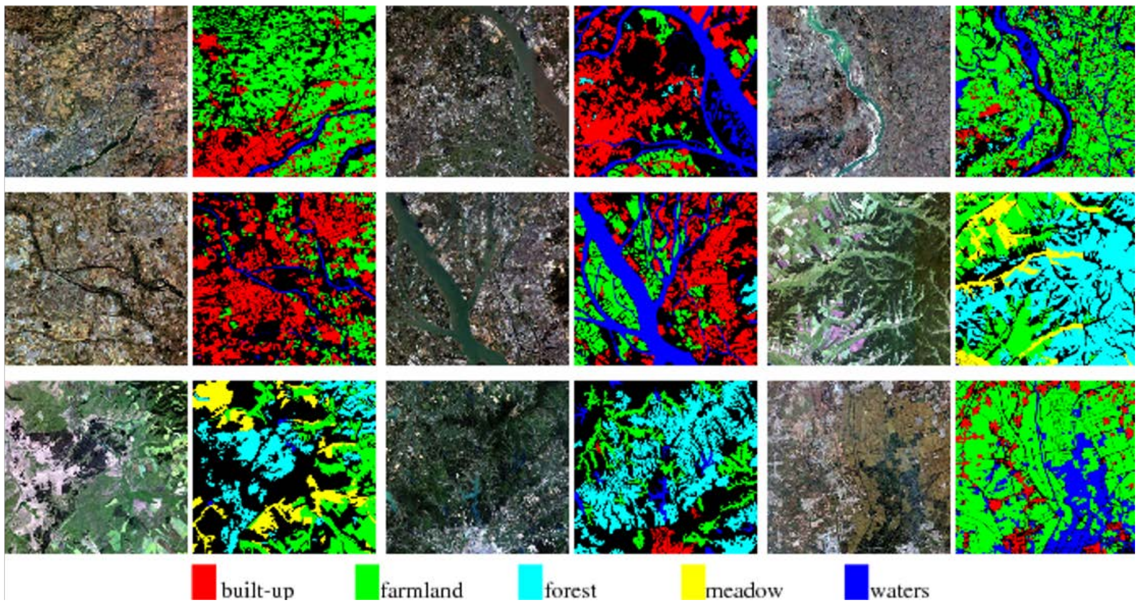


Figure 2.29: Some samples and corresponding class labels (Tong et al., 2018).

Thus, from the satellite images (Figure 2.29) it is possible to segment five class types: built-up, farmland, forest/vegetation, meadow and water. One of the big issues with this type of algorithm in satellite images is the choice of distance "d". The value of d must be large enough to include the texture pattern, but also small enough to retain the local character of spatial dependence. Therefore, small changes in this value greatly influence the results due to the low resolution of satellite images.

In (Caridade et al., 2008), the GLCM algorithm was also applied to satellite images, as can be seen in Figure 2.30 a).

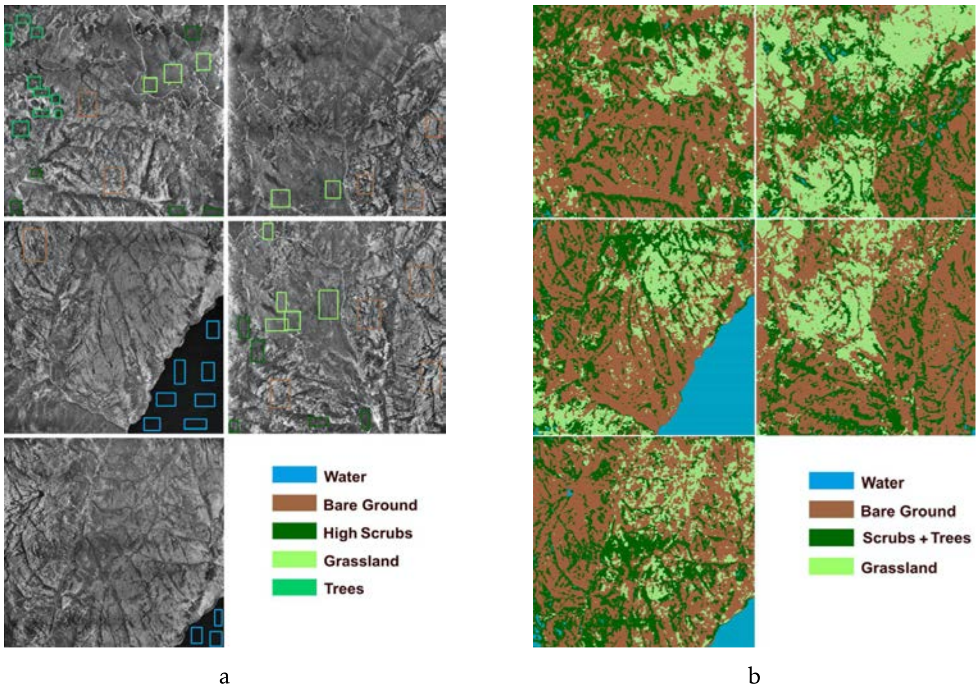


Figure 2.30: Combination with GLCM algorithm and Euclidean Classifier (Caridade et al., 2008): a) Training areas for the five land cover classes; b) Final classified images (four classes).

The author in (Caridade et al., 2008) studied in depth the results of changing the distance d and the angular orientation θ. To segment the outputs into four possible classes, the author tested three different classifiers:

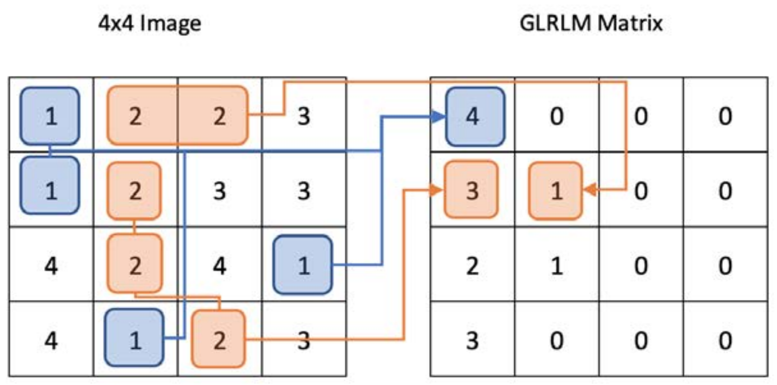
1. Euclidean Classifier;
2. Mahalanobis Classifier;
3. Bayes Classifier.

This is the best system for combining the GLCM algorithm with an Euclidean Classifier.

In addition to the use of the GLCM algorithm for terrain classification, GLCM is also used in other areas, such as the biomedical field, where this algorithm has been used to distinguish between benign and malignant breast lesions (Garra et al., 1993). Other biomedical applications can be found in (Nailon, 2010).

2.3.5 Gray-Level Run Length Matrix

This section presents the second global texture evaluation algorithm, the Gray-Level Run Length Matrix (GLRLM). It was introduced in (Galloway, 1975) to define various textural features. Like the GLCM, the GLRLM also evaluates the distribution of gray levels in an image or in multiple images. As explained in Section 2.3.4, whereas the GLCM evaluates the gray levels within neighboring pixels (taking into account the distance d and the angle θ), the GLRLM evaluates run lengths. A run length is defined as the length of a consecutive sequence of pixels with the same gray level intensity along direction t.



a

Normalized GLRLM

0.29	0	0	0
0.21	0.07	0	0
0.15	0.07	0	0
0.21	0	0	0

b

Figure 2.31: Design of the GLRLM matrix from a 4×4 image with four gray levels. (a) TheGLRLM matrix being $t = 0$; (b) The GLRLM normalized matrix.

As was done in Section 3.2.5, instead an 5×5 matrix, a 4×4 matrix was created to make the explanation easier to understand. As shown in Figure 2.31, in each line the length (columns) of each gray level intensity will be calculated. The GLRLM may also have up to eight different angular orientations, t, (0, 45, 135, 180, 225, 270 and 315 degrees).

After the GLRLM matrix is created, the same features mentioned in section 3.2.5 can be used and calculated on GLRLM matrix.

After explaining the operation of this algorithm, it is necessary to show its applicability. Although no applications have been found in the field of terrain classification, there are examples in the biomedical field showing the benefit of using the GLRLM (Castellano et al., 2005). The authors of (Vamvakas et al., 2018) used the GLRLM to differentiate glioblastoma multiforme from solitary metastasis. In other application fields, the GLRLM has been used to classify different varieties of maize seeds (Wang et al., 2015) and for the automated recognition of drill core textures (Pérez-Barnuevo, 2017).

2.4 Optical Flow

While the previous sections (Section 2.2 and Section 2.3) have explained static features for terrain classification, it is also important to study the dynamic behavior of the terrains under study in this dissertation. Thus, it is necessary to understand the concept of Optical Flow (OF).

OF is defined as the change of light in an image, e.g., on the retina or in a cameras sensor, due to the motion of the scene relative to the eyeball or the camera. In a bioinspired context, the changes of light captured by the eye's retina lead to an impression of movement of the object/scene projected onto the retina. In a more technical context of computer vision, changes in a computer vision environment are represented by a series of image frames.

Figure 2.32 shows three frames with an object in motion, separated by a spatial and a temporal sampling of the incoming light captured by the camera. The computed OF captures the changes in these frames through a vector field. From frame 1 to frame 2, the OF 1-2 is computed, capturing the movement of each pixel in that time difference. In general, OF algorithms from a pixel in the first image look for a nearby pixel in the second image with the same brightness.

OF methods try to calculate the motion at every pixel position between two image frames which are taken at times t and $t + \Delta t$. These methods are called differential since they are based on local Taylor series approximations of the image signal.

The following brightness constancy constraint can be given as follows (equation 2.42):

$$I(x,y,t) = I(x + \Delta x,\ y + \Delta y,\ t + \Delta t) \tag{2.42}$$

where Δx, Δy and Δt are the motion between the two image frames, (x,y,t) is the pixel location and I(x,y,t) is the pixel value.

Assuming the movement is small, the image constraint at I(x,y,t) with Taylor series can be developed to get:

$$I(x + \Delta x,\ y + \Delta y,\ t + \Delta t) = I(x,y,t) + \frac{\partial I}{\partial x}\Delta x + \frac{\partial I}{\partial y}\Delta y + \frac{\partial I}{\partial t}\Delta t \tag{2.43}$$

Equation 2.43 provides the following results:

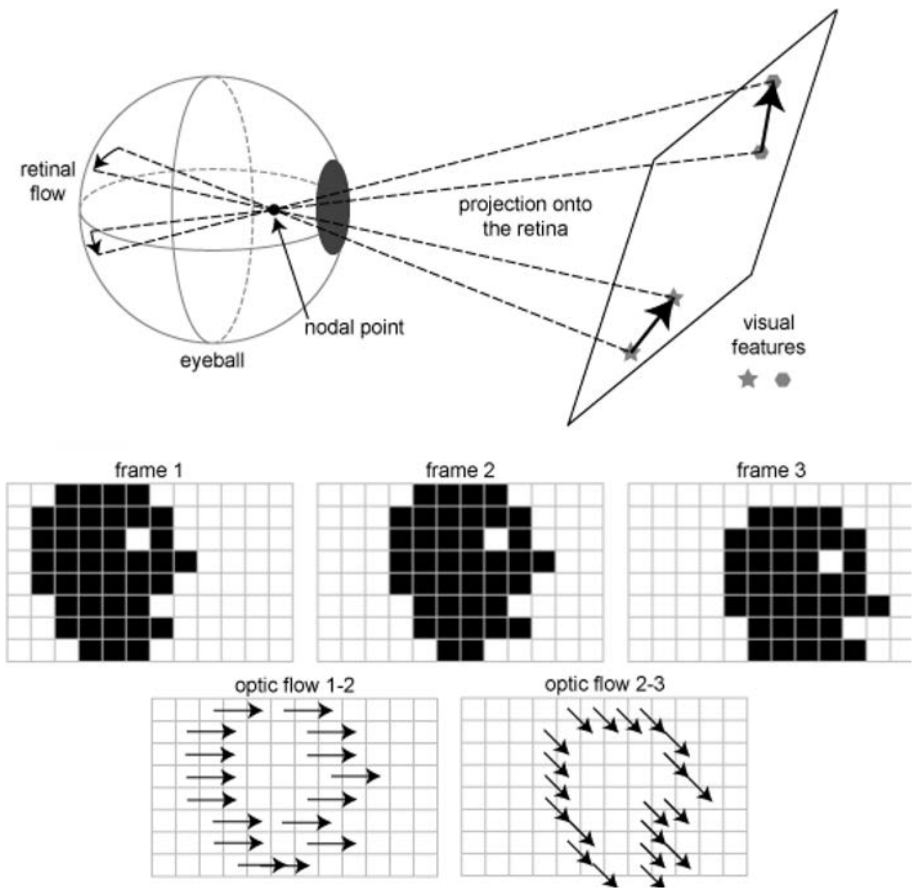


Figure 2.32: Representation of optic flow (Raudies, 2013). It is possible to observe the resulting flows from the sequential frames.

$$\frac{\partial I}{\partial x}\Delta x + \frac{\partial I}{\partial y}\Delta y + \frac{\partial I}{\partial t}\Delta t = 0 \tag{2.44}$$

$$\frac{\partial I}{\partial x}V_x + \frac{\partial I}{\partial y}V_y + \frac{\partial I}{\partial t} = 0 \tag{2.45}$$

where V_x and V_x are the x and y components of the velocity or OF of $I(x,y,t)$ and $\frac{\partial I}{\partial x}$, $\frac{\partial I}{\partial y}$ and $\frac{\partial I}{\partial t}$ are the derivatives of the image at (x,y,t) in the corresponding directions. Using equation 2.45 and replacing the image derivatives by I_x, I_y and I_t, results in the equation 2.46:

$$I_x V_x + I_y V_y = -I_t \tag{2.46}$$

This is an equation with two unknowns (V_x,V_y), and cannot be solved. This is known as the OF algorithm aperture problem (Raudies, 2013) and this issue can be seen in Figure 2.33.

This means that the image's OF cannot be determined. Another set of equations is needed to find the OF, with additional constraints. All OF methods introduce these additional conditions for estimating the actual flow. Some algorithms have been further

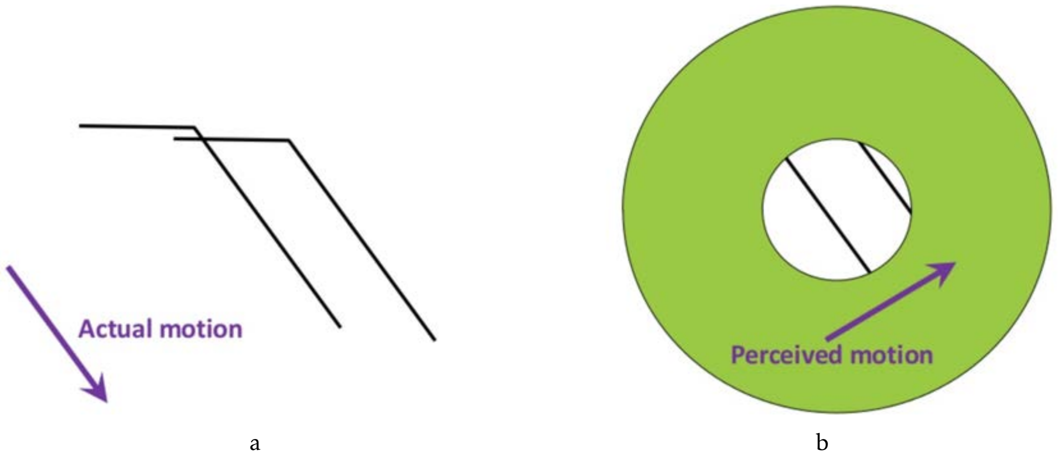


Figure 2.33: Aperture problem example.

developed, expanding the optical flow capabilities. Some of these techniques are classified as global methods (Horn-Schunck), local methods (Lucas-Kanade and Farneback) and region-based matching (Barron et al., 1994). Table 2.2 summarizes the advantages and disadvantages for each of these techniques.

Table 2.2: OF methods comparison.

	Advantages	**Drawbacks**

Global methods	Smooth Flow;
	Global information;
	Accurate time derivatives.
		Slow iterative method;
		Unsharp boundaries.

Local methods	Easy and fast calculation; Accurate time derivatives; Best combination between accuracy and speed.	Error on boundaries.
Region-based matching	Easy to calculate.	Inaccurate time derivatives.

Some of the works researched that use the optical flow concept to classify terrains (Campos et al., 2015; Lookingbill et al., 2007; Pombeiro et al., 2015), until now, without counting the articles published by the author of this dissertation, only one recent publication (Pombeiro et al., 2015) used this concept for terrain classification. The work in (Pombeiro et al., 2015), uses the optical flow concept to extract the dynamic part of an image using the Lucas-Kanade method (Lucas and Kanade, 1981), from an onboard RGB camera as can be seen in Figures 2.34 a) and b).

This work presents a method to determine if the terrain under study is water-type or not using a histogram of orientations by means of the radial dispersion of the trackers as

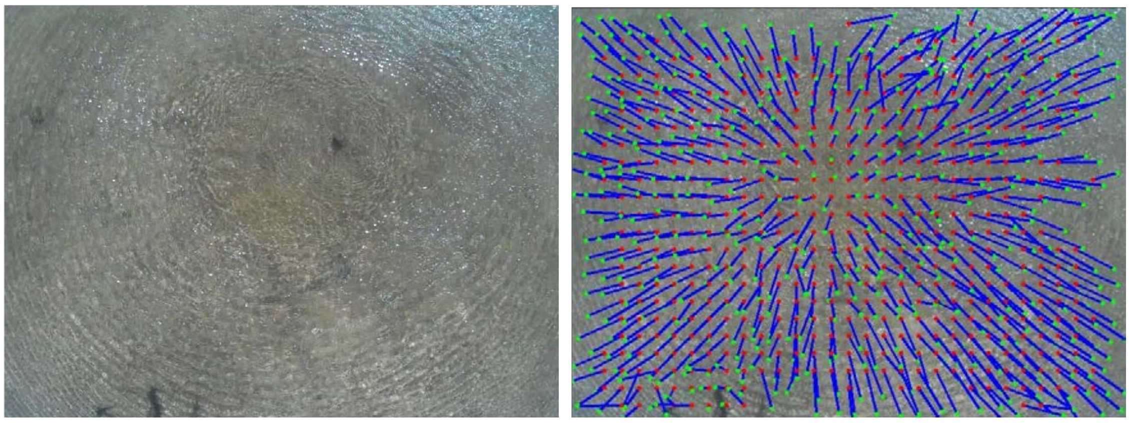


Figure 2.34: Optical flow using the Lucas-Kanade algorithm with the downwash effect. Courtesy of (Pombeiro et al., 2015)

shown in Figure 2.35.

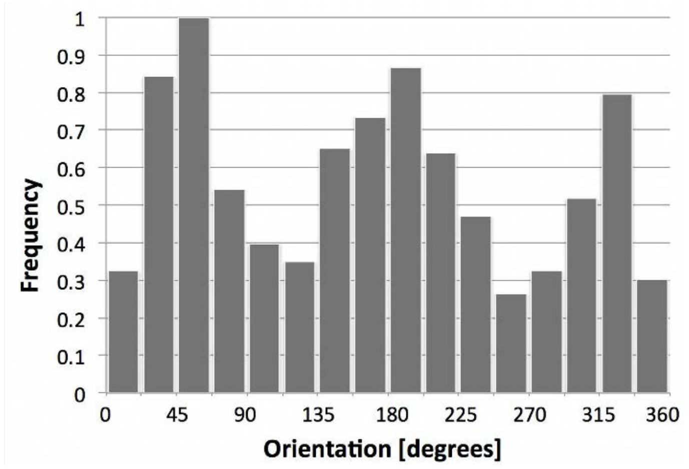


Figure 2.35: Optical flow using the Lucas-Kanade algorithm with the downwash effect. Courtesy of (Pombeiro et al., 2015)

As can be seen from Figure 2.35, the authors state that the downwash effect caused by UAV on water-type terrains gives rise to a circular motion in which the flows detected by the OF start from the center and point outwards from it.

However, the algorithm proposed by the authors in (Pombeiro et al., 2015) has two crucial points:

1. **Robustness of the algorithm:** Since in water-type terrains movement is circular, it is natural that there is a dispersion at all angles of the captured image. However, if the center of the downwash is not in the center of the image, this dispersion may not occur at all angles in water-type terrain. In this way the author may erroneously conclude that he is not on water-type terrain when apparently he is. Consequently, the drone would land and damage its hardware;

2. **Processing time:** In this dissertation, it is necessary to be concerned with developing robust algorithms and at the same time be fast enough to be able to take actions after classifying terrains, such as emergency landings. However, with regard to (Pombeiro et al., 2015), the extremely long algorithm's processing time is an issue, since it needs at least four seconds to classify whether the terrain under study is water-type or not. That means that the UAV needs to stand still for at least four seconds to classify which type of terrain it is flying over.

2.5 Spectral Information

As one of the most used features in the image processing field, this section will cover algorithms for classifying terrain using the terrain spectral information.

Water-terrain types have non-obvious geometric features, are sensitive to the surrounding environment such as illumination, reflection and appearance of clouds in the middle of a terrain classification, and thus it is difficult to obtain a classification using common image segmentation methods. As researched and studied in (Ebadi and Norouzi, 2017; Gracia et al., 2020; Matthies et al., 2005; Mora et al., 2017; Rankin and Matthies, 2010; Rankin et al., 2014; Salvado, 2018; Yao et al., 2007) in daytime, water regions have a higher brightness and weaker texture than their surroundings when viewed through RGB cameras (Ebadi and Norouzi, 2017; Matthies et al., 2005; Rankin and Matthies, 2010; Rankin et al., 2014; Yao et al., 2007) and lower brightness in NIR and RedEdge bands (Salvado, 2018).

In (Yao et al., 2007), an algorithm was proposed using a stereo-vision-based method to obtain the disparities from the stereo images, allowing the 3D points recovery to be acquired, including the elevation and distance information (Figure 2.36). In the height images, the reflection regions often present negative height, and in the distance images there will be sudden changes in the disparity value between the reflection and the surrounding regions. These are significant clues to eliminate water-reflection regions.

According to the autors of (Rankin and Matthies, 2010) and (Rankin et al., 2014), the horizon line is a useful cue for water detection for two primary reasons. First, knowing where the horizon line stands in a certain image limits the search for water bodies to the area below that line, decreasing the computational costs of such detection, and increasing the possibilities of success. Also, knowing the horizon line delimitates the search area for sky detection. The color of the sky is a strong cue for water bodies in wide-open areas, and knowing the location of the sky in color imagery is a critical component of water detection based on sky-reflection methods.

The method proposed in (Rankin et al., 2014) divides the detector into three different steps leading towards the final decision. First there is an estimate of where the horizon is, secondly the method tries to search above the horizon line and detect the sky, and thirdly, through the detected sky pixels it tries to determine if the ground pixels they are reflecting on have a similar color and terrain features. Figure 2.37 shows the output result of this

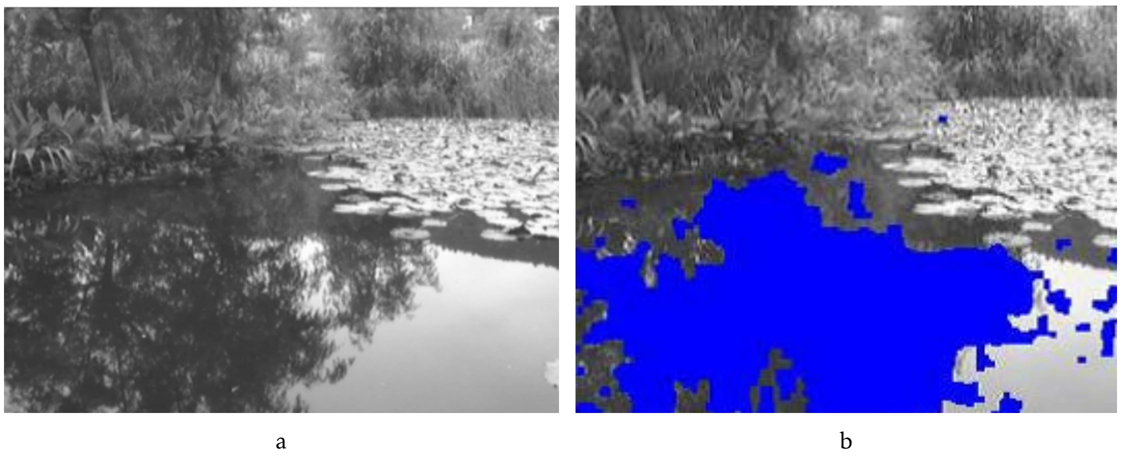


Figure 2.36: Stereo-vision-based reflection extraction (Yao et al., 2007).

algorithm. This technique can lead to miscalculation if the weather conditions/light conditions are not optimal for such a task, or if the water environments can be confused with the nearby land, for example in a swamp environment.

Other algorithms use color information to classify terrains, such as the one presented in (Ebadi and Norouzi, 2017), which is able to distinguish four different terrain types within an image. During this process, each channel's pixel is divided by the square root of its own three channels intensity, as can be seen in Figure 2.38.

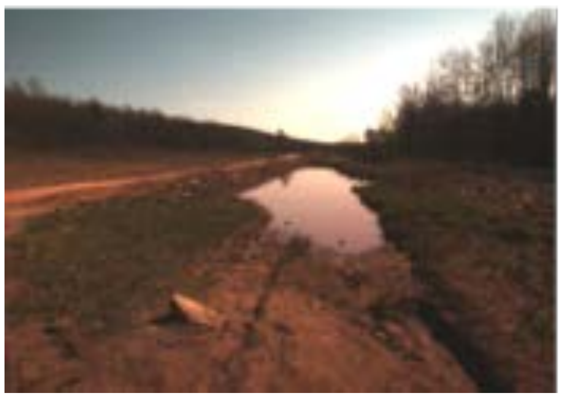
The final result will emphasize the color that most represents the terrain type (eg, blue for water). Next, a Neural Network (NN) will classify the terrain under study. However, even if the RGB channels are normalized, the system is still unstable both as a result of climate changes and during the course of the day.

In addition to the visible band (RGB), there are also bands with lower frequencies that are used in terrain classification. The cameras that use these bands are called multispectral cameras, as shown in Figure 2.39.

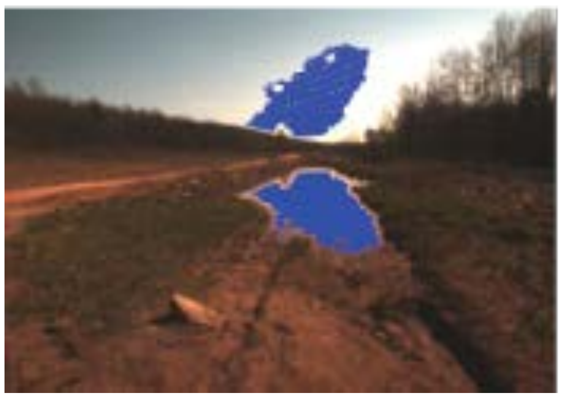
From the images obtained by the camera presented in Figure 2.39, the author, after aligning the lenses from the red band, used five multispectral indices (NDVI, ENDVI RDVI, MSAVI and SR) to classify four different terrain types: water, vegetation, sand and rocks (Salvado, 2018). However, despite the good accuracy of the system proposed by the author, detecting various types of terrain, when the height is higher than 60m, it can lose resolution when classified. Another disadvantage is that shadows can be confused with water-type terrains, due to the fact that water has low values in the NIR and Red-Edge bands.

2.6 Deep Learning

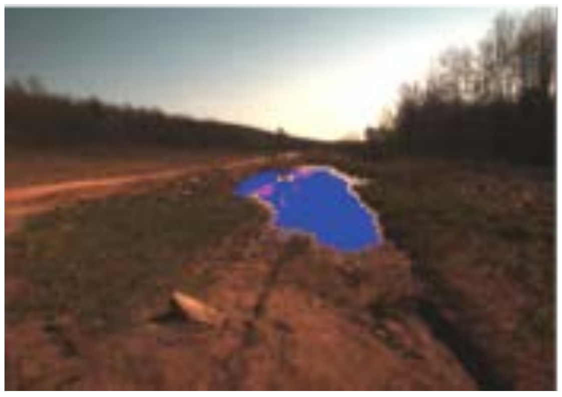
The term Deep Learning (DL) was introduced to the Machine Learning (ML) community in (Dechter, 1986; Schmidhuber, 2015) and to artificial NN in the context of "Boolean



a

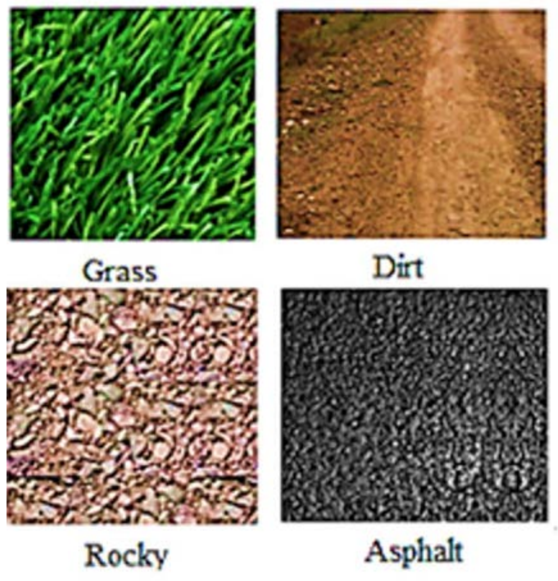


b

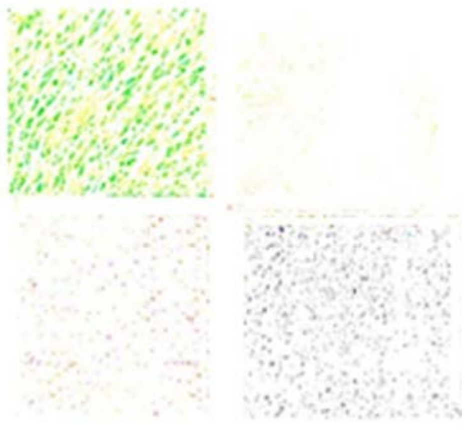


c

Figure 2.37: Example of water being directly detected by detecting the sky (Rankin et al., 2014).



a



b

Figure 2.38: Color normalization process (Ebadi and Norouzi, 2017): a) Raw data; b) RGB normalization result.

Figure 2.39: Multispectral camera: RedEdge-M (Salvado, 2018).

threshold neurons" (Chen et al., 2001; Gomez and Schmidhuber, 2005). Thus, DL is part of a broader family of ML methods, based on artificial neural networks (LeCun et al., 2015; Schmidhuber, 2015).

The work on DL (and in particular Convolutional Neural Networks (CNNs)) has recently been used in the field of computer vision. These networks are biologically inspired and trained with powerful algorithms.

As shown in Figure 2.40, the architecture of a typical CNN model is structured as a series of layers. Each layer performs multiple transformations using a bank of convolutional kernels (filters) (Lecun et al., 2010).

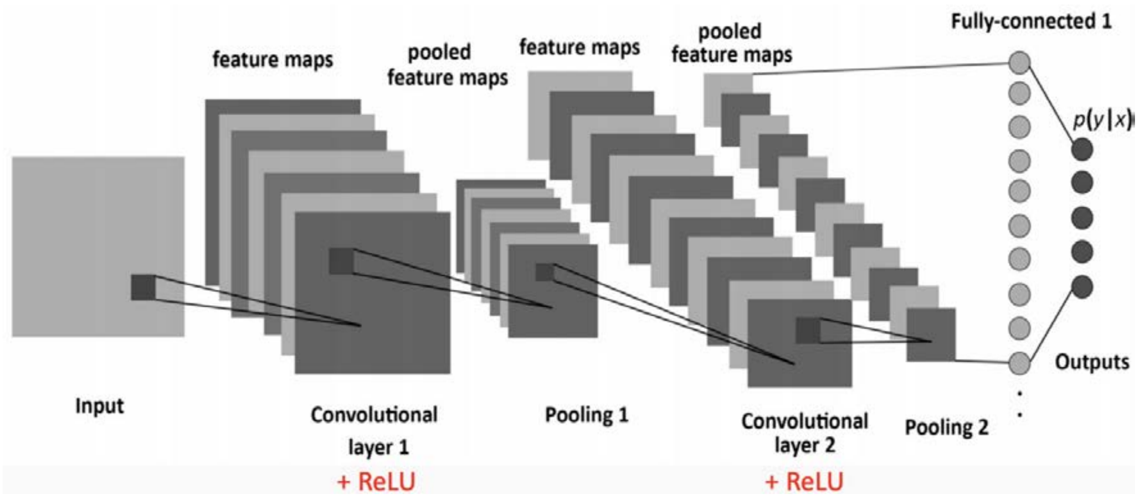


Figure 2.40: The architecture of a standard Convolutional Neural Network model (*Simple Introduction to Convolutional Neural Networks.*).

Convolutional layers convolve kernels with a small span over the entire input image area (Horn et al., 2017). Figure 2.41 shows an example of a convolution between a given input image with a kernel.

In addition to the convolusion presented in Figure 2.41, it is important to explain that to reduce the computational complexity, pooling steps are applied in order to decrease the size of the layers, as can be seen in Figure 2.40. It is also possible to observe in Figure 2.40, in each convolution, a step known as Rectified Linear Unit (ReLU) (one of the most used activation functions) as shown in equation 2.47.

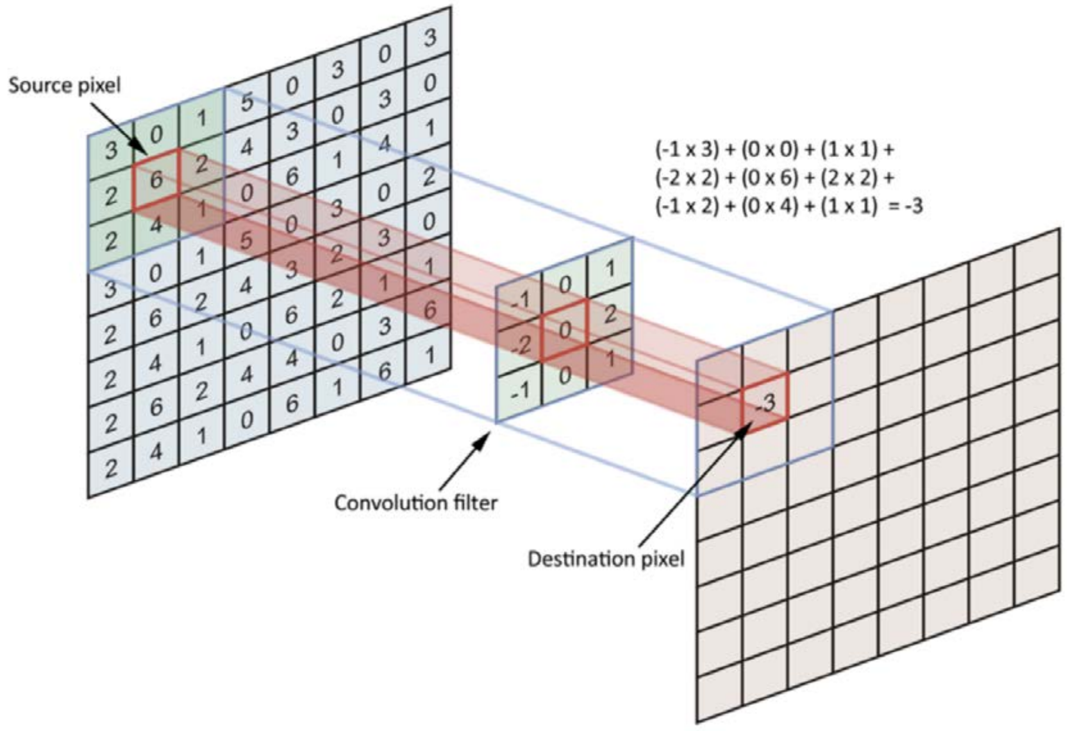

Source pixel
(-1 x 3) + (0 x 0) + (1 x 1) +
(-2 x 2) + (0 x 6) + (2 x 2) +
(-1 x 2) + (0 x 4) + (1 x 1) = -3
Convolution filter
Destination pixel

Figure 2.41: Convolutional layer destination feature value calculation example (*Simple Introduction to Convolutional Neural Networks.*).

$$f(z) = max(0, z) \tag{2.47}$$

This function works as a linear function for values greater than 1 and works as a non-linear function for negative values.

Some of the works studied in this dissertation regarding terrain classification, (Otte et al., 2015; Shen and Kelly, 2017; Özuysal, 2018) demonstrated excellent feature-representation capability. In (Shen and Kelly, 2017) the author, with a monocular vision camera, developed a CNN to classify six different terrains types as shown in Figure 2.42. From Figure 2.42, the author applied the CNNs developed, and the output is shown in Figure 2.43 with an overall accuracy of 86.25%. The author also compares the current work with an SVM and proved that for that specific case, CNNs give better results than SVM classifiers.

Another work studied is presented in (Otte et al., 2015). With a VGA camera mounted on a USV moving in an outdoor environment (Figure 2.44 a)), the author studied four different terrain types: asphalt; cobblestones; grass/vegetation and gravel. These terrain types can be observed in Figure 2.44 a).

Instead of using a CNN, the author in (Otte et al., 2015) chose to study three different types of Recurrent Neural Networks (RNN). Unlike CNNs, RNNs are able to store information and use it in the next data inputs in order to make predictions. Thus, of the three RNN architectures studied, the author concluded that RNN and Dynamic Cortex Memory networks (DCM) had better results, at 83.49% and 83.40% respectively.

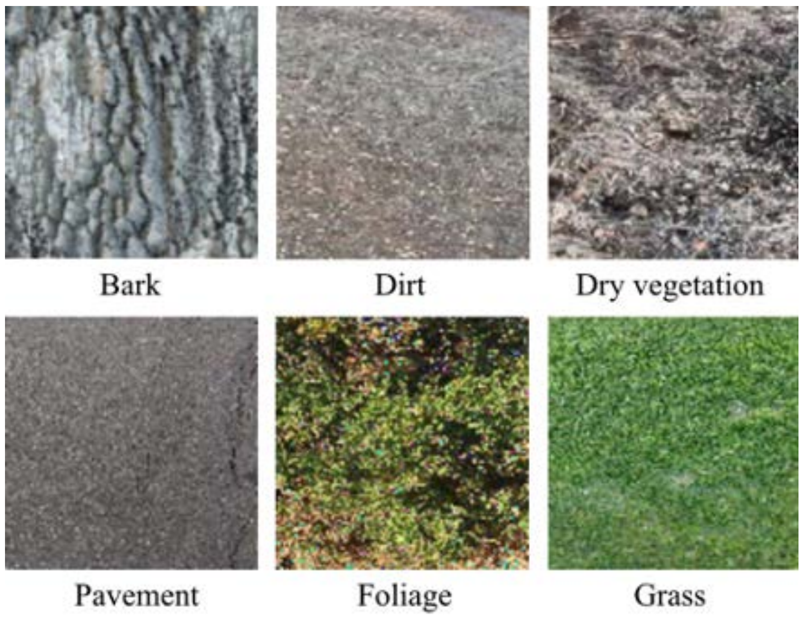


Figure 2.42: Samples from the six classes of terrains and textures with a size of 100x100. (Shen and Kelly, 2017).

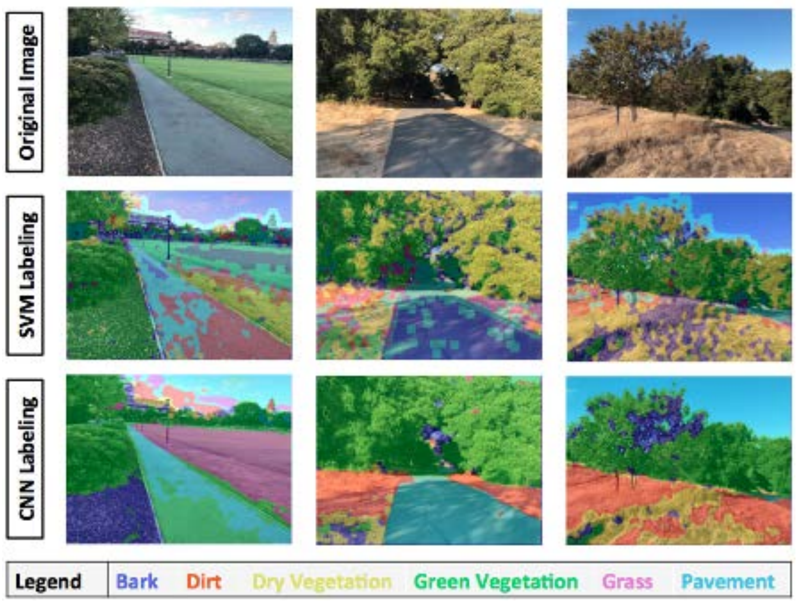


Figure 2.43: Multi-terrain classification comparing SVM and CNN classifiers (Shen and Kelly, 2017).

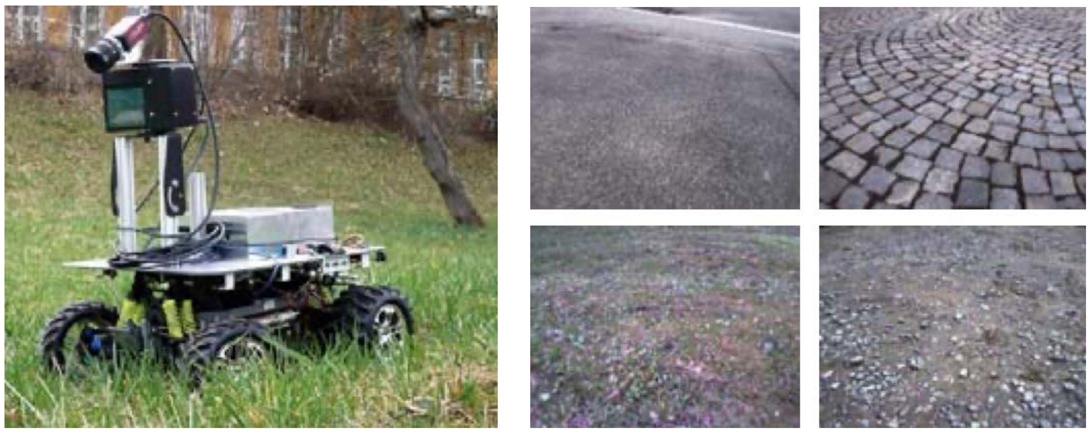


Figure 2.44: a) The USV used for ground image acquisition; b) The four terrain classes used in the ground dataset (Otte et al., 2015).

In other fields of application, CNNs were used for the recognition of malignant lymphomas based on gender, diet and age (Andrearczyk, 2017).

2.7 Light Detection and Ranging

Although in this dissertation RGB cameras are used to capture 2D images and classify them as water or non-water terrain type, it is also necessary to present an important sensor for 3D classification: Light Detection and Ranging (LiDAR).

LiDAR, or Light Detection and Ranging (or, laser range finder) is one of the most popular remote sensing technologies being used nowadays since it allows one to collect information with high precision and more rapidly compared to other technologies of the same type (González et al., 2008) such as RADAR, which has been used on aerial vehicles with the purpose of measuring distance to the ground (altitude), (Ramasamy et al., 2016). Figure 2.45 is an example of a LiDAR.

Figure 2.45: Hokuyo UTM 30LX-EW Laser Scanner. Courtesy of (Koch et al., 2017)

LiDAR's functioning principle is the following: light pulses are transmitted by the sensor and are scattered/reflected back, when they strike the surface of an obstacle. Then, based on the transmitted pulses' time of travel, the distance to the object is calculated:

$$d = \frac{\delta t \cdot c}{2} \tag{2.48}$$

Where δt is the elapsed time and c the speed of light.

Additionally, it is possible for LiDAR technology to be used in both indoor and outdoor environments and under difficult weather conditions such as fog, or during the night, with accurate depth precision. Nevertheless it works better during daylight and clear sky conditions. A solution such as RADAR would be preferable in this respect.

To sum up, LiDAR is well suited for short/medium distance obstacle detection on unmanned vehicles (Halterman and Bruch, 2010), by enabling the creation of a 3D structure dataset without causing any damage to crops (R. Rosell et al., 2009). However the biggest drawback is its price. These types of sensor are extremely expensive and this was one of reason why in this dissertation RGB cameras were used instead of LiDAR.

Algorithms that use laser scanners proved to be suitable for accurately distinguishing between water and non-water terrains (Gruszczyński et al., 2017; Sofman et al., 2006; Wallace et al., 2016; Yan et al., 2015).

According to (Sofman et al., 2006), from the LiDAR, the author could take the 3D information (Figure 2.46 a)) and train with a small set of labeled data to classify and help the ground vehicles in autonomous navigation (Figures 2.46 b) and c)).

However, and to meet the goal of this dissertation, the author might not be able to detect shallow water terrain from LiDAR because the shallow water terrains increase the decision error due to laser reflection, which leads to a misclassification as non-water terrain.

2.8 Summary Related Work - Terrain Classification

Table 2.3 shows an overview of all the TC algorithms studied in the course of this dissertation. The green colors indicate the best performances for specific proposed algorithms.

Although prior research work has proposed many good solutions for terrain classification, there is still a gap regarding the study of dynamic terrain. The previously mentioned algorithms suffer from a high sensitivity to changes in the environment, mainly due to changes in brightness, color and texture. The use of the downwash effect will improve these limitations, as will be explained in Chapter 3.

Table 2.3: Terrain Classification Method - Accuracy Comparison.

Ref. and Method	Water (%)	Vegetation (%)	Sand (%)	Asphalt	Rock (%)	Overall (%)
(Tong et al., 2018) - GLCM	57.18	51.94	-	-	-	-
(Tong et al., 2018) - LBP	80.62	71.21	-	-	-	-
(Caridade et al., 2008) - GLCM	86.80	89.90	-	-	-	81.90
(Mboga et al., 2017) - GLCM	-	-	-	-	-	86.60
(Mboga et al., 2017) - LBP	-	-	-	-	-	90.48
(Woods et al., 2013) - Depth Texture Features	-	93.00	93.40	88.90	90.60	92.30
(Sofman et al., 2006) - LiDAR	-	66.73	82.79	-	-	75.55
(Salvado, 2018) - Multispectral Camera	89.13	82.93	84.54	89.59	-	80.00
(Wang et al., 2019) - Hyperspectral Camera	-	79.84	-	79.05	-	81.17
(Bai et al., 2019) - Vibration	-	82.14	78.57	85.71	78.57	-
(Ebadi and Norouzi, 2017) - Normalized Color	-	76.00	-	68.00	61.00	67.00
(Ghosh and Sharma, 2015) - HOG	-	-	-	-	-	86.54
(Zhao et al., 2016) - Voronoi	-	-	-	-	-	82.30
(Khan et al., 2011) - LBP	-	-	-	-	-	96.90
(Khan et al., 2011) - LTP	-	-	-	-	-	98.10
(Khan et al., 2011) - LATP	-	-	-	-	-	97.20
(Ma et al., 2017) - Gabor	-	-	-	-	-	96.11
(Pombeiro et al., 2015) - Optical Flow	87.00	-	-	-	-	87.00
(Shen and Kelly, 2017) - CNNs	-	-	-	-	-	86.25
(Otte et al., 2015) - RNN	-	-	-	-	-	83.49

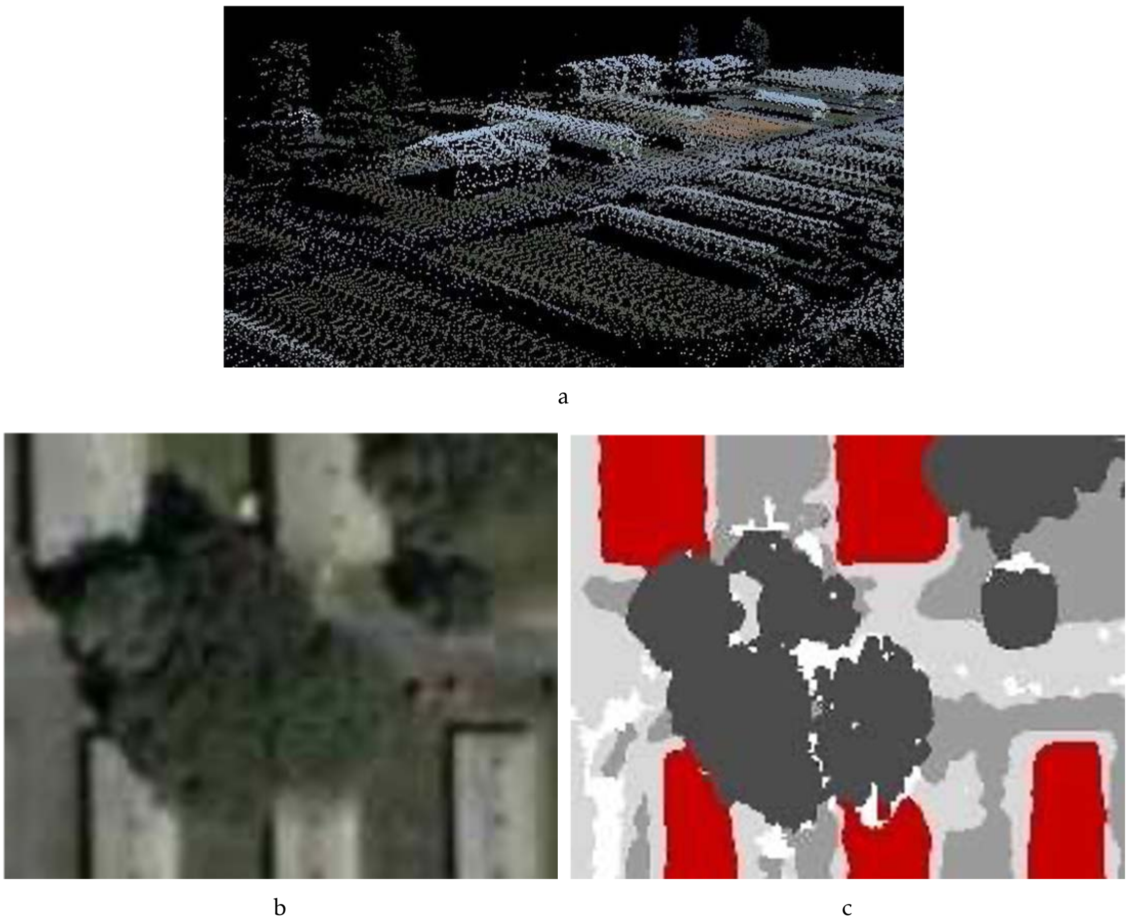


Figure 2.46: Results with LiDAR in (Sofman et al., 2006): a) Raw 3D data; b) Terrains under study (road, grass, trees and buildings); c) Output result.

Chapter 3

Methodology

The purpose of this chapter is to depict all of the details related to the solutions found throughout this dissertation. These solutions aim to classify water terrain type in two ways: Static Feature Extraction; Dynamic Feature Extraction.

Some of specificities of the aforementioned solutions are related to the UAV localization (by fusing several of its sensors); to how to fuse the terrain classification information with mapping; and how to use the latter to provide, for example, the type of terrain where the UAV is flying over or an emergency landing request from the high-level module.

Thus, this chapter will be divided into five main sections:

- Section 3.1 introduces the experimental setup in use, detailing the environment where the proposed algorithms are studied;
- Section 3.2 explains the Static Feature Extraction algorithms for Terrain Classification;
- Section 3.3 presents the Dynamic Feature Extraction algorithms for Terrain Classification;
- Section 3.4 describes the importance of using GPU capabilities to improve the proposed algorithms processing time. In this dissertation, only GLCM was used with GPU resources;
- Section 3.5 explains how the RGB images obtained by a camera are placed on a georeferenced dynamic map.

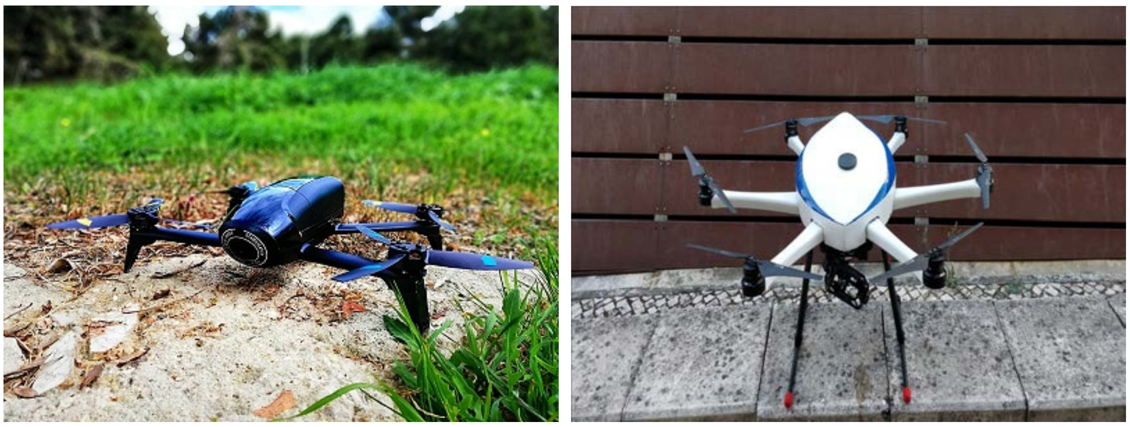
The overall system goal is to improve autonomous navigation and cooperation between robots.

3.1 Experimental Setup

This section will detail the material used in this dissertation namely the UAV, RGB cameras and their calibration, and the types of terrain under study in this thesis. Two aerial vehicle were used in the experiments:

- Parrot Bebop2 (Figure 3.1 a)), is a quad-rotor solution that endows the system with graceful degradation, as it is able to land with one motor off, although without yaw control in that condition.

- HEIFU is a custom made solution, aimed at the agricultural sector, developed through a collaboration between Beyond Vision (BEV) and Projeto Desenvolvimento Manutenção Formação e Consultadoria (PDMFC). The HEIFU is equipped with an onboard computer running Ubuntu and ROS. This is an open platform that allows the integration of different inputs and is used to run multiple tasks, such as image processing, data relaying, remote control of a drone (and more). HEIFU can be used with different communication systems, such as a mobile network or Wi-Fi connection. HEIFU is a Hexacopter, as can be seen in Figure 3.1 b), with a diagonal wheelbase of 1.4m. The drone weight is around 6.2kg, including battery, and the hovering time is about 38min with a battery of 16Ah.



a

b

Figure 3.1: UAVs used in this dissertation: Parrot Bebop2 (a); HEIFU (b).

Some of the Bebop2 and HEIFU specifications are:

- Ardupilot (Pixhawk) hardware is used to control low level operation. This hardware contains the IMU and GPS to provide the UAV's position and orientation. Also, the Pixhawk connects to the Jetson nano embedded system via a MAVlink protocol and the UAV's battery. Lastly, to control the UAV's motors it is connected to a UHF receiver;

- An Jetson nano is used to control the high level operation. It receives data from the distance sensor (depth cameras), the RGB camera, and communicates with external devices via a Wifi link;

- An RGB camera with a gimbal stabilizer is installed capturing onboard images at a resolution of 640x480 pixels;

- The camera and lens specifications are known, allowing the field of view (FOV) and the pixel size in meters to be computed.

For a better understanding of the communication between various system layers, Figure 3.2 represents the communication layers.

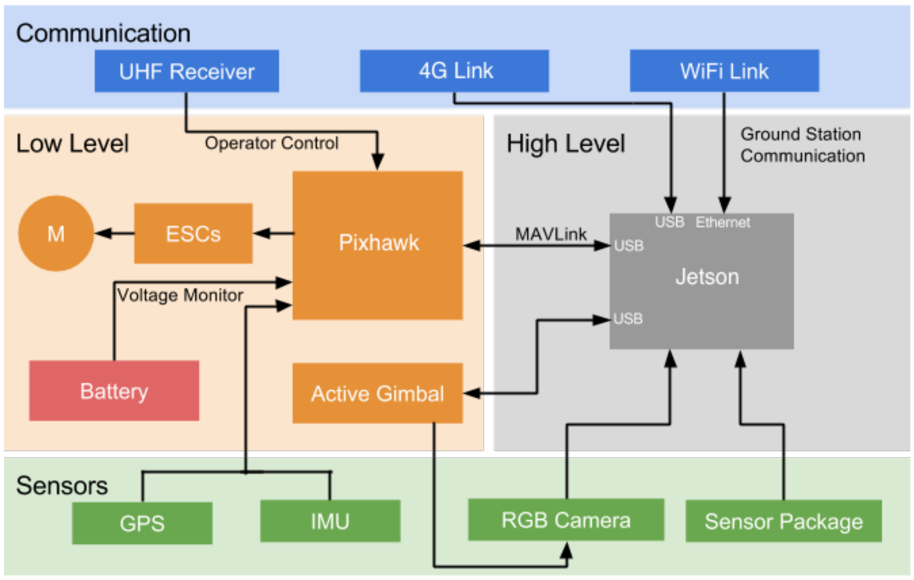


Figure 3.2: High-level communication layers.

3.1.1 Perception Sensors

The first step of this thesis, before starting algorithm development, was to identify the sensors that were available to acquire information about the surrounding environment. There are sensors, in this case RGB cameras, that have been used for the purpose of terrain classification. However, it was necessary to calibrate the RGB cameras so that the algorithms proposed in this dissertation work for any type of camera.

An RGB camera can be used to obtain various data from a given environment, such as length, distance, size and other elements. For this, it is necessary to know the ccamera's calibration parameters (Dias, 2015).

As seen in Figure 3.3, the camera used has a distortion known as fish-eye. Such distortion is caused by the wide Field Of View (FOV) (allows a greater opening), however the

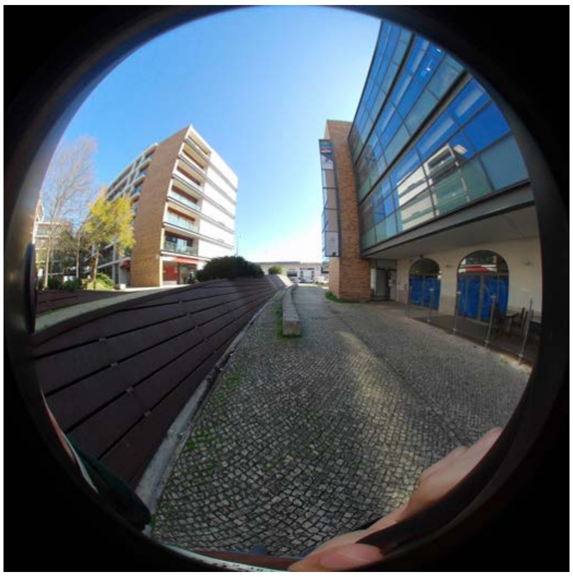


Figure 3.3: Raw data - Fish-eye effect.

size of the objects are not coherent. Therefore, the camera was submitted to a calibration process[1] using a tool available in ROS.

The calibration uses a table that adheres to the method proposed in (Ntouskos et al., 2007), where a planar chessboard pattern with known dimensions is presented. The planar chessboard is presented at different angles and positions for computing the relationship between the distances between each point as can be seen in Figure 3.4.

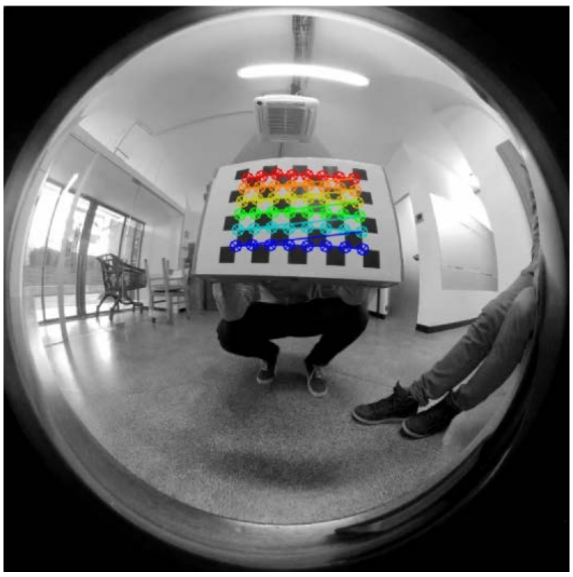


Figure 3.4: Camera calibration using the (Ntouskos et al., 2007) method.

After calibration, a file is generated containing the camera parameters, which can be used to remove distortion and compute the images to obtain scene data. The image

[1]`http://wiki.ros.org/camera_calibration`

without the fish-eye distortion through the collected parameters can be seen in Figure 3.5.

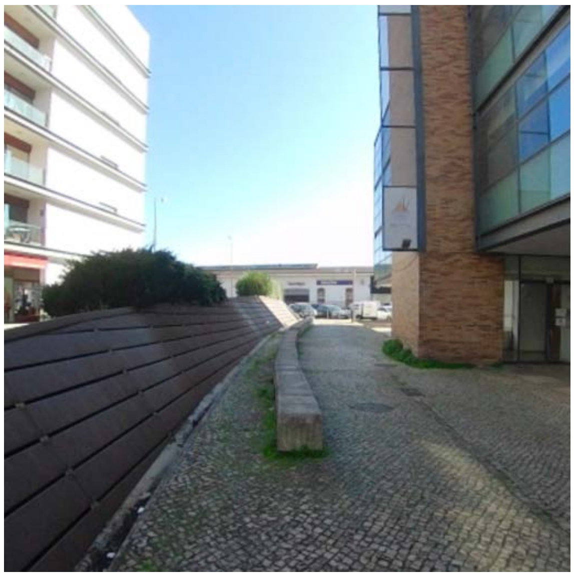


Figure 3.5: Camera calibration result.

3.1.2 Terrain Types

As mentioned in Section 1.3 the dynamism of different terrains when exposed to wind provokes singular texture patterns that can be used in their identification. In this thesis, for terrain classification, the importance of static and dynamic image features is studied, such as colour and texture and the OF when exhibited by the downwash effect. Although several terrain types were studied, the obtained dataset is mainly divided in two different groups: Water (pool and lake) and Non-Water (vegetation, sand and asphalt).

As seen in Figure 3.6, in this dissertation four different terrain types (water, vegetation, asphalt and sand) were studied together with the study of downwash effect impact caused by UAVs in order to classify water or non-water terrain types. The data of the water-type terrain were taken in pools, lakes, and seas in order to have more reliable results in a greater variety of data.

3.1.3 System Specifications

A proposal for the system's architecture is depicted in Figure 3.7. As mentioned before, a Jetson nano was used with several algorithms for terrain classification.

The Jetson computational speed is affected by using some terrain classification algorithms such as the travel distance (Section 3.3.1) and circular motion (Section 3.3.2) algorithms. Thus, a ground station was used to visualize the data and run these heavy algorithms. This task is facilitated by the fact that ROS already contains tools that allow multiple machines to communicate remotely.

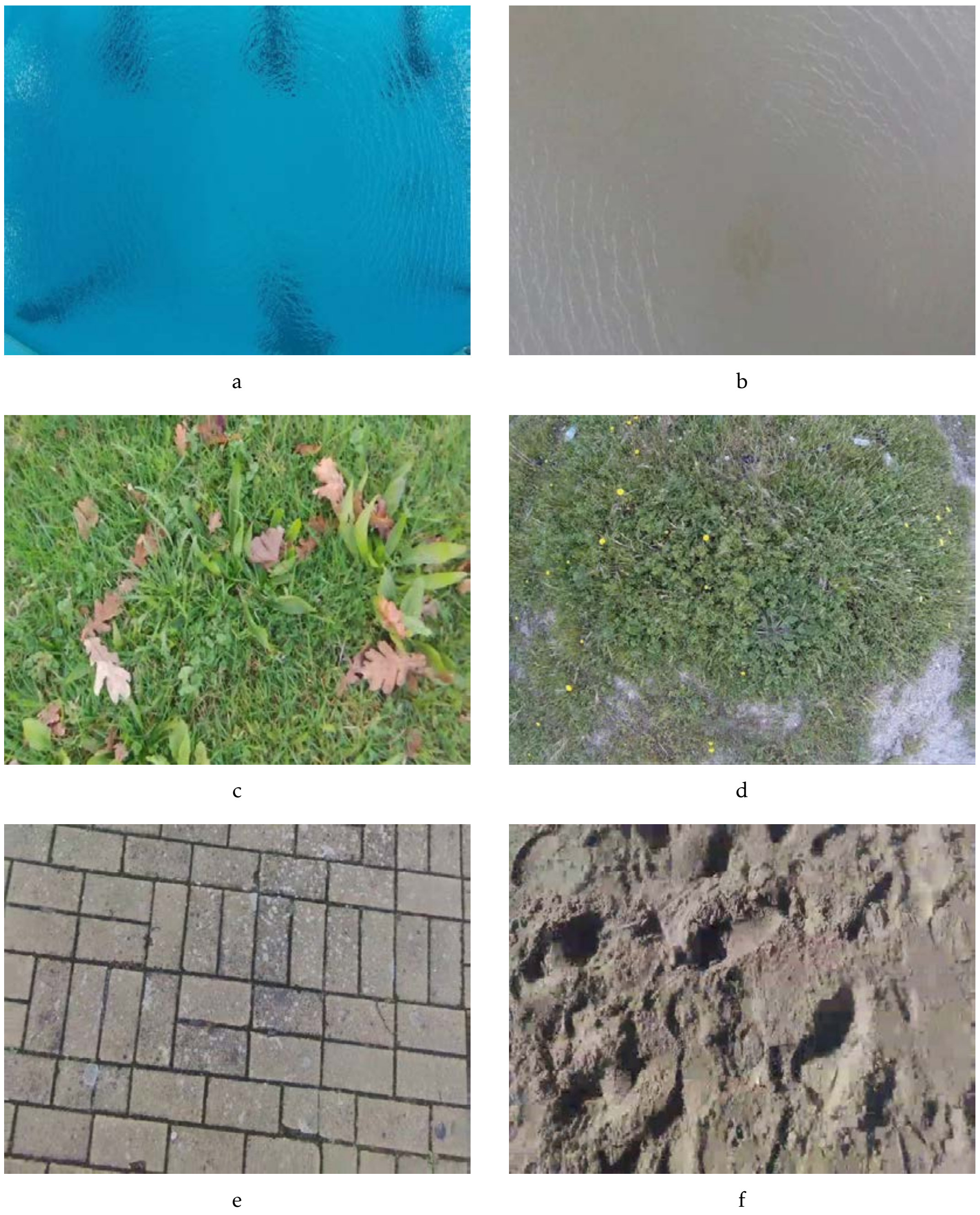


Figure 3.6: Examples of terrain types: water (a)(b); vegetation (c)(d); and sand (e)(f).

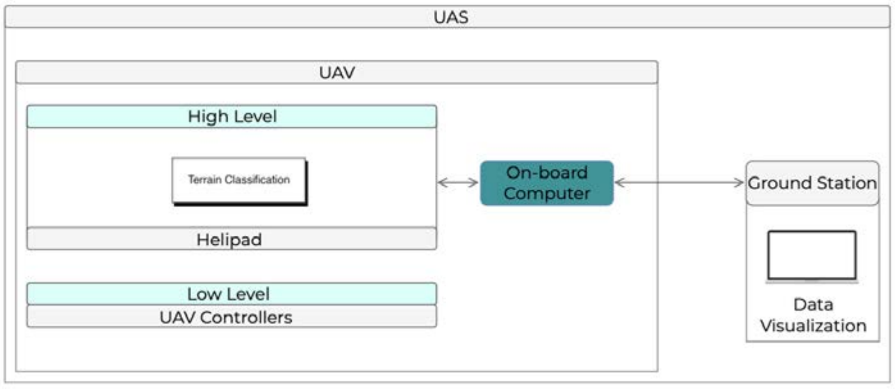


Figure 3.7: Overall Architecture of the System.

The "on-board" computer that was used in this thesis has the following software and hardware specifications:

- Ubuntu 18.04 LTS (64-bit version);
- Intel Core i7-8510U CPU (@ 2.00 GHz × 4);
- 16Gb of RAM;
- GeForce GTX 1060.

Finally, both the on-board computer and the ground station have ROS installed, which are explained in Appendix B. This allowed the system to be built in a modular way, which, according to Lacaze et al. in (Lacaze et al., 2002), has advantages such as less computational effort, easier representation and the option to run different modules at different speeds, according to the requirements.

3.2 Static Feature Extraction for Terrain Classification

The ability to precisely classify different types of terrain is useful for Unmanned Aerial Vehicles (UAVs). As mentioned before, there are multiple situations in which terrain classification is fundamental for achieving UAV mission success, such as emergency landing, aerial mapping, decision making and cooperation between robots in autonomous navigation.

The main goal of this section is to propose a computer vision algorithm that using RGB images captured by a camera onboard a UAV, is capable of classifying a terrain by analysing static image features (colour and texture) and the rotor-downwash effect on the underlying surface. There are several issues that must be addressed in order to achieve this goal, namely:

- Which algorithms are more accurate for classifying water terrain types using the downwash effect?

- What are the texture and motion patterns of each terrain (water movement for example)?
- Which static and dynamic image features can be extracted to classify the terrain?

To address these challenges, new texture feature extraction algorithms will be proposed in the following sub-sections, aiming at the best possible performance.

3.2.1 Gabor with Lowess Regression

As mentioned in Section 1.3, the downwash effect caused by UAV propellers, shown in Figure 1.4, might have a major impact on terrain classification, because each terrain, when subject to this effect, presents different behaviors. Take the example of water-type terrains, where no color variation exist, i.e, no motion is presented, these are wrongly classified by algorithms designed to evaluate static textures. This happens because in water it is not possible to observe any texture under these conditions. Using the downwash effect, motion is added to the environment, and, for example in water type terrains, circular textures can be observed.

The first algorithm presented in this section will be a conjunction between:

1. Gabor Filter;
2. Width Projection;
3. Lowess Regression.

To better understand these steps, Figure 3.8 presents the diagram for the proposed method.

Five main processes have been identified in the model, namely:

- **Rectified Image:** For the proposed classifier to be generic enough to work for all RGB cameras (depending on the resolution), it is necessary first to calibrate the image and mitigate its distortion. Section 3.1.1 explains how to calibrate these images;
- **Texture Filter:** Extracts the terrain's static textural information using Gabor filters;
- **Threshold:** A thresholding is applied to the static texture image (gabor filter) to highlight terrain roughness;
- **Projections:** In conjunction with Lowess Regression, width projection was applied to the thresholded image, extracting unique features that help differentiate the different types of terrains;

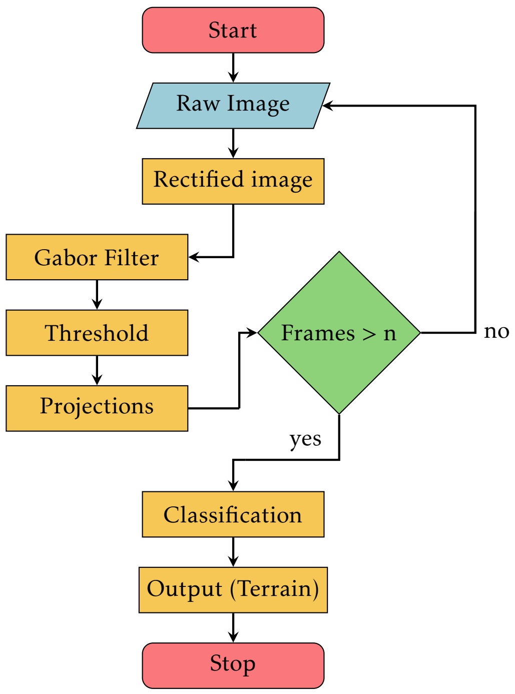


Figure 3.8: Proposed diagram for the Gabor and Lowess method.

Listing 3.1: Gabor filter implementation.

```
Image Gabor(img, kernelSize, sigma, theta, lambda, gamma, psi){
    // Create the Kernel
    kernel = getGaborKernel(kernelSize, sigma, theta, lambda, gamma, psi);

    // Convolution calculation
    gaborImage = filter2D(img, kernel);

    return gaborImage;
}
```

- **Classification:** To increase certainty and automate the classification of the type of terrain, a machine learning technique was used, namely a feed-forward neural network (NN). The architecture of the designed neural network, was composed of two layers, a hidden layer with 10 neurons and an output layer with two neurons (water, non-water). A sigmoidal function was used as an activation function and the final classification was derived from the output neuron with highest activation value. In this section, 70% of the total data were for training, 15% for testing and 15% for validation.

Starting with the Gabor filter, as explained in Section 2.3.2, this filter is capable of extracting a texture in any direction (by changing the filter orientation). Thus, Listing 3.1 shows how this filter was implemented.

The following were used as input parameters:

- **Kernel Size:** 4;
- **Lambda** (λ): 5;
- **Theta (θ):** 0 and 90 degrees;
- **Psi (ψ):** 0;
- **Sigma (σ):** 70;
- **Gamma (γ):** 0.

Thus, as can be seen in Figure 3.9, it is possible to obtain, for example, the texture of a water-type terrain when it is affected by the downwash effect of the UAV using the Gabor filter at two different angles (with θ equals to 0 and 90 degrees). Next, in order to emphasize the water-terrain texture, a binarization (Listing 3.2) with a given threshold was used (Figure 3.9 d)), with $threshold\_value = 25$.

Listing 3.2: Gabor Binarization.

```
// Gabor 0 and 90 degree Binarization
imgGabor0Binarized = Binarization(imgGaborTheta0, thresholdVale, 255, 1);
imgGabor90Binarized = Binarization(imgGaborTheta90, thresholdVale, 255, 1);

// Both matrices are added (Gabor with 0 and 90 degree) into a new Matrix - ↩
    imageGaborTheta0And90
imageGaborTheta0And90 = addMatrix(imgGabor0Binarized, imgGabor90Binarized);
```

At this stage, the second point of this algorithm was performed: width projection. From Figure 3.9 d), the number of white pixels is counted the columns of each image to create an output vector from this projection, as can be seen, in Listing 3.3.

From the observed vertical projection of water-type terrain (Figure 3.10) it can be seen that it produces an ondulatory effect with a local minimum in the center of the downwash. This effect in water-type terrains is due to the lower roughness in the center of the downwash. However, due to the water movement, around the center greater roughness is observed (white pixels in the binarized image in Figure 3.9.d). The next step was to translate this observed feature into a computational model.

A non-parametric regression was used in order to calculate a curve that most closely approximates vertical projection points (green dots) in Figure 3.10. Lowess's theory calculates the closest line to a given set of points, through weights assigned to each point in a neighboring window. The number of points that this window can contain is defined by the user. An example is provided with the data-set presented in Table 3.1:

To calculate the new Y of each point of estimation, and starting with the first X value (0.5578196), where the window size equals seven the next steps are the following:

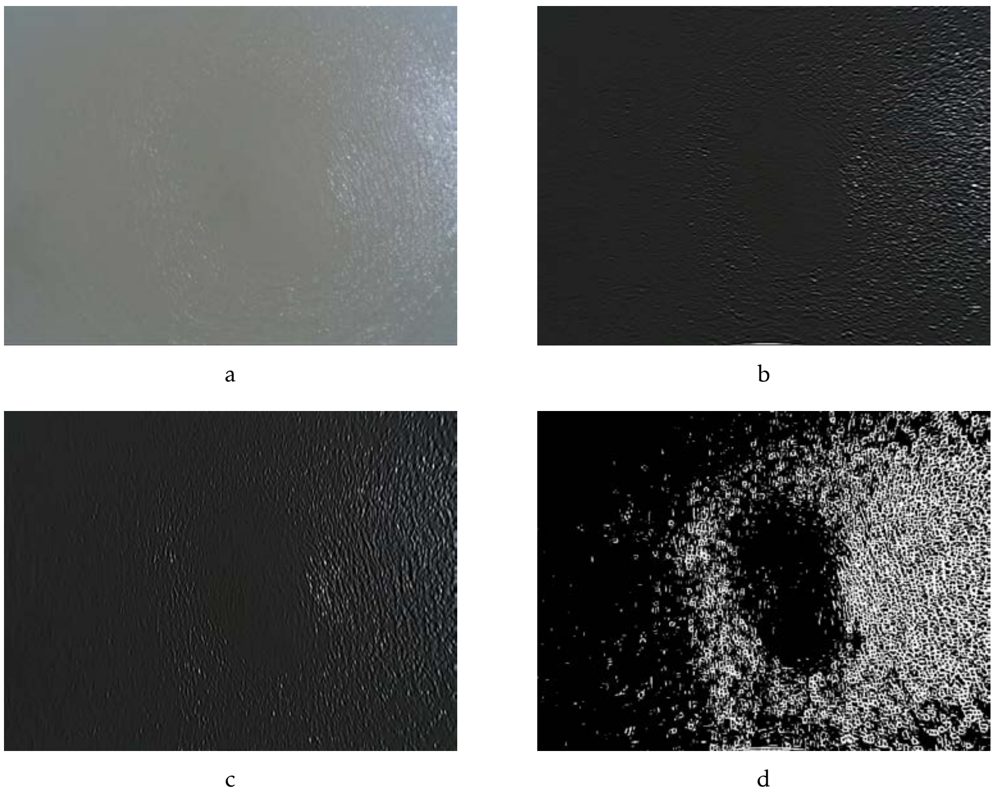


Figure 3.9: Example of static texture extraction: a) Raw image; b) c) Convolution with the Gabor filter θ-0 degrees (b) and θ-90 degrees (c); d) Sum of images b) and c) after thresholding.

Listing 3.3: Width Projection from Figure 3.9 d).

```
// Define the variables
colums = image.cols;
rows = image.rows;
histArray(colums);
pixelPtr = image.data;

// Loop to count how many white pixels (value 255) are in each image column
for (x = 0; x < colums; ++x) {
    for (y = 0; y < rows; ++y) {
        pixelValue = pixelPtr(y,x);
        f(pixelValue == 255)
            histArray[x]++;
}
```

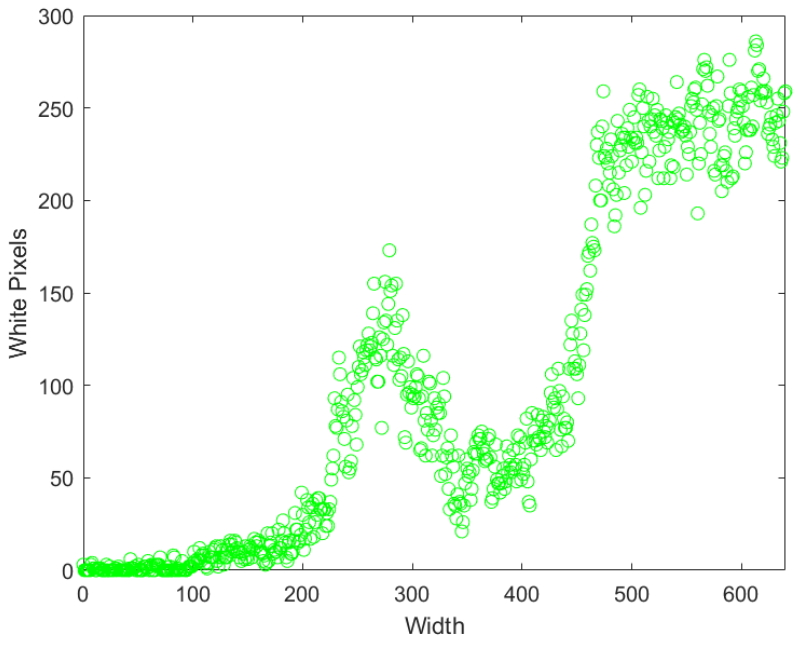


Figure 3.10: Width projection of the example in Figure 3.9.d.

Table 3.1: Example data-set.

	X	Y		X	Y		X	Y
1	0.5578196	18.63654	8	6.5411662	233.55387	15	13.2728619	152.61107
2	2.0217271	103.49646	9	6.7216176	234.55054	16	14.2767453	160.78742
3	2.5773252	150.35391	10	7.2600583	223.89225	17	15.3731026	168.55567
4	3.4140288	190.51031	11	8.1335874	227.68339	18	15.6476637	152.42658
5	4.3014084	208.70115	12	9.1224379	223.91982	19	18.5605355	221.70702
6	4.7448394	213.71135	13	11.9296663	168.01999	20	18.5866354	222.69040
7	5.1073781	228.49353	14	12.3797674	164.95750	21	18.7572812	243.18828

1. Calculate the distance on the X-axis between the point under study and its six closest neighbors (in this example the windows size is equal to seven as mentioned above):

Table 3.2: Local Subset of Data.

Point under Study	X	Y	X-axis Distance	X-axis Scaled Distance	Weight
0.5578196	0.5578196	18.63654	0.000000	0.0000000	1.0000000
	2.0217271	103.49646	1.463907	0.3217691	0.90334913
	2.5773252	150.35391	2.019506	0.4438904	0.75988974
	3.4140288	190.51031	2.856209	0.6277992	0.42621714
	4.3014084	208.70115	3.743589	0.8228466	0.08686171
	4.7448394	213.71135	4.187020	0.9203134	0.01072308
	5.1073781	228.49353	4.549558	1.0000000	0.0000000

2. Calculate the normalized distance using the X-axis distance column (Table 3.2), where each previously calculated distance is now divided by the greater distance (in this case the greater distance is 4.549558). Equation 3.1 translates this step into a mathematical formula. The scaled distance results are presented in the Scaled Distance column in Table 3.2.

$$NormDistance(i) = \frac{X - axis\ Distance}{max(X - axis\ Distance)},\ where\ i = 1...Windows\ size \tag{3.1}$$

3. Calculate the weight of each point. The weight function w is defined using equation 3.2 and the results are presented in the Weight column in Table 3.2.

$$w(d) = \begin{cases} (1 - |d|^3)^3 & for|d| < 1 \\ 0 & for|d| \geq 1 \end{cases} \tag{3.2}$$

It should be noted in Table 3.2 that the further the neighboring points are from the estimation point, the value of the weights will decrease so as to have less influence on the new estimated Y value.

4. In order to calculate the regression function, it is necessary to calculate the linear equation slope and intercept, by a weighted least squares fit of the data subset and previous weights, as shown in equation 3.3.

$$S(m,b) = \sum_{i=1}^{n} w_i \left((m \cdot x_i + b) - y_i\right)^2 \tag{3.3}$$

Where m is the slope, b is the intercept, w is the weight of each x and y value.

Since equation 3.3 represents the residual of each point (the difference between $m \cdot x + b$ and y), the goal of this function is to make the sum of the squares of these residuals as small as possible to find the minimum slope and intercept values. Thus, the minimum of each value is determined by calculating the partial derivatives of $S(m,b)$ with respect to m and b and equate the function to zero:

$$\begin{gathered} \frac{\partial S}{\partial m} = 0 \\ \frac{\partial S}{\partial b} = 0 \end{gathered} \tag{3.4}$$

From equation 3.4, the result of the equations system given by these two partial derivatives, for this example is:

$$\begin{gathered} \frac{\partial S}{\partial m} = m' = 59.03309 \\ \frac{\partial S}{\partial b} = b' = -12.33679 \end{gathered} \tag{3.5}$$

5. The last step is the calculation of new estimated Y value (regression function value). With the results given by equation 3.5, the new estimated Y value of the point under study is represent by the following equation (equation 3.6):

$$Regression\ Function\ Value = m' \cdot x + b' \tag{3.6}$$

These processes must be calculated for every point of Table 3.1. The result of using this locally regression can be observed in Figure 3.11, i.e. the red line. After the implementation of the red line, as show in Figure 3.11, local maxima and minima were calculated on this same line. The red line is represented by a $lineVector$ array.

The local minima and maxima are calculated as described in Listing 3.4.

Using the vector values obtained in Listing 3.4: maxLocal; maxLocalIndex; minLocal; and minLocalIndex, Figure 3.11 shows the output of these values.

The obtained local minima and maxima are used to obtain two features:

1. Valley area between the local minimum and its respective two local maxima, represented by the gray color in Figure 3.12;

Listing 3.4: Local Minima and Maxima calculation.

```cpp
void MaxMinHistogram(std::vector<int>& histArray){
    // Define the variables
    colums = lineVector.cols;
    tmpMax= lineVector[histrogramStep];
    tmpMin= lineVector[histrogramStep];

    // Check if the first point is a local minimum or local maximum
    if(histArray[0] > lineVector[histrogramStep]{
        maxLocal.add(lineVector[0]);
        maxLocalIndex.add(0);
    }
    else{
        minLocal.add(lineVector[0]);
        minLocalIndex.add(0);
    }

    // Save all Minima and Maxima locals into maxLocal and minLocal vectors,↩
         respectively
    for (i = histogramStep; i<colums-histogramStep; i=i+histogramStep) {

        if(tmpMax > lineVector[i+histogramStep] && tmpMax > lineVector[i-↩
            histogramStep]){
            maxLocal.add(tmpMax);
            maxLocalIndex.add(i);
            tmpMax = lineVector[i+histogramStep];
        }
        else
            tmpMax = lineVector[i+histogramStep];

        if(tmpMin < lineVector[i+histogramStep] && tmpMin < lineVector[i-↩
            histogramStep]){
            minLocal.add(tmpMin);
            minLocalIndex.add(i);
            tmpMin = lineVector[i+histogramStep];
        }
        else
            tmpMin = lineVector[i+histogramStep];
    }
}
```

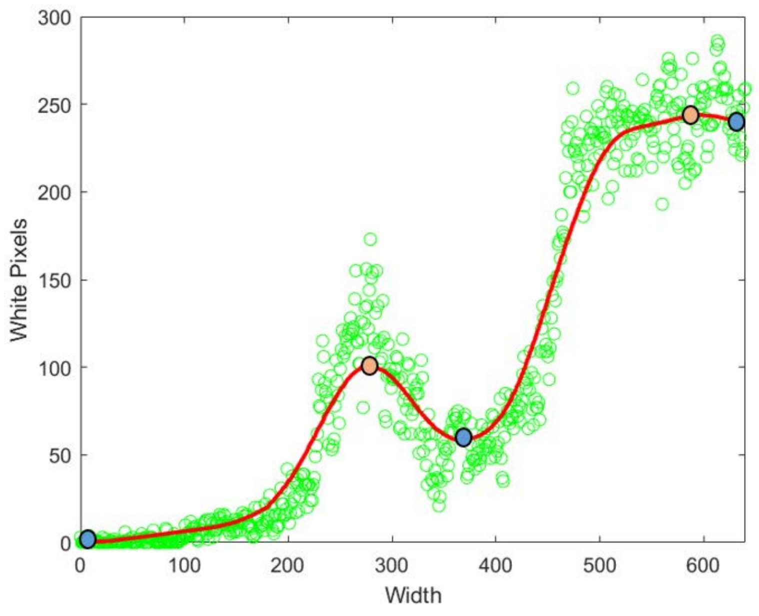


Figure 3.11: Width projection (green dots), Lowess regression fitting (red line), local minima (blue dot) and maxima (orange dot).

2. Integral between the local minimum and its two respective local maxima, represented by the yellow color in Figure 3.12.

The first has the advantage of being relative to minima and maxima values, while the integral gives an absolute value and will vary for the center and for increased roughness. Both of these data are sent to a neural network in order to classify the terrain type under study.

Now that the system has been explained, four different types of terrain (water, vegetation, sand and asphalt) are analysed in this section. Figures 3.13, 3.14, 3.15 and 3.16 describe different texture outputs from different types of terrain. It is possible to conclude that by implementing the Gabor filter, vegetation and asphalt terrains were found to have a higher texture than sand and water-type terrains. On the other hand, water-type terrain, also presents a singular characteristic due to the downwash effect provoked by the UAV, which can be decisive for differentiating it from other types of terrain.

3.2.2 Particle Swarm Optimization

One of the main shortcomings of the existing clustering methods is the need for users to specify threshold parameters, cluster amount, etc beforehand. Therefore, to avoid this issue, the second algorithm proposed in this dissertation, known as Particle Swarm Optimization (PSO), is a clustering algorithm that does not require parameters configuration, i.e. it eliminated human involvement.

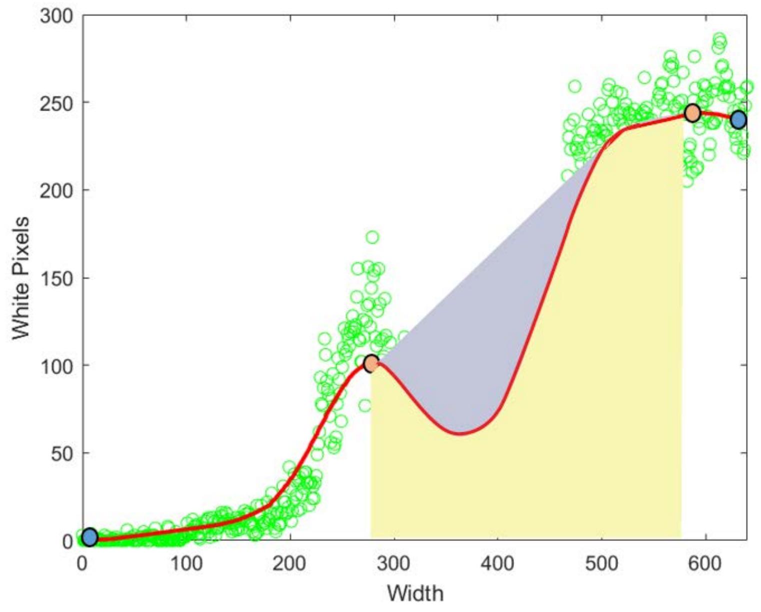


Figure 3.12: Width projection showing the valley area (gray color) in relation to integral (yellow).

The clustering method based on PSO mentioned earlier uses swarm particle group motion (pixel-by-pixel passage through the determined region in the image). Within the swarm, a search for the best solution, namely pixels with a maximum average intensity value, is conducted. Also, *k*-means that a clustering method (Oyelade et al., 2010) is included, which is intended to group pixels into a predetermined number of clusters by calculating the minimum value of the distance function. However, in comparison with the original *k*-means method, the need to specify the cluster size in advance by the user, is eliminated and, apart from the distance function minimization, an operation of color function minimization is added.

The initial normalization is needed in order to make an image insensitive to light changes and thus, simplify the classification process. Normalization works, according to equation 3.7:

$$(r', g', b') = \left(\frac{r}{\sqrt{r^2 + g^2 + b^2}}, \frac{g}{\sqrt{r^2 + g^2 + b^2}}, \frac{b}{\sqrt{r^2 + g^2 + b^2}} \right), \tag{3.7}$$

where:

- r, g, b are the initial values of a pixel's RGB vector;
- r', g', b' are the normalized values of a pixel's RGB vector.

The developed clustering algorithm consists of the following stages:

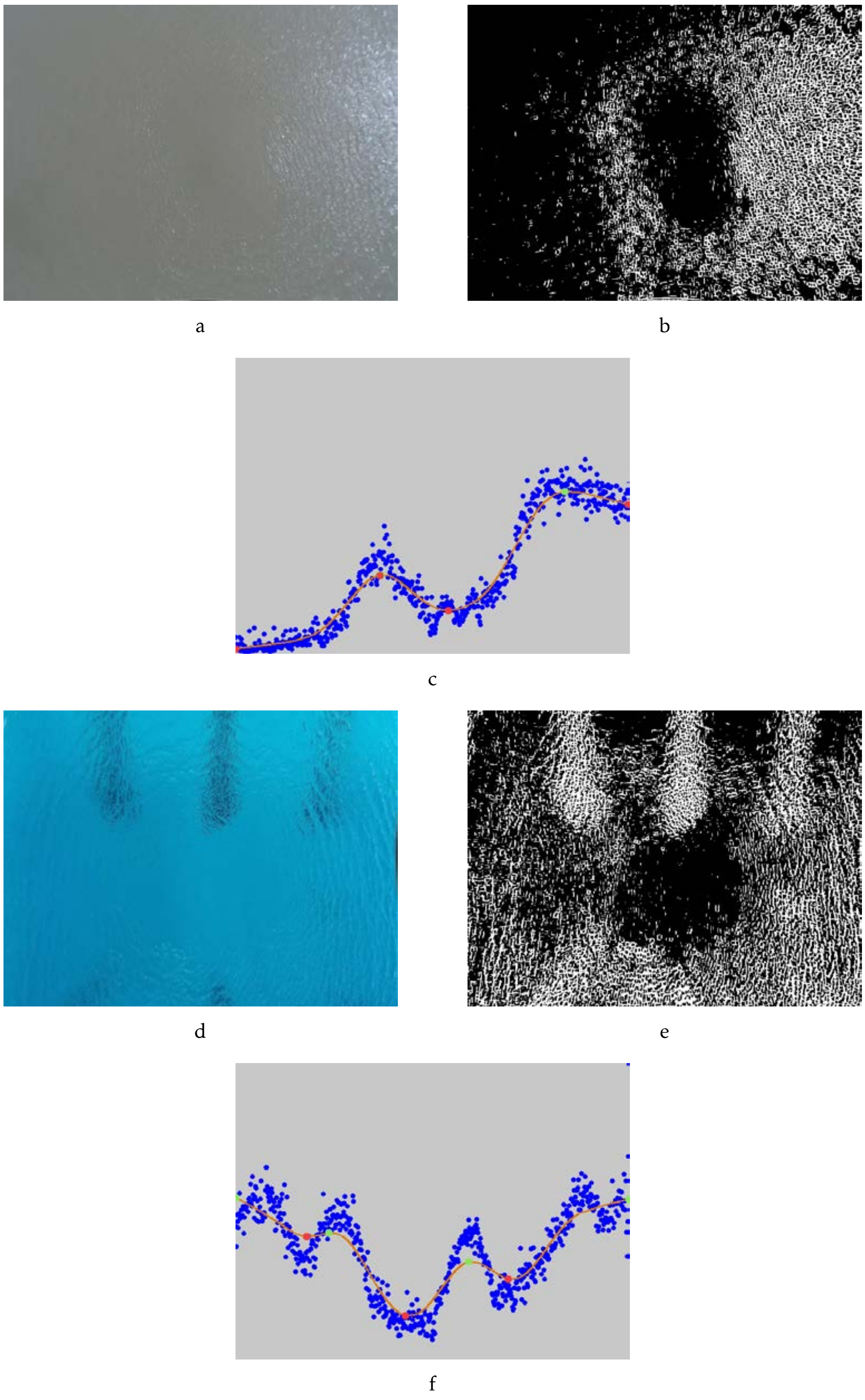
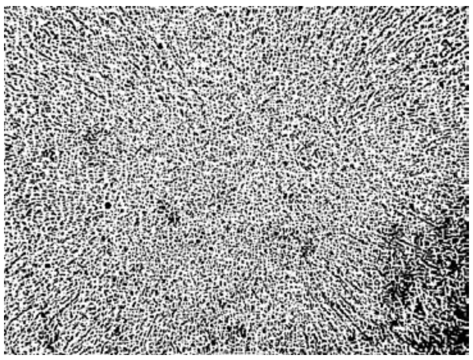
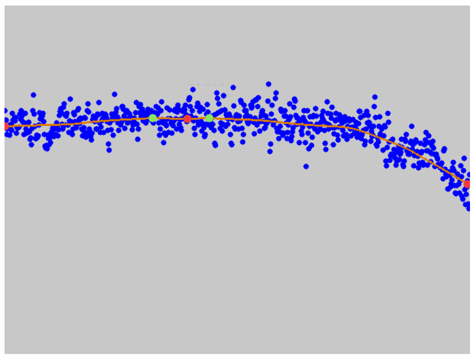
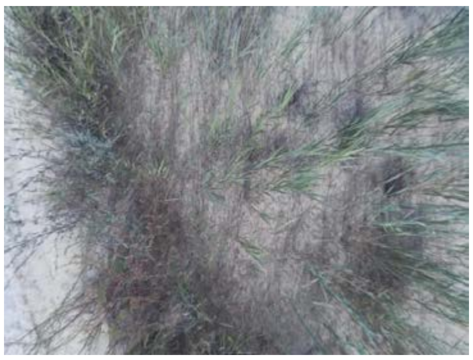


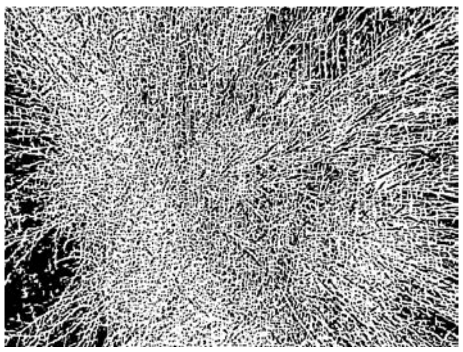
Figure 3.13: Example of static texture feature extraction for two water-terrains. (a,d) Input images; (b,e) Gabor filter; and (c,f) the width projection calculation.

a


b


c


d


e

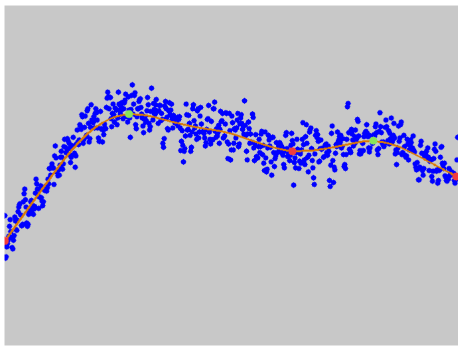
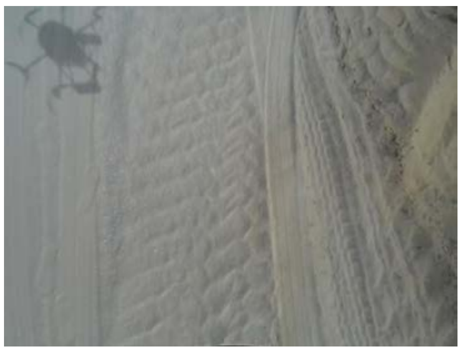
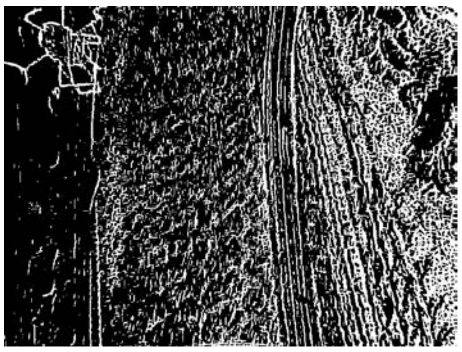
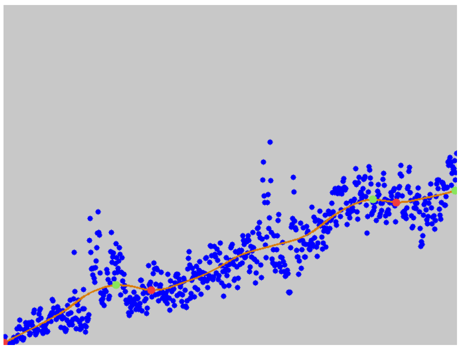
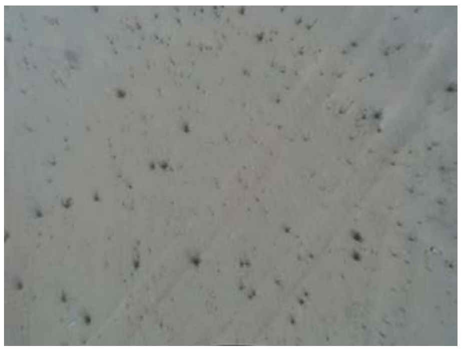

f

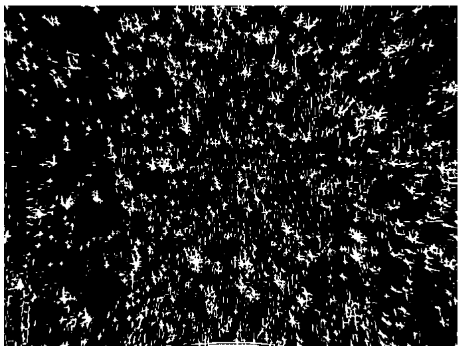
Figure 3.14: Example of static texture feature extraction for two vegetation-terrains. (a,d) Input images; (b,e) Gabor filter; and (c,f) the width projection calculation.


a


b


c


d


e

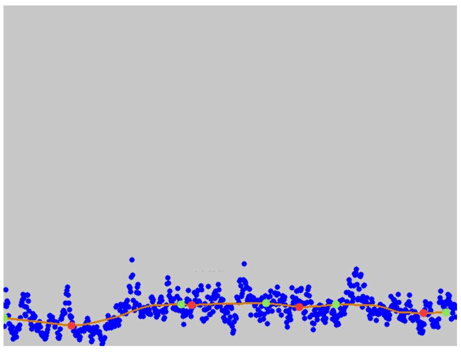

f

Figure 3.15: Example of static texture feature extraction for two sand-terrains. (a,d) Input images; (b,e) Gabor filter; and (c,f) the width projection calculation.

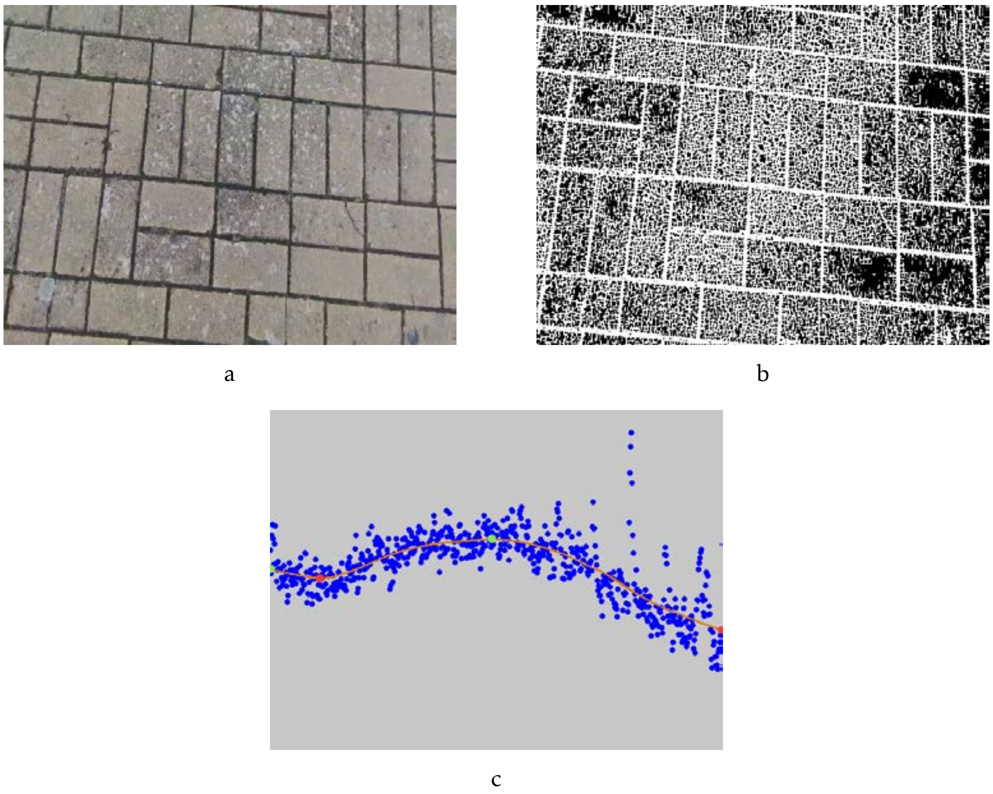


Figure 3.16: Example of static texture feature extraction for asphalt-terrain. (a) Input image; (b) Gabor filter; and (c) the width projection calculation.

1. Rounding the pixels horizontally and vertically to the nearest values of W and H, respectively, which are multiple of 10, in the normalized image. This procedure is intended to normalize raw images;

2. Sequential selection of 10 by 10 regions (initial clusters) in the image;

3. Automated initial allocation of centroids c_j, $j \in [1, W \cdot H/100]$. The pixel's average intensity calculation is represented in (3.8):

$$I_{av} = \frac{r' + g' + b'}{3}, \tag{3.8}$$

   where r', g', b' are the normalized values of a pixel's RGB vector.

4. Comparison of the rounded average intensity values for centroids from neighboring regions. The comparison proceeds vertically and horizontally relatively to each cluster. If the rounded average intensity values are equal to each other, two neighboring clusters are combined into one. In the new cluster the centroid is the pixel with a maximum average intensity value. This step needs to be repeated until there are M clusters c_j, $j \in [1, M]$ with the pairwise distinct rounded average intensity values of the centroids;

5. For each pixel x_i, $i \in [1, W \cdot H]$ two parameters are calculated: distance function d and color function f. They are shown in equation 3.9) and equation 3.10, respectively:

$$d(x_i, c_j) = \sqrt{(x_{i_w} - c_{j_w})^2 + (x_{i_h} - c_{j_h})^2}, \tag{3.9}$$

   where:

   - x_{i_w} and x_{i_h} are coordinates of the given pixel per width and height of the image;
   - c_{j_w} and c_{j_h} are coordinates of the centroid per image width and height.

$$f(x_i, c_j) = \sqrt{(r'_{x_i} - r'_{c_j})^2 + (g'_{x_i} - g'_{c_j})^2 + (b'_{x_i} - b'_{c_j})^2}, \tag{3.10}$$

   where:

   - r'_{x_i}, g'_{x_i}, b'_{x_i} are the normalized values of the RGB vector of the pixel x_i;
   - c_j is centroid of the cluster C_j;
   - r'_{c_j}, g'_{c_j}, b'_{c_j} are the normalized values of the RGB vector of the centroid c_j.

6. For each pixel x_i it is necessary to find a centroid c_a, $a \in [1, M]$ whose distance function value will be minimum and a centroid c_b, $b \in [1, M]$ whose color function value will be minimum. After that, the following differences are calculated, as it is expressed in equation 3.11 and equation 3.12:

$$d_{diff} = |d(x_i, c_a) - d(x_i, c_b)| \tag{3.11}$$

$$f_{diff} = |f(x_i, c_a) - f(x_i, c_b)| \tag{3.12}$$

7. The equation with lowest value is chosen as a priority function. Where d_{diff} equals f_{diff}, the distance function obtains priority, because pixels that are closer to each other are more likely belong to the same object than ones that have similar colors. The allocation of pixels to clusters is achieved according to the priority function, i.e. pixel x_i will be assigned to a cluster if the priority function value between this pixel and this cluster's centroid is minimum. Thus, the objective function of the clustering method can be expressed as in equation 3.13:

$$\begin{cases} \sum_{k=1}^{M} \sum_{i=1}^{N} d(x_i, c_k) \to min \\ \sum_{k=1}^{M} \sum_{i=1}^{N} f(x_i, c_k) \to min \end{cases} \tag{3.13}$$

8. Noise elimination. Due to its high effectiveness, the denoising non-local means method (Salmon, 2010) was chosen. This method is based on equation 3.14:

$$u(p) = \frac{1}{C(p)} \int_{\Omega} v(q) f(p,q) dq, \tag{3.14}$$

where *u(p)* is the filtered intensity value of a pixel color component at point *p*, *v(q)* is an unfiltered intensity value of a pixel color component at point *q*, *f(p,q)* is a weighting function and *C(p)* is a normalizing factor. Gaussian function is used as the weighting function, and it is represented in equation 3.15:

$$f(p,q) = e^{\frac{-|B(q)-B(P)|^2}{h^2}}, \tag{3.15}$$

where:

- *h* is the filter parameter (for RGB images *h* equals 3);
- *B(p)* is a local average intensity value of pixel color components around point *p*;
- *B(q)* is a local average intensity value of pixel color components around point *q*.

Normalizing factor *C(p)* is calculated, as in equation 3.16:

$$C(p) = \int_{\Omega} f(p,q) dq \tag{3.16}$$

In order to explain how the PSO was applied in this dissertation, the model of the proposed system, for terrain classification is presented in Figure 3.17.

From the proposed system model in 3.17, it is possible to visualize five main layers, namely:

- **Raw Image:** The images obtained by the RGB camera have a resolution of 640x480 pixels. After obtaining these images, the RGB information is then sent to the next layer;
- **Rectified Image:** In order for all RGB cameras to work with the same proposed algorithm, it is necessary to calibrate the cameras. Section 3.1.1 explained how to calibrate these RGB cameras;
- **Swarm Optimization:** This layer is responsible for segmenting the image according to the terrain types captured by the RGB camera;
- **Gray-Level Co-Occurrence Matrix:** This layer is responsible for transforming a segmented image into a GLCM matrix and calculating the features described in Section 3.2.5. Section 3.2.5 will cover in more detail how this layer was developed throughout this dissertation;

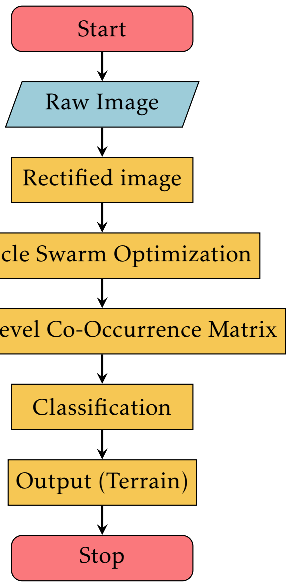


Figure 3.17: Proposed PSO system model.

- **Classification:** The outputs generated by the GLCM features feed a neural network (NN) which classifies the terrain. In this section, Multilayer Perceptron (MLP) architecture was used. The neural network inputs are the output values from the GLCM phase.

Given the system model explanation, in this section, four different terrain types (water, vegetation, asphalt and sand) were analysed. The following six Figures (Figures 3.18, 3.19, 3.20, 3.21, 3.22 and 3.23) show the results of the PSO of each terrain under study in this section. It is possible to conclude that by implementing the PSO algorithm, given the image segmentation, it filters the high frequencies caused by the downwash effect. In this way, water-type terrain, more specifically lakes, becomes more similar to sand-type terrain and therefore, the classification becomes less accurate when differentiating between water-type and non-water-type terrain.

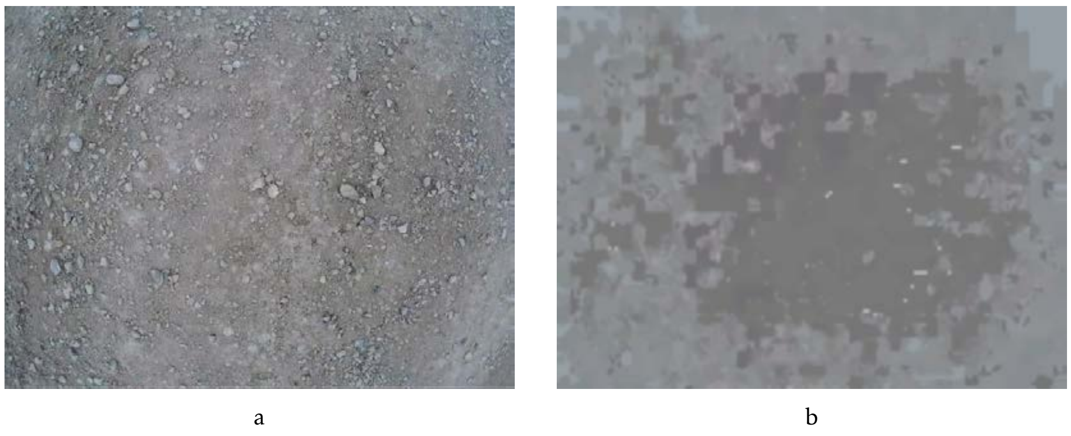


Figure 3.18: a) Input image (sand terrain A); b) PSO segmented data.

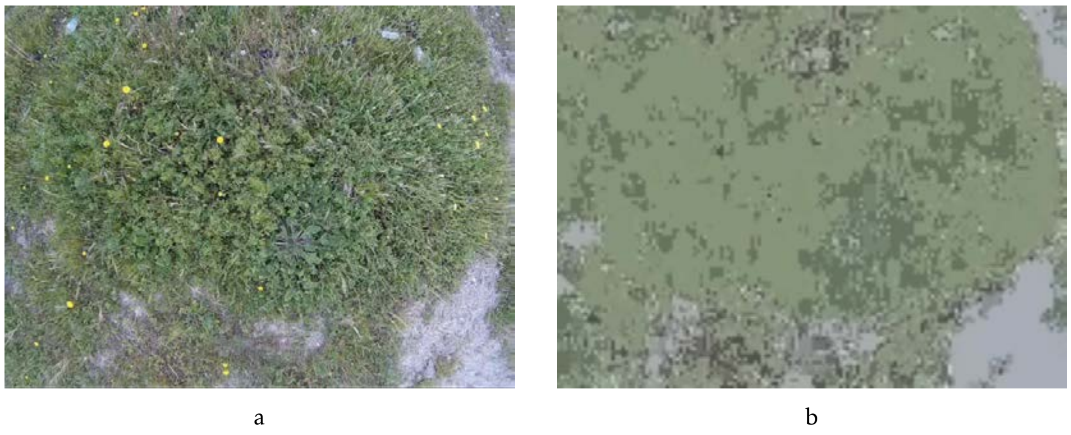


Figure 3.19: a) Input image (vegetation terrain); b) PSO segmented data.

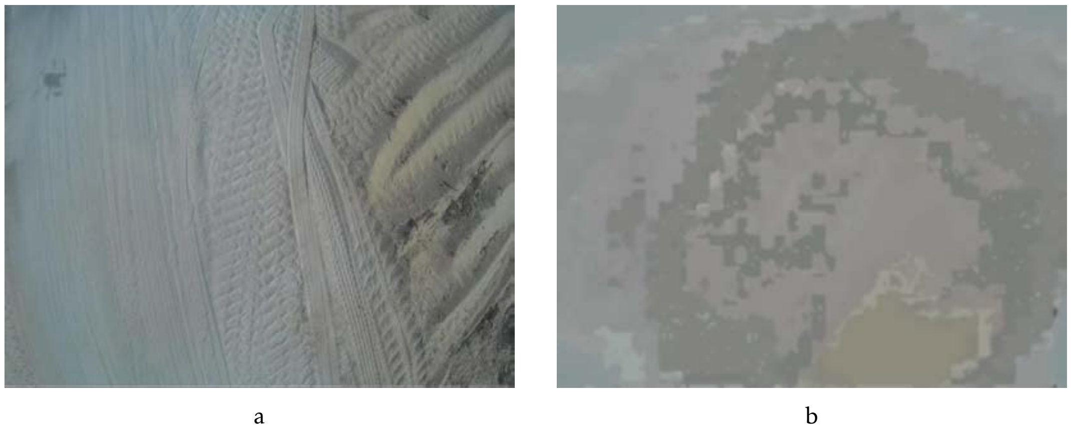


Figure 3.20: a) Input image (sand terrain B); b) PSO segmented data.

3.2.3 Fourier and Empirical Mode Decomposition

Although there are many ways to perform signal decompositions, two major groups can be defined:

1. Decompositions by prefixed base functions, such as Fourier decomposition;
2. Decompositions by data characteristics, such as Linear Discriminant Analysis (LDA), Principal Components Analysis (Principal Components Analysis (PCA)), Empirical Mode Decomposition (EMD).

A salient example of decomposition in the first group is Fourier analysis, where the base functions are complex exponentials. A recent example of decomposition in the second group is empirical mode decomposition, or EMD (Rato, 2012).

Starting with Fourier decomposition: this is well known for the paradigmatic case of signal decomposition by prefixed base functions (Oppenheim et al., 1999). For a discrete

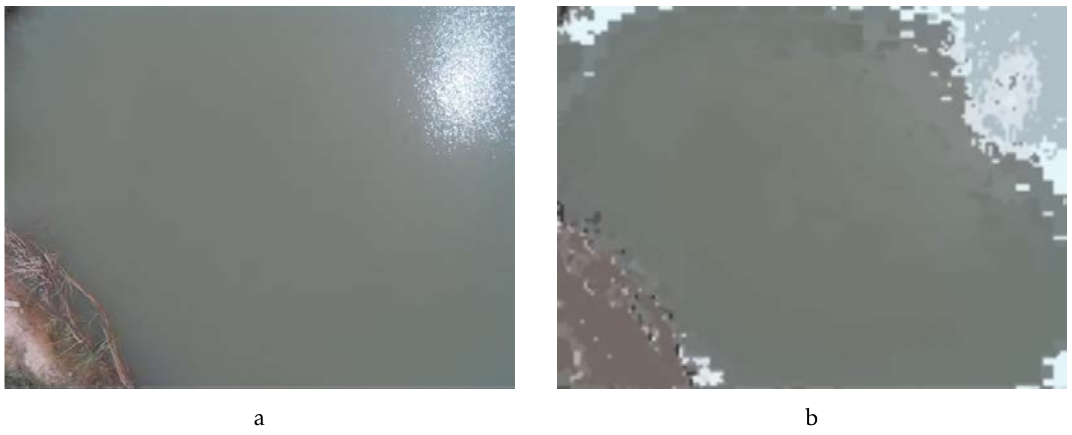


Figure 3.21: a) Input image (water terrain A); b) PSO segmented data.

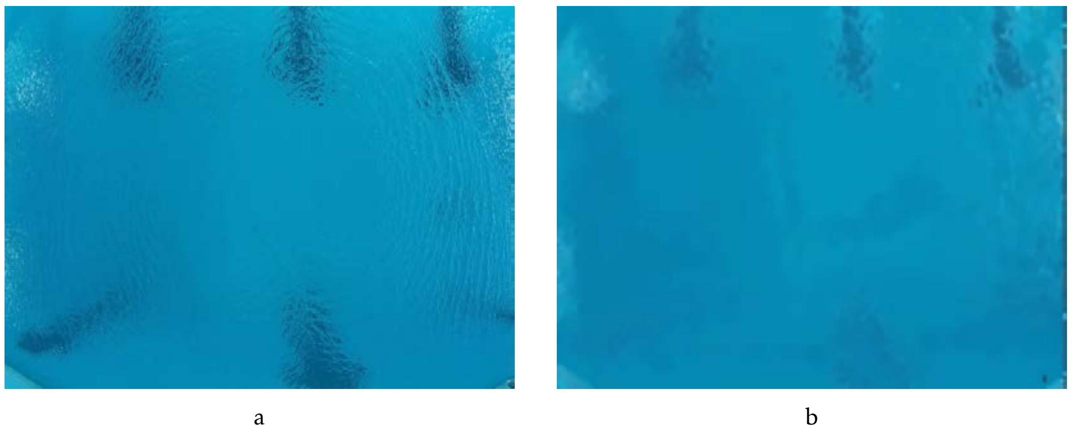


Figure 3.22: a) Input image (water terrain B); b) PSO segmented data.

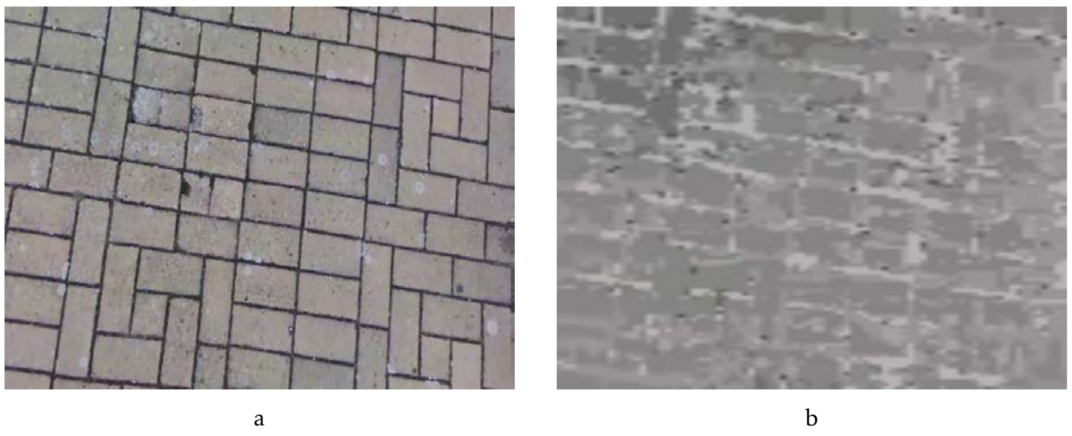


Figure 3.23: a) Input image (asphalt terrain); b) PSO segmented data.

signal F with N samples, Fourier decomposition produces a C sequence of N complex coefficients, $C = \{c[k]\}$, $0 \le k < N$, which correspond to N sinusoidal signals, S_k, $0 \le k < N$, each with N samples, $S_k = \{s_k[n]\}$, $0 \le n < N$. The sum of these N sinusoids reconstructs the original signal.

Where $F = \{f[n]\}$, $0 \le n < N$, a discrete signal with N samples. Then the coefficients $c[k]$ will be given by equation 3.17.

$$c[k] = \frac{1}{N} \cdot \sum_{n=0}^{N-1} f[n] \cdot e^{-j2\pi \frac{k}{N} n} \tag{3.17}$$

Fourier decomposition can be seen as the projection of the X signals into complex exponentials, in each base function. Instead of complex exponentials, it is possible to work with sinusoids. Thus, defining $c[N] = c[0]$, the following equations (3.18 and 3.19) are defined as base functions, which are all orthogonal to each other (Oppenheim et al., 1999):

$$a[k] = \frac{c[k] + c[N-k]}{2}, \ where \ 0 \le k < N \tag{3.18}$$

$$b[k] = j \cdot \frac{c[k] + c[N-k]}{2}, \ where \ 0 \le k < N \tag{3.19}$$

The values $s_k[n]$ are obtained by equation 3.20:

$$s_k[n] = a[k] \cdot \cos\left(2\pi \frac{k}{N} n\right) + b[k] \sin\left(2\pi \frac{k}{N}\right) \tag{3.20}$$

The signal F can be reconstructed by the sum of the N sinusoids S_k (equation 3.21).

$$F = \sum_{k=0}^{N-1} S_k \tag{3.21}$$

The following example will hel better understand this signal-decomposition method. Considering $N = 16$ and F the input signal with the following data: [1,3,-1,4,5,-1,-8,-3,1,5,8,2,-4,-7,-1,2], where $F[0] = 1$, $F[1] = 3$, ..., $F[15] = 2$. Figure 3.24 represents this data sequence.

The Fourier decomposition sequence can be applied in coefficients $c[k]$, $a[k]$ and $b[k]$, obtaining the following N sinusoids S_k

Although Fourier decomposition works for sinusoidal signals, it depends on the defined orthogonal base functions which are fixed amplitude and fixed frequency sinusoidal functions, as mentioned above. Thus, for this dissertation, the EMD was chosen in order to work with all signal types excluding dependency on exclusively sinusoidal signals and predefined functions (Huang et al., 1998). This decomposition gives rise to various empirical zero-mean oscillating modes, $\varphi_i(t)$, conventionally known as Intrinsic Mode Functions (IMF). In addition to a set of IMFs, EMD also produces a residual signal, known as baseline, where the average and baseline information is shown.

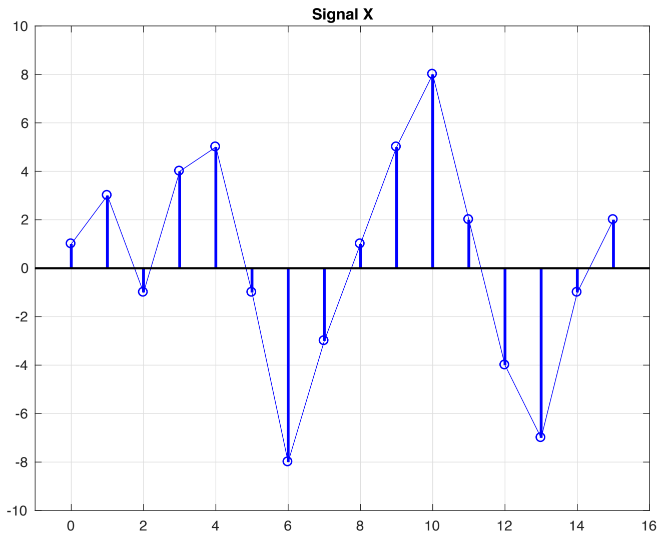


Figure 3.24: Input data-set- sequence representation. Adapted from (Rato, 2012).

EMD was developed for the signals decomposition that oscillate around a baseline but has non-stationary characteristics. This is the case with many signals, such as biological, climatic, econometric, among others. Unusually, EMD has no analytical definition, only algorithmic, and only for discrete time finite signals.

IMFs are determined using an iterative algorithm called "sifting" (Huang et al., 1998), and do not use pre-fixed base functions. This algorithm provides a collection of IMFs, Φ 0 < i < L, and one baseline, Φ_L.

Given the same data-set that was used in Figure 3.24, the input data was decomposed by EMD. The results can be observed in Figure 3.25.

It is important to note in Figure 3.25 that the first diagram represents the original sequence and the last diagram represents the sequence reconstructed by the sum of all IMFs and the baseline.

Before explaining how EMD was used in this dissertation, it is necessary to present the general description of EMD and how it works:

- **General Description:** EMD - decomposition in empirical modes - is performed by an intrinsically discrete algorithm with local characteristics, and no analytical counterpart is developed. EMD decomposes a signal into its constituent IMFs. The algorithm works by successive refinements, “sifting” the signal, until reducing it to an IMF. It removes this IMF from the original and resumes considering the residue from the previous operation as a new original signal. It ends when the residue is irreducible. It is an algorithm considered “simple” and “natural”, with local characteristics (Huang et al., 1998);

- **The Algorithm:** The EMD decomposition algorithm is intrinsically computational

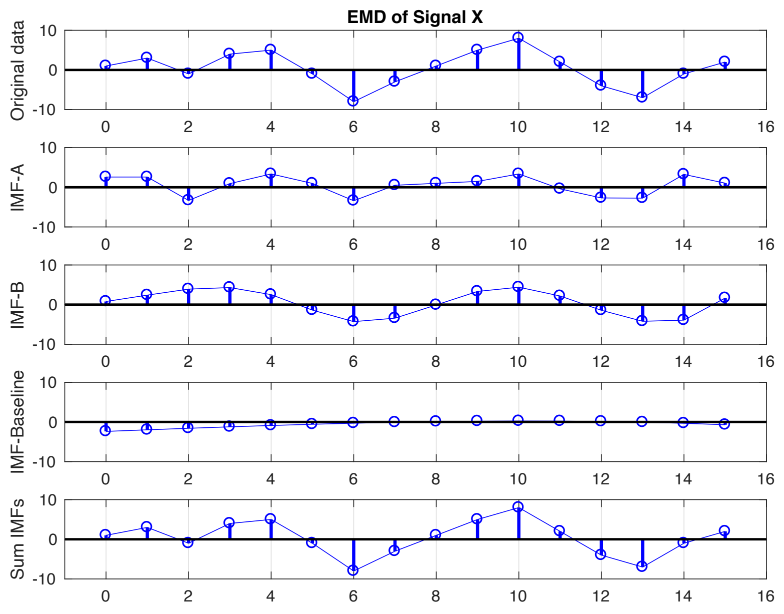


Figure 3.25: Signal decomposition using EMD. Adapted from (Rato, 2012).

and no analytical counterpart is developed. The description of the algorithm is as follows:

1. Let S be a discrete time signal and let $i = 1$ be the iteration variable;
2. The following step is to sift S to obtain an IMF Φ_i. Before start this step, let $\tilde{S}$ be an auxiliary copy of S:
   a) Determine all the maxima and minima of $\tilde{S}$;
   b) Determine the corresponding envelopes: $\hat{\mathrm{M}}=\left\{\hat{m}[n]\right\}$ corresponds to the upper envelope and $\check{\mathrm{M}}=\left\{\check{\mathrm{m}}[n]\right\}$ corresponds to the center envelope;
   To be easier to observe, Figure 3.26 shows the corresponding envelopes of the signal S, where the upper envelope is defined by the red line and the lower envelope is defined by the green line;
   c) Define $\mathrm{M}=\frac{\hat{M}+\check{M}}{2}$ as the mean between the upper envelope and center envelope as shown in Figure 3.26 (purple line);
   d) To continue the iteration, M will have to fulfill the following condition:
      – The energy of M must be above a certain threshold. To calculate the energy, the max energy ratio concept was applied (Wang et al., 2010). Energy ratio is the ratio of the energy of the signal at the beginning of sifting, S, and the average envelope energy, M. Equation 3.22 translates this concept into a mathematical formula:

$$ER \triangleq 10\log_{10}\left(\frac{||S||^2}{||M||^2}\right) \tag{3.22}$$

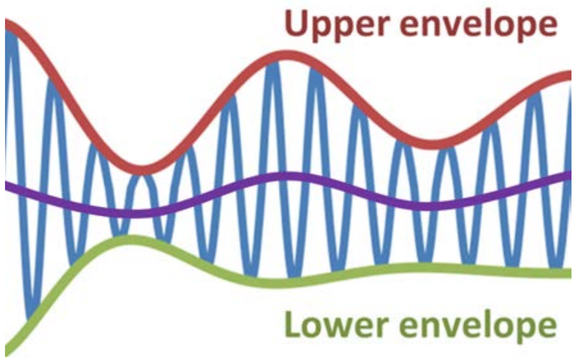


Figure 3.26: Signal S and its corresponding envelopes.

e) If M does not satisfy the condition, i.e., if it the energy is below the set threshold value, part 1) ends. Hence, $\Phi_i = \tilde{S}$ and proceed to step 3);

f) If M satisfies the condition, i.e., if energy is higher than the set threshold, then it will jump again to step 2);

g) The process is reinitialized by subtracting M from $\tilde{S}$ to get a new $\tilde{S}$, i.e. $\tilde{S} \leftarrow \tilde{S}$ - M and it is necessary to jump to 2 a) to restart the algorithm;

3. Set the new signal S to be $S \leftarrow S - \Phi_i$. Also the iteration variable is incremented, i.e., i $\leftarrow$ i +1;
4. For the new signal S, it is necessary to verify that both conditions have been validated to continue the iteration:
   - Energy must be higher than the set threshold;
   - Display of more than three extremes (local, minimum and maximum).
5. If signal S meets both iteration conditions, it is resumed from stage 2). If S does not meet at least one of the conditions to continue the iteration, consider S the baseline, Φ_L, and finish the iteration.

To conclude, this describes the EMD algorithm that allows one to obtain a collection of IMFs Φ_i, where $0 < i < L$, and enables its residual baseline Φ_L to be obtained. However, the number of IMFs that are obtained, when using the EMD algorithm, is not fixed because different signals can have different energy and also different local maximum and minimum numbers. Thus, in order to solve this issue, only the first 3 IMFs and the baseline are used to classify the terrains. Figure 3.27 shows an example of a water terrain analysed using the EMD algorithm.

In order to explain how the EMD was applied in this dissertation, the model of the proposed system, for terrain classification, is presented in Figure 3.28.

It is possible to visualize six main processes in the architecture, namely:

- **Rectified Image:** It is necessary to calibrate the camera before any process, to make the proposed algorithms universal. Only then, is it possible to work with all RGB

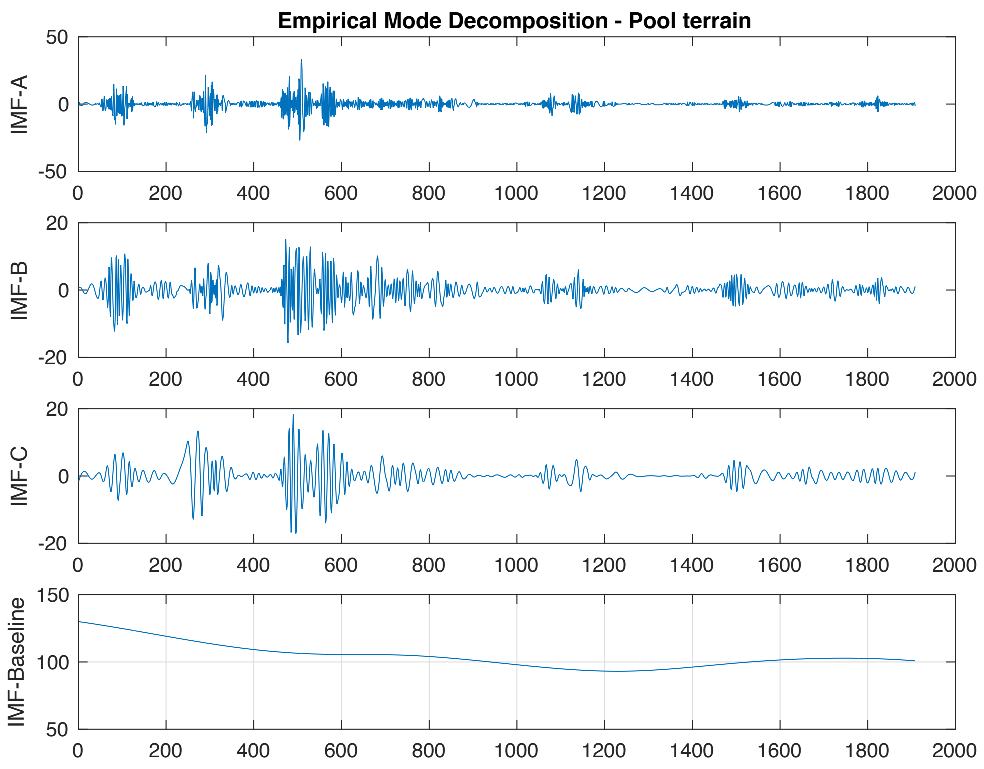


Figure 3.27: Empirical Mode Decomposition in Pool terrain type - First three IMFs and the baseline. Each EMD analysis was performed with a resolution of 40 dBs.

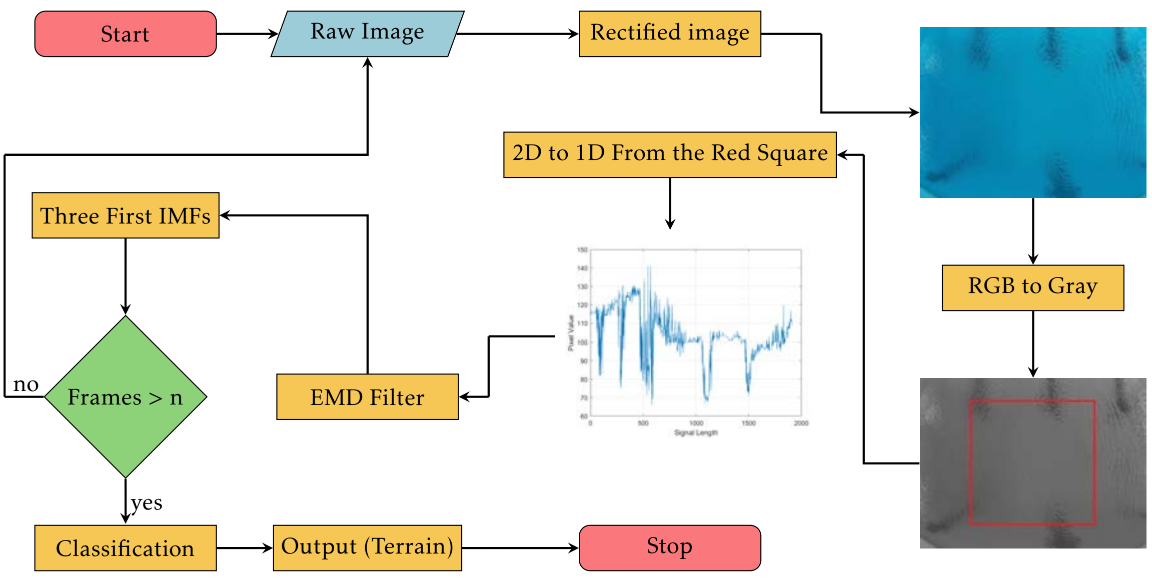


Figure 3.28: Proposed EMD system model.

cameras regardless of their resolution. Section 3.1.1 explained how to calibrate these RGB cameras;

- **RGB to Gray:** Regarding the gray scale conversion, the system merges the three channels (red, green and blue) using the CCIR 601 standard (Matos-Carvalho et al., 2019) as shown in equation 3.23:

$$Gray_{scale} = 0.299\,Red + 0.587\,Green + 0.114\,Blue \tag{3.23}$$

- **2D to 1D From the Red Square:** In order to increase the computational speed, the proposed algorithm is not used on the entire image but only on a part of it. In this dissertation, the use of a 477x477 square is proposed (red square as can be seen in Figure 3.28), where each value represents the pixel that constitutes the edges of the square. Next, the 2D representation was converted into 1D and a vector was created where all values were stored (blue line);

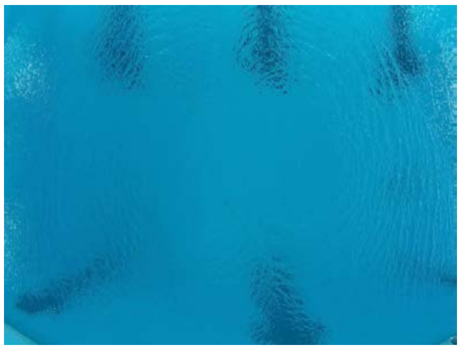
- **EMD Filter:** This phase will apply the EMD algorithm, which has been explained in this section, to the 1D values. As also mentioned, only the first 3 IMFs and the baseline are used to be sent as the NN's input information;

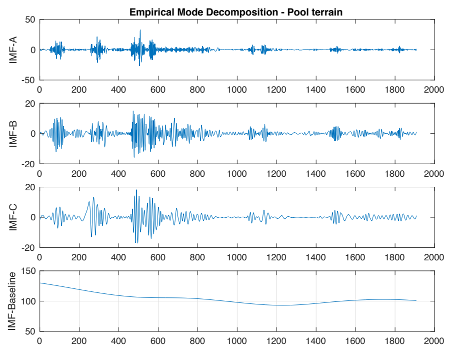
- **Frames > n:** "n" is the total number of frames required to increase the algorithm's accuracy. This threshold value was chosen empirically by the author.

- **Classification:** The outputs generated by the EMD phases are turned into inputs for a Neural Network (NN) (Specht, 1991) tasked with classifying the terrain the UAV is flying over. Machine learning techniques have already been proven to be efficient for terrain classification (Giusti et al., 2016; Heung et al., 2016; Mora et al., 2017). In this section, an NN was used, namely a Multilayer Perceptron (MLP) architecture. The Neural Network inputs are the output values from EMD algorithms.

Now that the system has been explained, in this section five different terrain types (lake, pool, vegetation, dirt and sand) were analysed. The behavior changes according to the type of terrain: in lake and pool, as it is a water-like terrain, a circular movement is observed; in vegetation, the movement occurs linearly from the inside to the outside; in sand, the movement is almost static. The following six Figures (Figures 3.29, 3.30, 3.31, 3.32, 3.33 and 3.34) show the results of the first three IMFs and the baseline of each terrain under study in this section. It is possible to visualize and conclude that, using the EMD algorithm, the more texture there is in an image, the greater the frequency of each IMF that is obtained, allowing one to differentiate between water-type terrain and non-water type terrain. Section 4.3 will detail more about this study.

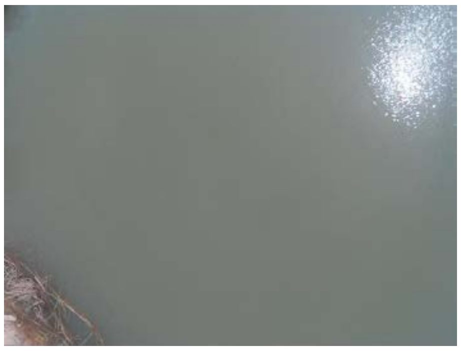
In the same way as for Section 3.2.1, this section also proposes two heuristics characteristics in order to send these two features to the NN and classify the terrain under study. The following heuristics characteristics take into consideration the three IMFs of each terrain:

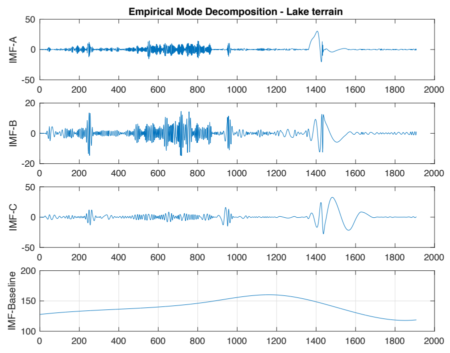




a b

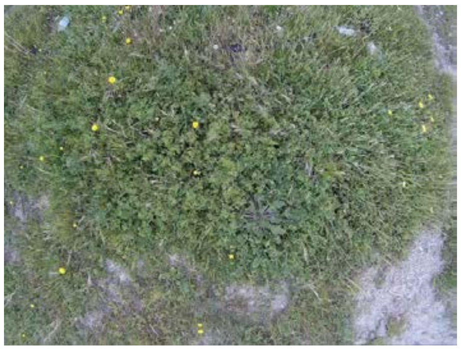
Figure 3.29: a) Input image (water terrain A); b) its first three IMFs.

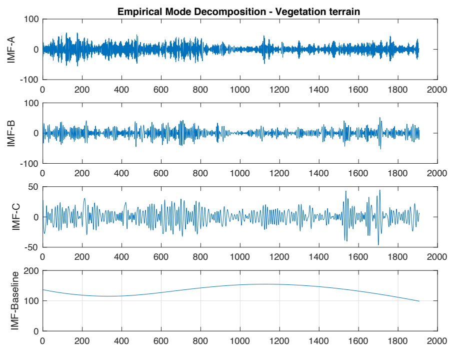




a b

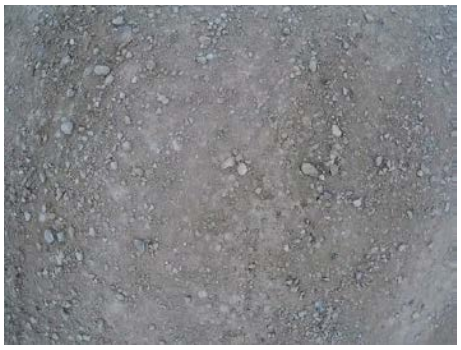
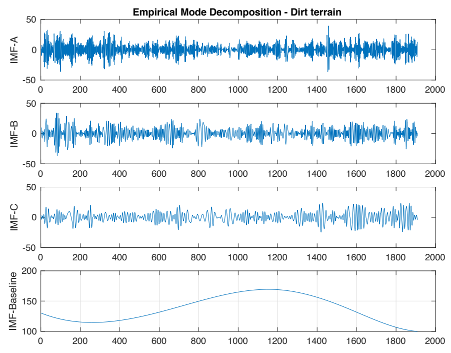
Figure 3.30: a) Input image (water terrain B); b) its first three IMFs.





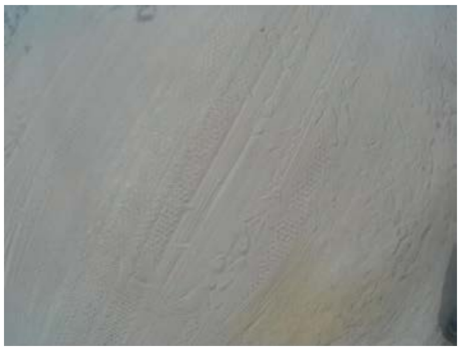
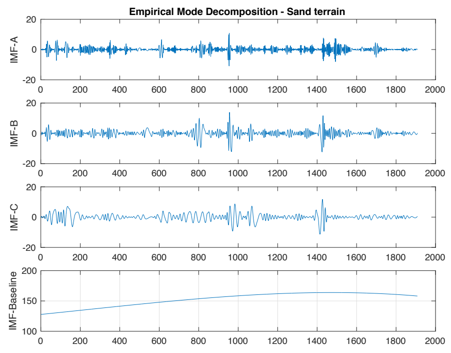
a b

Figure 3.31: a) Input image (vegetation terrain); b) its first three IMFs.

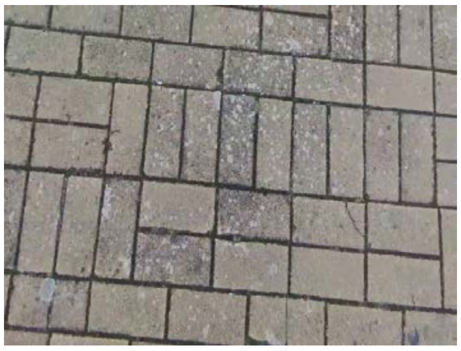
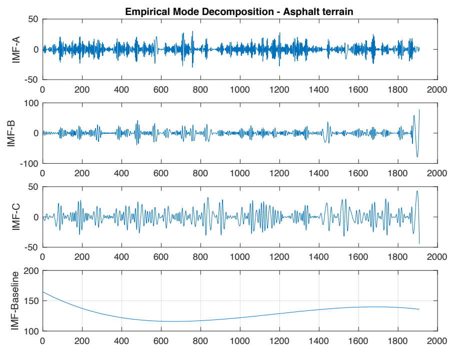
a b

Figure 3.32: a) Input image (sand terrain A); b) its first three IMFs.

a b

Figure 3.33: a) Input image (sand terrain B); b) its first three IMFs.

a b

Figure 3.34: a) Input image (asphalt terrain); b) its first three IMFs.

1. Number of local maximums given by an IMF;
2. Number of times the IMF has crossed zero along the vector.

3.2.4 Wiener-Khinchin

It is known that the UAV rotation and its camera in relation to the terrain is an extremely important factor in the terrain classification. Using the downwash effect, it is necessary to study an algorithm that is not influenced by these factor presented above. Hence, this section presents the fourth algorithm developed in this dissertation: The W-K Filter, where the core of the algorithm occurs in the frequency domain (the algorithms presented in 3.2.1 and 3.2.3 are developed in the spatial domain).

As explained above, the W-K Filter, with the downwash effect, is used in order to classify any terrain regardless of UAV/Camera rotation. In order to make this concept easier to understand, an example is provided in Figures 3.35, 3.36 and 3.37, where two factors are observed:

1. UAV rotation;
2. Circumference (represented as purple) caused by the downwash effect.

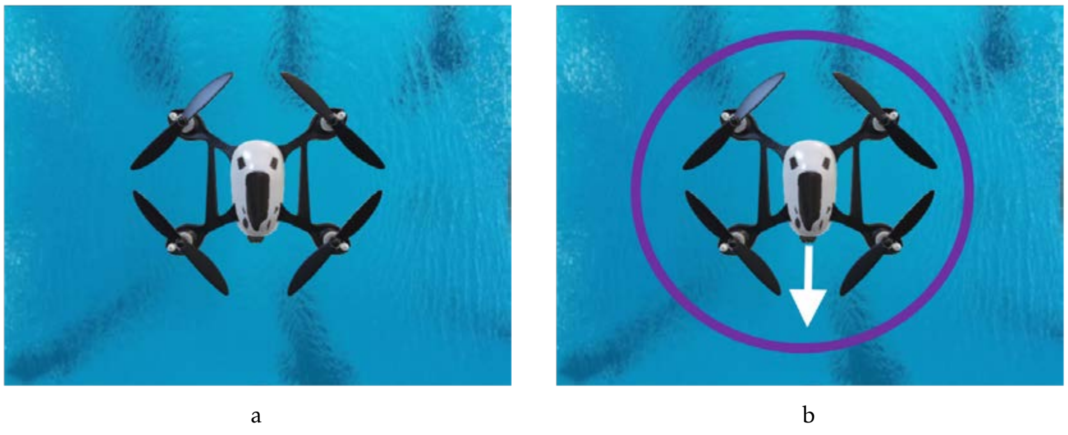


Figure 3.35: W-K Filter concept explanation. a) UAV on water terrain; b) UAV with 0º rotation.

As can be seen from Figures 3.35, 3.36 and 3.37, it can be concluded that the purple circumference caused by the downwash effect is almost the same regardless of the UAV position and rotation, i.e., the objective of using the W-K Filter is then to have a rotation invariant algorithm. Mathematically speaking, this concept can also be explained by the autocorrelation of the signal produced by the circumference as shown in the following steps.

1. **Average Values:** After obtaining the purple circumference values, in order to be able to compare with different terrain types, it is necessary and important that all

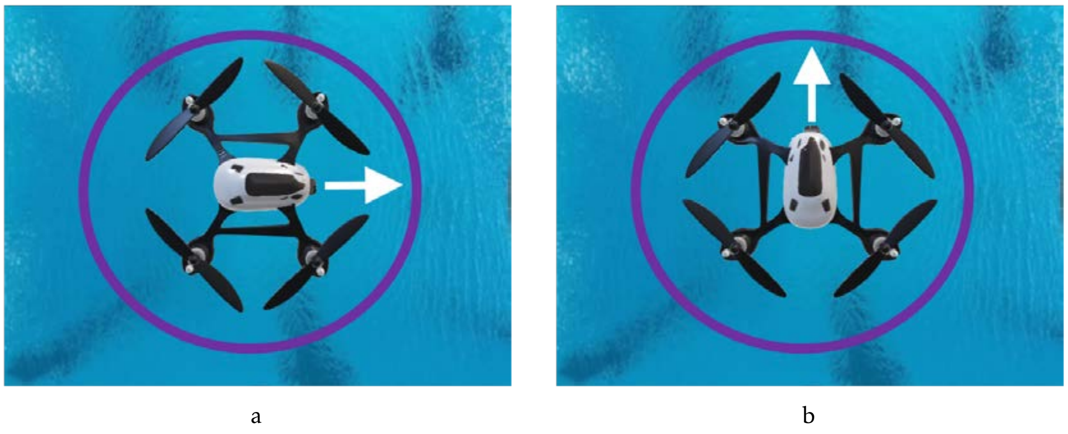


Figure 3.36: W-K Filter concept explanation. a) UAV with 90° rotation degrees; b) UAV with 180° rotation.

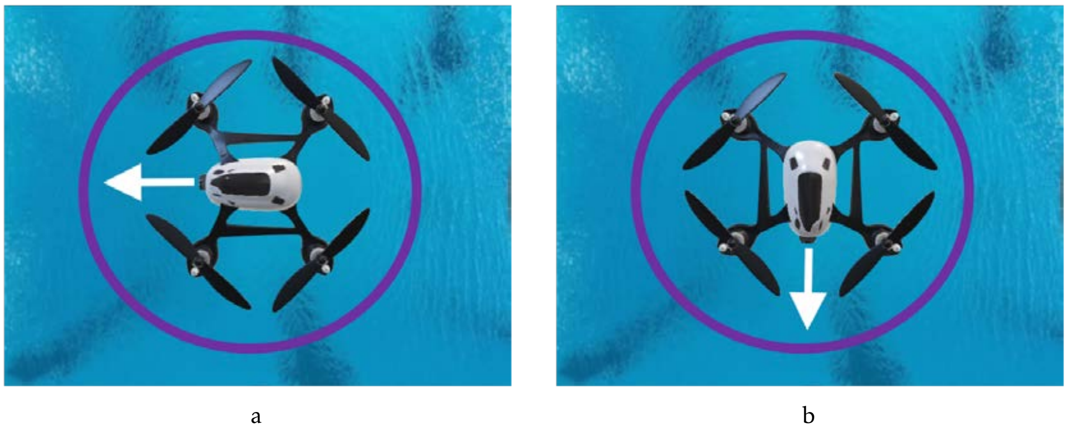


Figure 3.37: W-K Filter concept explanation. a) UAV with 270° rotation degrees; b) UAV with 360° rotation.

terrains under study have the same average value. For this reason, the average value is removed from the original signal.

2. **Normalization:** To complement the idea of being able to compare different types of terrain, it is necessary to normalize the input data in order to be able to use general thresholds.

   As the Wiener-Khinchin theorem uses the concept of energy in the frequency domain, it becomes intuitive to normalize the input data according to energy, as shown in equation 3.24:

$$X = \frac{X}{\|X\|} = \frac{X}{\sqrt{X \cdot X^T}} \tag{3.24}$$

3. **Frequency Domain:** After normalizing the circumference values, they were transformed to the frequency domain:

$$F(k) = \sum_{j=0}^{N-1} f(j) \cdot e^{\frac{-i2\pi kj}{N}} \tag{3.25}$$

Where F(k) is the DFT coefficients of f(j) and the N is the resolution (the resolution must be equal or greater than the signal length).

4. **Autocorrelation Process:** With the values in the frequency domain, the signal crossed with itself is then performed, i.e., the core of Wiener-Khinchin theorem is presented in equation 3.26:

$$F'(k) = F(k)^2 \tag{3.26}$$

5. **Spatial Domain:** As the last step, an inverse transform is performed to obtain the output in the spatial domain (equation 3.27):

$$f(j) = \frac{1}{N} \sum_{k=0}^{N-1} F'(k) \cdot e^{\frac{i2\pi kj}{N}} \tag{3.27}$$

Figure 3.38 shows an example of applying the Wiener-Khinchin filter to a water terrain type.

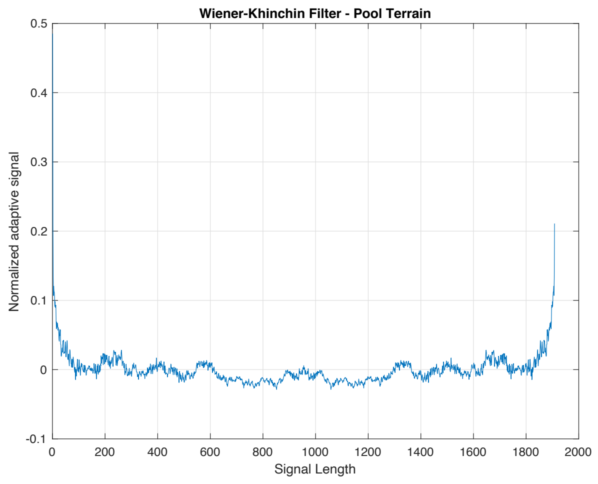


Figure 3.38: Wiener-Khinchin Filter - Autocorrelation result from pool (water) terrain type.

In order to explain how the W-K Filter was applied, the model of the proposed system, for terrain classification, is presented in Figure 3.39.

Based on the system model proposed in 3.39, it is possible to visualize six main processes that have been identified in the architecture, namely:

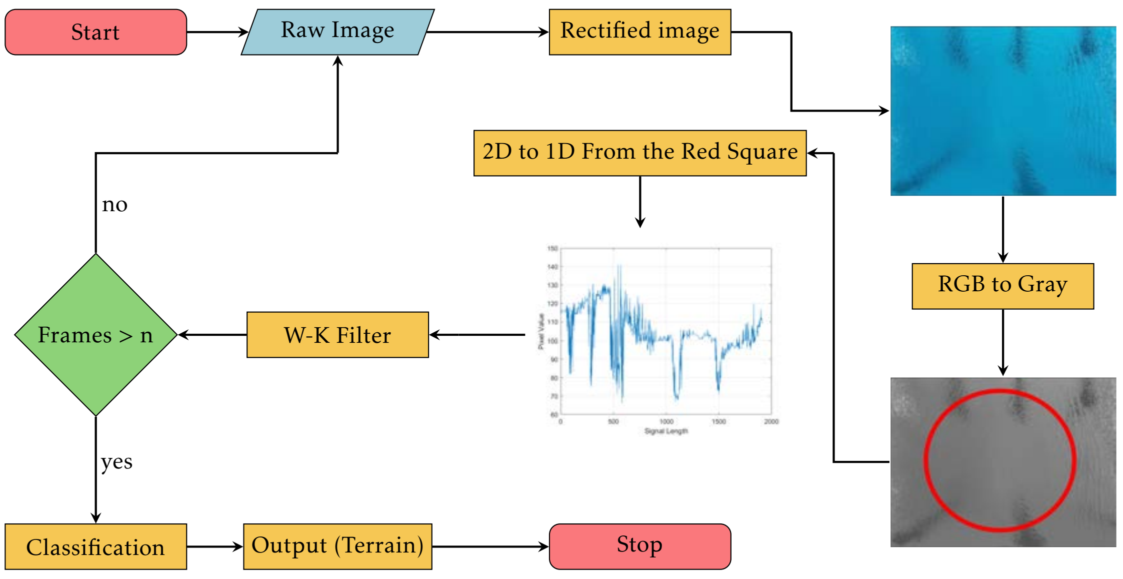
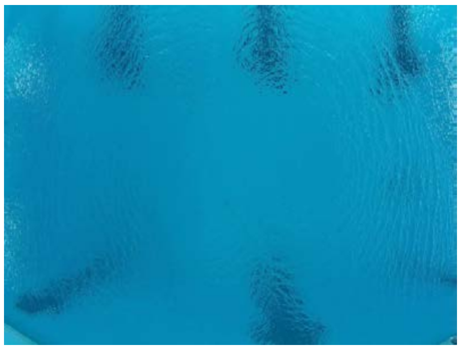


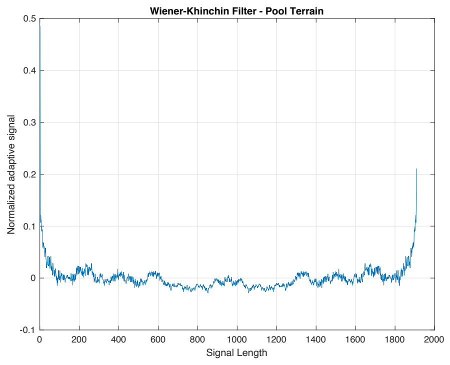
Figure 3.39: Proposed W-K Filter system model.

- **Rectified Image:** As mentioned in Section 3.2.3, to work with all RGB cameras with the same proposed algorithm, it is necessary to calibrate the cameras. Section **??** explained how to calibrate these RGB cameras;
- **RGB to Gray:** In the same way as in Section 3.2.3, a gray scale conversion was used in order to increase the system speed. Equation 3.23 was used to convert the RGB scale into a gray scale image;
- **2D to 1D from the Red Circumference:** For the terrain classification to be independent of UAV rotation, in this dissertation the use of a circumference is proposed (a red circumference as can be seen in Figure 3.39), where each value represents the pixel that constitutes the edges of the circumference (in this thesis a 477x477 pixel circumference was used). Next, the 2D representation was converted into 1D and a vector created where all values were stored (blue line);
- **W-K Filter:** Receiving the output from the previous block, this phase will use the 1D values to apply the W-K Filter to calculate the autocorrelation;
- **Frames > n:** "n" is the total number of frames required to increase the algorithm's accuracy. This threshold value was chosen empirically by the author.
- **Classification:** The outputs generated by the W-K Filter phases are turned into inputs for a Neural Network (NN). The NN output classifies the terrain over which the UAV is flying. In this section, a Multilayer Perceptron (MLP) architecture was used. The Neural Network inputs are the output values from the W-K Filter phase.

To validate the proposed algorithm, four different terrain types (lake, pool, vegetation and sand) were analysed. The following five Figures (Figures 3.40, 3.41, 3.42, 3.43, 3.44

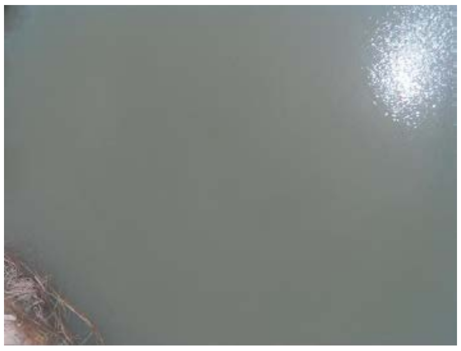
and 3.45) show the W-K Filter results for each terrain under study. It is possible to conclude that by implementing the W-K Filter algorithm, the more texture there is in an image, the greater the frequency of each output, as shown in Figure 3.42 and Figure 3.45. However, water-type terrain, more specifically in lakes, becomes more similar to sand-type terrain (low frequencies) and therefore, the classification becomes less accurate when attempting to differentiate between water-type and sand-type terrain.

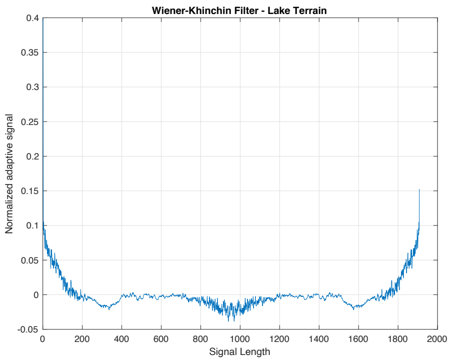




a b

Figure 3.40: a) Input image (water terrain A); b) its W-K Filter result.





a b

Figure 3.41: a) Input image (water terrain B); b) its W-K Filter result.

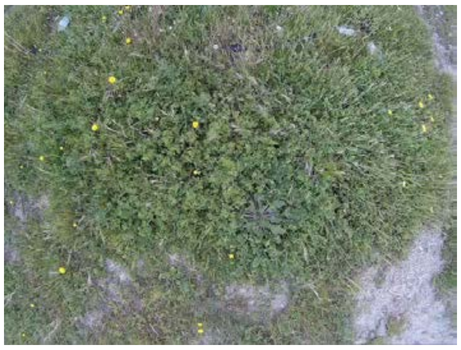
Three features were extracted from the W-K Filter results, which are then inputs of the NN terrain classifier.

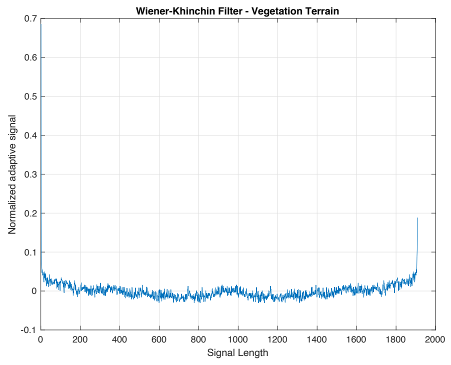
The selected features were:

1. Number of local maximums given by the W-K Filter output. Listing 3.5 shows how this features is calculated.

2. Number of times the W-K Filter result has crossed zero along the vector. This feature can be observed in Listing 3.6.

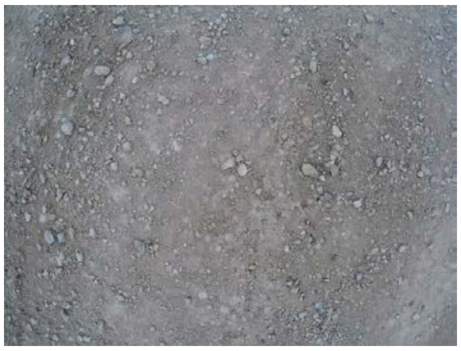
3. W-K Filter area. This feature can be observed in Listing 3.7.

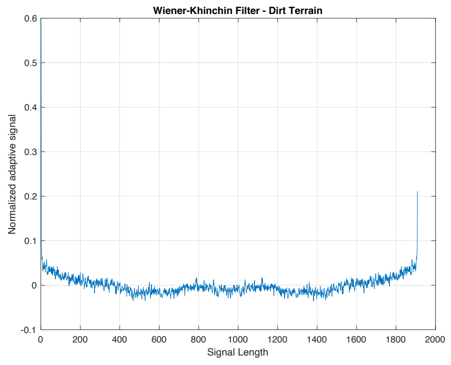



Wiener-Khinchin Filter - Vegetation Terrain
Normalized adaptive signal
Signal Length

a b

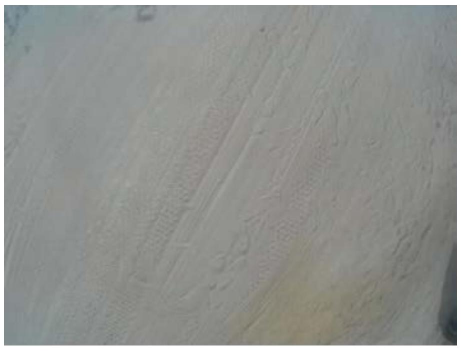
Figure 3.42: a) Input image (vegetation terrain); b) its W-K Filter result.

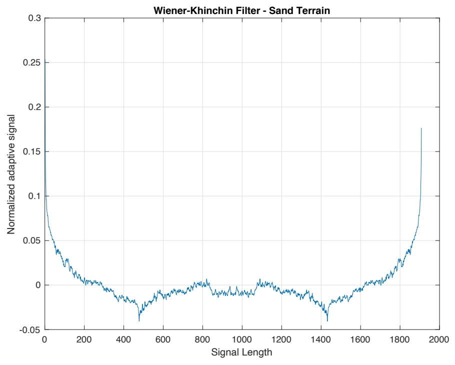



Wiener-Khinchin Filter - Dirt Terrain
Normalized adaptive signal
Signal Length

a b

Figure 3.43: a) Input image (sand terrain A); b) its W-K Filter result.




Wiener-Khinchin Filter - Sand Terrain
Normalized adaptive signal
Signal Length

a b

Figure 3.44: a) Input image (sand terrain B); b) its W-K Filter result.

Listing 3.5: Local maximums calculation.

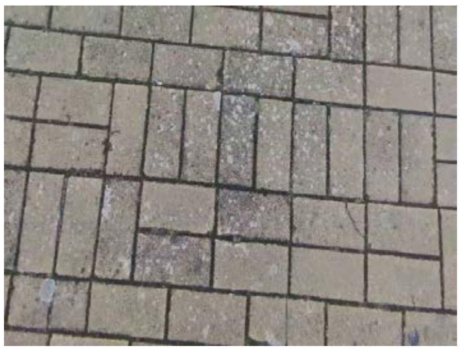
```
maxLocal(Vector wiener){
    // Define the variables
    totalMax = 0;
    currentMax = wiener[1];
    wienerSize = wiener.size();

    // Local maxima calculation
    for (j = 2; j < wienerSize; ++j) {
        if(currentMax > wiener[j-2] && currentMax > wiener[j])
            totalMax++;
        currentMax = wiener[j];
    }

    // Return, in percentage, the signal's local maxima
    totalMax = (totalMax/wienerSize)*100;
    return totalMax;
}
```

Listing 3.6: Zero crossing calculation.

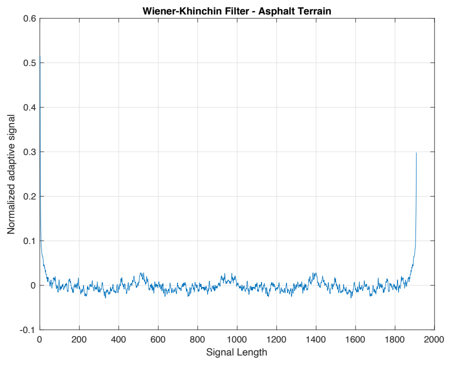
```
crossZero(Vector wiener){
// Define the variables
totalZeros = 0;
last = wiener[0];
wienerSize = wiener.size();

// Zero crossing calculation
for (j = 1; j < wienerSize; ++j) {
    auto current = wiener[j];
    if(current > 0 && last > 0 || current < 0 && last < 0) {
        last = current;
        continue;
        }
    totalZeros++;
    last = current;
}

// Return, in percentage, the signal's zero crossings
totalZeros = (totalZeros/(wienerSize-1))*100;
return totalZeros;
}
```

Listing 3.7: Area calculation.

```
area(Vector wiener){
    // Define the variables
    xLeft = 0;
    xRight=0;
    wienerSize = wiener.size();
    index=0;
    maxArea=0;
    interestArea=0;
    finalArea=0;

    // Calculate xLeft
    for (; index < wienerSize; ++index) {
        if(wiener[index] < 0) {
            xLeft = index;
            break;
        }
    }

    // Calculate xRight
    for (index = wienerSize-1; index >=0; --index) {
        if(wiener[index] < 0) {
            xRight = index;
            break;
        }
    }

    // Calculate max area
    for (index=0; index < wienerSize; ++index) {
        maxArea += (1 - wiener[index]);
    }

    // Calculate the interest Area
    for (index=xLeft; index < xRight; ++index) {
        interestArea += wiener[index];
    }

    // Return, in percentage, the area
    finalArea = (interestArea/maxArea)*100;
    return finalArea;
}
```



a



b

Figure 3.45: a) Input image (asphalt terrain); b) its W-K Filter result.

3.2.5 GLCM

As explained in Section 3.2, previous research works describe different terrain classification approaches mainly using static features from RGB images taken onboard UAVs. The next two sections (Section 3.2.5 and 3.2.6) present two features to be extracted from the UAV's downwash effect: the Gray-Level Co-Occurrence Matrix (GLCM) and the Gray-Level Run Length Matrix (GLRLM).

With regard to this section, two main steps are considered:

1. GLCM mask design;

2. The calculation of 27 features.

As a first step, it is necessary to create the GLCM N × N matrix. In this dissertation, the matrix dimensions are 256 × 256 as mentioned in Section 2.3.4, since the input images are 8Bit ranging between 0 and 255 levels (256 gray levels). Three variables were defined: $d = 100$ (offset), $\theta = 0$ (orientation) and a symmetrical matrix; the purpose is to obtain a trade-off between noise and system speed. These values were obtained by trial and error.

Listing 3.8 shows how the initial GLCM matrix was created. In that listing it is possible to observe the value in each GLCM matrix position, which is obtained by the sum of the image pixels, given a certain distance $d = 100$ and rotation $\theta = 0$. The second stage, to transpose matrix M^T, was developed as in Listing 3.9.

The last step to create the current algorithm, the symmetrical matrix, was developed, as shown in equation 2.30. Thus, Listing 3.10 was designed in order to present the final GLCM matrix.

Figure 3.46 illustrates an example of water terrain results using the GLCM algorithm. With a GLCM matrix calculated, 26 textural features and one intensity colour were extracted to classify the terrain. This takes into account the following base equations:

Listing 3.8: GLCM mask design.

```
glcm(img, glcmOffset, xi, yi, xf, yf){
    // Define the variables
    sizeGlcmMask = 256;
    glcmArray [sizeGlcmMask][sizeGlcmMask];
    rows = yf;
    cols = xf;
    pixelPtrRead = imageGray.data;

    // GLCM matrix - first step design
    for (y = yi; y < rows; ++y) {
        for (x = xi; x < cols-glcmOffset; ++x) {
            i = pixelPtrRead(y,x);
            j = pixelPtrRead(y,x+glcmOffset);
            glcmArray[i][j] = glcmArray[i][j] +1;
        }
    }
}
```

Listing 3.9: GLCM transpose mask design.

```
glcmTranspose(glcmArray [256][256]){
    // Define the variables
    sizeGlcmMask = 256;
    glcmTranspose [sizeGlcmMask][sizeGlcmMask];

    // GLCM transpose matrix design
    for (y = 0; y < sizeGlcmMask; ++y) {
        for (x = 0; x < sizeGlcmMask; ++x) {
            glcmTranspose[x][y] = glcmArray[y][x];
        }
    }
}
```

Listing 3.10: GLCM transpose mask design.

```
glcmAddedTranspose(glcmArray [256][256], glcmTranspose [256][256]){
    // Added transpose mask into GLCM matrix
    for (y = 0; y < sizeGlcmMask; ++y) {
        for (x = 0; x < sizeGlcmMask; ++x) {
            glcmArray[y][x] += glcmTranspose[y][x];
        }
    }
}
```

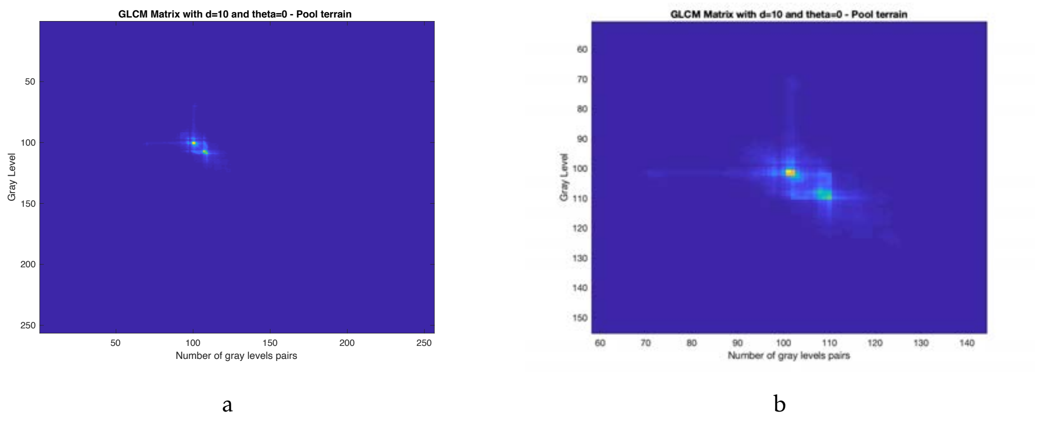


Figure 3.46: GLCM - a) Result from pool (water) terrain type. b) is a zoom of a).

$$p_{i-j,k} = \sum_{i=0}^{N-1}\sum_{j=0}^{N-1} p(i,j) \ if \ \ k = |i-j| \quad k = 0,\dots,N-1 \tag{3.28}$$

$$p_{i-j,k} = \sum_{i=0}^{N-1}\sum_{j=0}^{N-1} p(i,j) \ if \ \ k = i+j \quad k = 2,\dots,2N \tag{3.29}$$

$$p_x(i) = \sum_{j=1}^{N} p(i,j) \tag{3.30}$$

$$p_y(j) = \sum_{i=1}^{N} p(i,j) \tag{3.31}$$

$$HX = -\sum_{i=1}^{N} p_x(i)\cdot\log_2(p_x(i)) \tag{3.32}$$

$$HY = -\sum_{j=1}^{N} p_y(j)\cdot\log_2(p_y(j)) \tag{3.33}$$

$$HXY = -\sum_{i=1}^{N}\sum_{j=1}^{N} p(i,j)\cdot\log_2(p(i,j)) \tag{3.34}$$

$$HXY1 = -\sum_{i=1}^{N}\sum_{j=1}^{N} p(i,j)\cdot\log_2(p_x(i)\cdot p_y(j)) \tag{3.35}$$

$$HXY2 = -\sum_{i=1}^{N}\sum_{j=1}^{N} p_x(i)\cdot p_y(j)\cdot\log_2(p_x(i)\cdot p_y(j)) \tag{3.36}$$

where:

$p(i,j) = (i,j)^{th}$ entry is a GLCM normalized matrix and N is the maximum gray intensity level (in this dissertation it is 255). Thus, Table 3.3 describes the total texture and color features used in this section to classify the terrain under study.

Table 3.3: Calculation of 26 textural features.

Feature	Computation
Auto-correlation	$\sum_{i=0}^{N-1} \sum_{j=0}^{N-1} ij\ p(i,j)$
Cluster Prominence	$\sum_{i=0}^{N-1} \sum_{j=0}^{N-1} ((i-\mu)+(j-\mu))^4\, p(i,j)$
Cluster Shade	$\sum_{i=0}^{N-1} \sum_{j=0}^{N-1} ((i-\mu)+(j-\mu))^3\, p(i,j)$
Cluster Tendency	$\sum_{i=0}^{N-1} \sum_{j=0}^{N-1} ((i-\mu)+(j-\mu))^2\, p(i,j)$
Contrast	$\sum_{i=0}^{N-1} \sum_{j=0}^{N-1} \lvert i-j\rvert^2 \cdot p(i,j)$
Correlation	$\sum_{i=0}^{N-1} \sum_{j=0}^{N-1} \frac{(i-\mu_x)(j-\mu_y)\cdot p(i,j)}{\sigma_x \cdot \sigma_y}$
Difference Average	$\sum_{k=0}^{N-1} k \cdot p_{i-j,k}$
Difference Entropy	$-\sum_{k=0}^{N-1} p_{i-j,k} \cdot \log_2(p_{i-j,k})$
Difference Variance	$\sum_{k=0}^{N-1} (k - DifferenceAverage)^2 \cdot p_{i-j,k}$
Energy	$\sum_{i=0}^{N-1} \sum_{j=0}^{N-1} p(i,j)^2$
Entropy	$-\sum_{i=0}^{N-1} \sum_{j=0}^{N-1} p(i,j) \cdot \log_2(p(i,j))$
Homogeneity	$\sum_{i=0}^{N-1} \sum_{j=0}^{N-1} \frac{p(i,j)}{1+\lvert i-j\rvert}$
Homogeneity Moment	$\sum_{i=0}^{N-1} \sum_{j=0}^{N-1} \frac{p(i,j)}{1+\lvert i-j\rvert^2}$
Homogeneity Moment Normalized	$\sum_{i=0}^{N-1} \sum_{j=0}^{N-1} \frac{p(i,j)}{1+\frac{\lvert i-j\rvert^2}{N^2}}$
Homogeneity Normalized	$\sum_{i=0}^{N-1} \sum_{j=0}^{N-1} \frac{p(i,j)}{1+\frac{\lvert i-j\rvert}{N}}$
Inverse Variance	$\sum_{i=0}^{N-1} \sum_{j=0}^{N-1} \frac{p(i,j)}{1+\lvert i-j\rvert^2},\ i \neq j$
Informational Measure of Correlation 1	$\frac{HXY-HXY1}{max(HX,HY)}$
Informational Measure of Correlation 2	$\sqrt{1-e^{-2\cdot(HXY2-HXY)}}$
Joint Average	$\sum_{i=0}^{N-1} \sum_{j=0}^{N-1} i\ p(i,j)$
Joint Maximum	$max(p(i,j))$
Kurtosis	$\frac{\frac{1}{N}\cdot\sum_{i=0}^{N-1}(x_i-\mu_x)^4}{\left(\frac{1}{N}\cdot\sum_{i=0}^{N-1}(x_i-\mu_x)^2\right)^2}$
Skewness	$\frac{\sum_{i=0}^{N-1}(x_i-\mu_x)^3}{N\cdot\sigma_x^3}$
Sum Average	$\sum_{k=2}^{2N} k \cdot p_{i+j,k}$
Sum Entropy	$-\sum_{k=2}^{2N} p_{i+j,k} \cdot \log_2(p_{i+j,k})$
Sum Variance	$\sum_{k=2}^{2N} (k - SumAverage)^2 \cdot p_{i+j,k}$
Variance	$\sum_{i=0}^{N-1} \sum_{j=0}^{N-1} (i-\mu)^2 \cdot p(i,j)$

where:

- **Auto-correlation:** Measurement of the coarseness of texture and the magnitude of fineness;
- **Joint Average:** Is the gray level weighted sum of joint probabilities;
- **Cluster Prominence:** Measures the GLCM and GLRLM asymmetry. Higher values indicate more asymmetry whereas center values indicate the peak of the distribution centred around the mean;
- **Cluster Shade:** Measures skewness of the GLCM and GLRLM matrixes. Higher values indicate asymmetry;
- **Cluster Tendency:** Measures the heterogeneity that places higher weights on neighboring intensity level pairs further from the mean;
- **Contrast:** Used to return the intensity level between a pixel and its neighbor throughout the entire image;
- **Correlation:** Important method to define how a pixel is correlated with its neighbor throughout the entire image;

- **Difference Average:** Measures the mean of the gray level difference distribution of the current frame;
- **Difference Entropy:** Measures the disorder related to the to the current frame's gray level difference distribution;
- **Entropy:** This feature measures the randomness of intensity distribution. The greater the information's heterogeneity in an image, the higher the entropy value is. However, when homogeneity increases, the entropy tends to 0;
- **Homogeneity:** Known as Inverse Difference Moment, this equation returns 1 when the GLCM is uniform (diagonal matrix);
- **Homogeneity Moment:** This measures the local homogeneity of an image. The result is a low homogeneity moment value for heterogeneous images, and a relatively higher value for homogeneous images;
- **Homogeneity Moment Normalized:** This has the same purpose as the homogeneity moment however its neighboring intensity values are normalized by higher intensity value with the power of two;
- **Homogeneity Normalized:** This has the same goal as the homogeneity moment but the neighboring intensity values are normalized by the highest intensity value;
- **Sum Average:** This measures the mean of the intensity level sum distribution of the current frame;
- **Sum Entropy:** This measures the disorder related to the intensity value sum distribution of the current frame;
- **Sum Variance:** This measures the dispersion of the intensity number sum distribution of the current frame;
- **Variance:** Represents the measurement of the values' dispersion around the mean.

For the system to have more information, not only the textural information but also the terrain color being studied, this section uses HSV method.

Figure 3.47 shows one way to represent this color model. The Hue component has a certain color defined by an angle (therefore, values between 0 and 360[2]). The Saturation and Value components usually correspond to the range [0, 255] or [0, 1]. The smaller the saturation value is, the grayer the image becomes. Regarding the Value, the higher its value, the brighter the image will be.

Thus, for RGB to HSV conversion images the following equations (3.37, 3.38 and 3.39) were used:

[2]Values in degrees. The range can also be represented in radians $[0, 2\pi]$ or values between [0, 1] if the RGB channels are divided by 255 (max byte value)

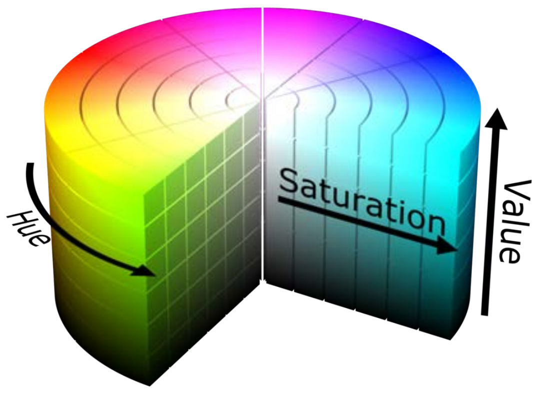


Figure 3.47: HSV color model.

$$Hue = \begin{cases} 0 & if \ Max = Min \\ 60 \cdot \dfrac{G-B}{Max-Min} & if \ Max = R \ \& \ G \geq B \\ 60 \cdot \dfrac{G-B}{Max-Min} + 360 & if \ Max = R \ \& \ G < B \\ 60 \cdot \dfrac{B-R}{Max-Min} + 120 & if \ Max = G \\ 60 \cdot \dfrac{R-G}{Max-Min} + 120 & if \ Max = B \end{cases} \tag{3.37}$$

$$Saturarion = \frac{Max - Min}{Max} \tag{3.38}$$

$$Value = Max \tag{3.39}$$

Where R, G and B correspond to the red, green and blue (respectively) pixel components, Min is the three channels' minimum value (Min(R,G,B)) and Max is the three channels' maximum value (Max(R,G,B)).

To detail how the GLCM matrix was applied in this dissertation, Figure 3.48 presents the proposed system model for terrain classification. It is possible to observe four main processes identified in the architecture, namely:

- **Rectified Image:** As mentioned in Section 3.2.3, the camera needs to be calibrated before any process to make the proposed algorithms universal. Section 3.1.1 describes how to calibrate these RGB cameras;

- **GLCM:** In order to extract the terrain's static textures, GLCM was used to calculate features capable of providing information to classify terrain types;

- **Frames > n:** "n" is the total number of frames required to increase the algorithm's accuracy. This threshold value was also chosen empirically by the author.

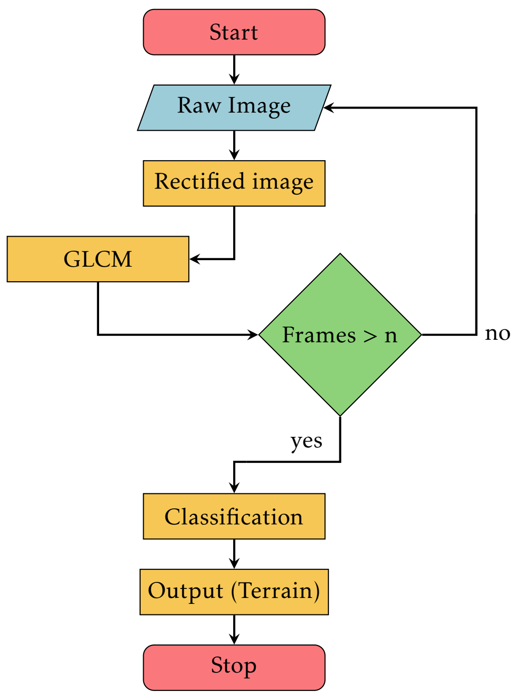


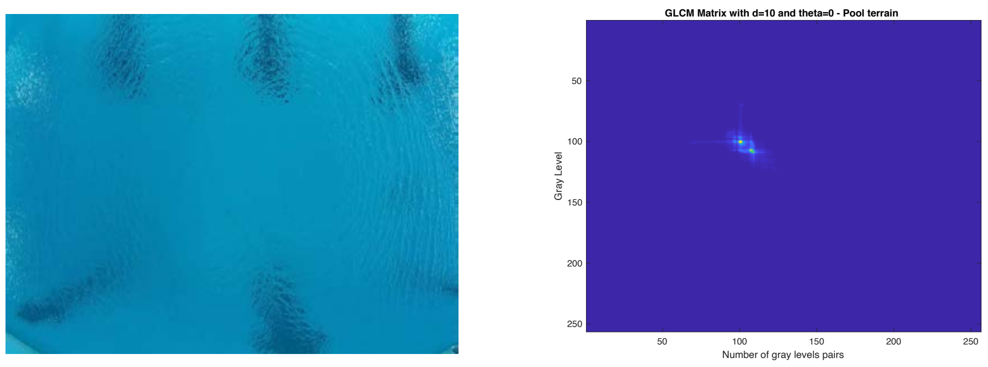
Figure 3.48: GLCM proposed system model.

- **Classification:** The outputs generated by the GLCM phase are turned into inputs for a Neural Network (NN). In this section, a Multilayer Perceptron (MLP) architecture was used. The design of the NN is the same as in previous sections: the Neural Network model consists of the hidden layer with 10 neurons, and the third layer corresponds to the system output and has two possible outputs: water (1) or non-water (0).

To test the GLCM and its respective features, four different terrain types (water, vegetation, asphalt and sand) were analysed. The dataset used in this section is the same as in Sections 3.2.3 and 3.2.4. The following five Figures (Figures 3.49, 3.50, 3.51, 3.52, 3.53) show GLCM results from each terrain studied in this section. It is possible to conclude that by implementing the GLCM algorithm, vegetation, asphalt and sand terrains were found to have a higher color variation than water-type terrains, which can be decisive to differentiate between water and non-water terrain types. These results will be studied in detail in Section 4.5.

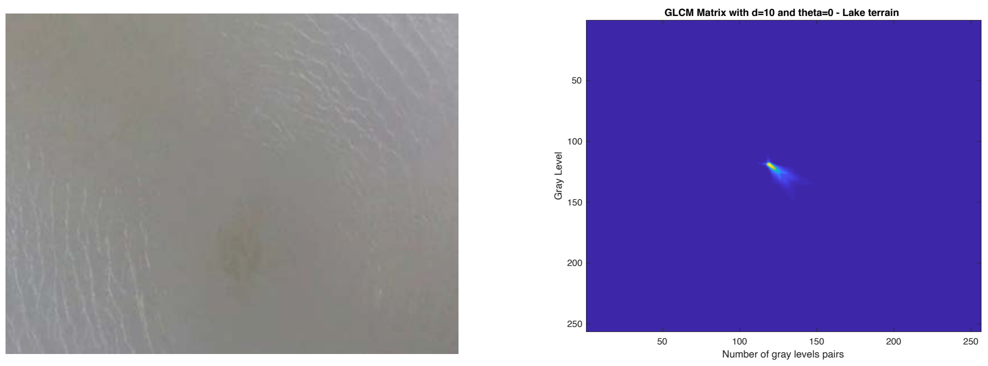
3.2.6 GLRLM

After explaining the operation and presenting the GLCM results in Section 3.2.5 for TC and shape dependency problems, the following section introduces both the second statistical texture order and the sixth algorithm used in this dissertation, in order to classify the terrains under study: The Gray-Level Run Length Matrix (GLRLM).



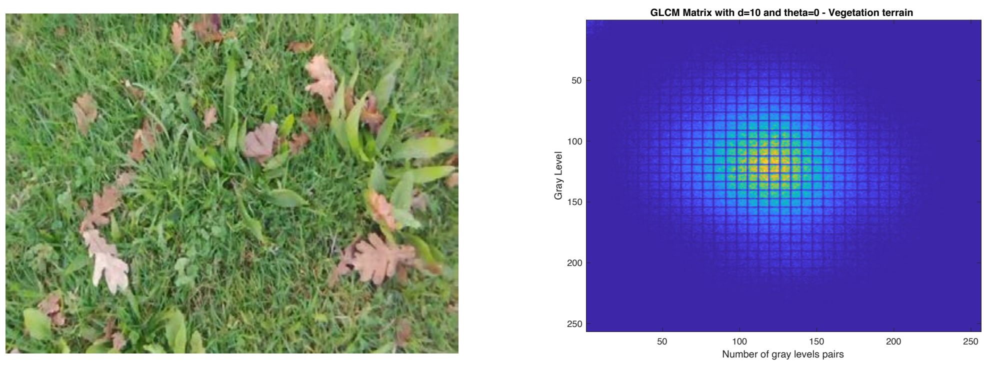
a b

Figure 3.49: a) Input image (water terrain A); b) its GLCM result.



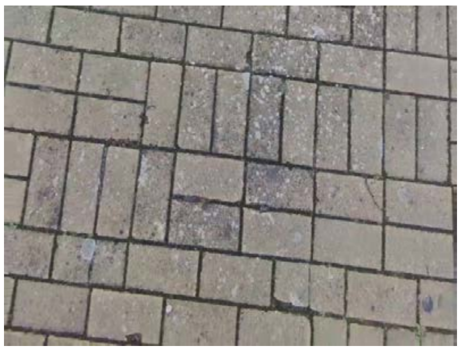
a b

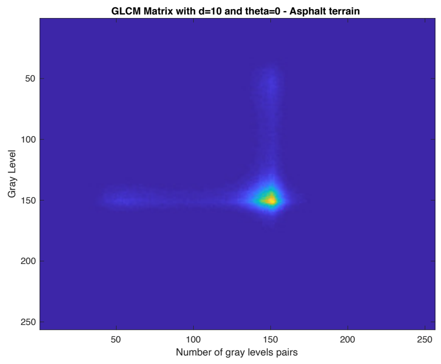
Figure 3.50: a) Input image (water terrain B); b) its GLCM result.



a b

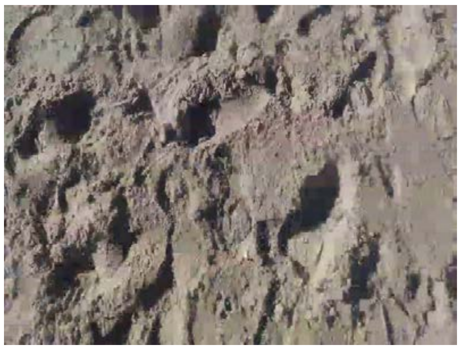
Figure 3.51: a) Input image (vegetation terrain); b) its GLCM result.

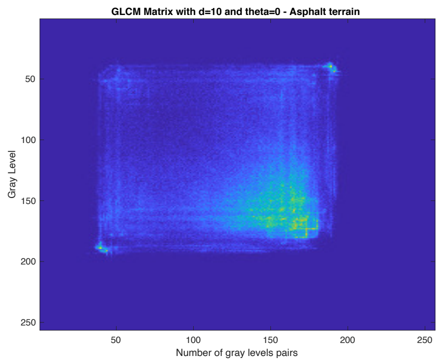



GLCM Matrix with d=10 and theta=0 - Asphalt terrain
Gray Level
Number of gray levels pairs
50
100
150
200
250

a b

Figure 3.52: a) Input image (asphalt terrain); b) its GLCM result.




GLCM Matrix with d=10 and theta=0 - Asphalt terrain
Gray Level
Number of gray levels pairs
50
100
150
200
250

a b

Figure 3.53: a) Input image (sand terrain); b) its GLCM result.

The GLRLM is a matrix which attempts to quantify runs of the same grey level in an image. This algorithm is set up slightly differently from the GLCM; instead of having grey levels along the table's abscissa, the GLRLM has run lengths for the terrain image under study.

Regarding to this section, and as described in Section 2.3.5, two main steps are considered:

1. GLRLM mask design;
2. Calculation of 27 features.

As a first step, it is necessary to create the GLRLM NxN matrix. In this dissertation, and as mentioned in Section 2.3.5, the input images are 8Bit ranging between 0 and 255 levels (256 gray levels).

Furthermore, a $\theta = 0$ was chosen in the present dissertation (this value was obtained by trial and error).

Listing 3.11: GLRLM mask design.

```
void glrlm(img, xi, yi, xf, yf){
    // Define the variables
    int sizeGlrlmMask = 256;
    glrlmArray [sizeGlrlmMask][sizeGlrlmMask];
    rows = yf;
    cols = xf;
    pixelPtrRead = imageGray.data;

    // GLRLM matrix
    for (y = yi; y < rows; ++y) {
        // Get the first pixel column with Y row.
        auto value = pixelPtrRead(y,0)[0];
        auto pixelLength =1;

        for (x = xi+1; x < cols; ++x) {
            // Check if it is the last column pixel to save the length ↩
                result
            if((x+1) == cols){
                glrlm[value][pixelLength]++;
                break;
            }
            // Increase the run length if the next pixel is the same that ↩
                the current one
            if (value == pixelPtrRead(y,x)[0]) {
                pixelLength++;
            // If not, save the length result and get the next pixel value
            } else {
                glrlm[value][pixelLength]++;
                value = pixelPtrRead(y,x)[0];
                pixelLength = 1;
            }
        }
    }
}
```

Listing 3.11 indicates how the GLRLM matrix has been created. Note that the number of columns in the GLRLM matrix is subjective since it differs from case to case. This dissertation intends to have the maximum run length, so the image's width was considered equal to the number of GLRLM matrix columns.

Figure 3.54 shows an example of water terrain results using the GLRLM algorithm. In this case, it is possible to observe a greater length in the pixel of intensity equal to 150 (average value of the three channels) in relation to the other pixel values. This is due to the fact that the water-type terrains are practically homogeneous.

With a GLRLM matrix defined, the same features in Section 3.2.5 were also used in this section: 26 textural features (Table 3.3) and one intensity colour (equation 3.37) were extracted to classify the terrain.

Thus, in order to explain how the GLRLM matrix was applied in this dissertation, the proposed system model, for terrain classification, is illustrated in Figure 3.55.

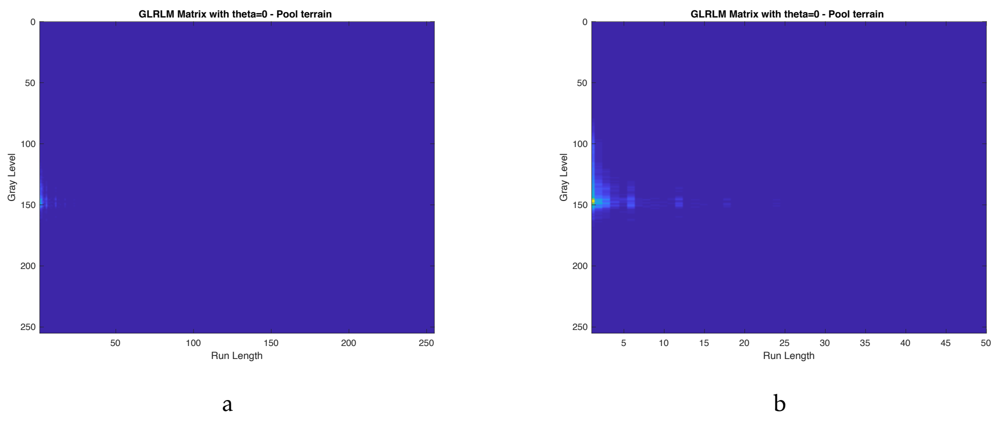


Figure 3.54: GLRLM - Result from pool (water) terrain type. b) is a zoom of a).

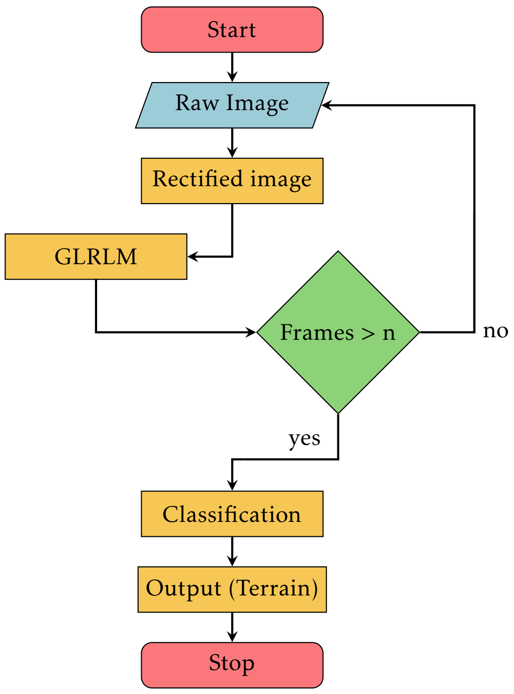
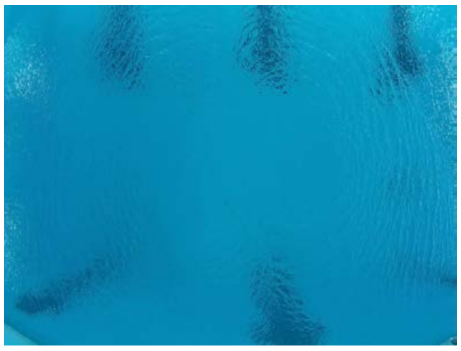


Figure 3.55: GLCM proposed system model.

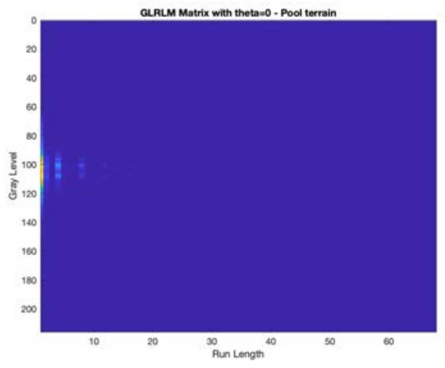
From the system model proposed in 3.55, it is possible to observe four main processes in the architecture, namely:

- **Rectified Image:** As mentioned in Section 3.2.3, it is necessary to calibrate the camera before any process to make the proposed algorithms universal. Section 3.1.1 described how to calibrate these RGB cameras;
- **GLRLM:** In order to extract the terrain's static textures, as with GLCM, GLRLM was also used to calculate features capable of providing information to classify terrain types;
- **Frames > n:** "n" is the total number of frames required to increase the algorithm's accuracy. This threshold value was also chosen empirically by the author.
- **Classification:** The outputs generated by the GLRLM phase are turned into inputs for a Neural Network (NN). In this section, a Multilayer Perceptron (MLP) architecture was used. The design of the NN is the same as in Section 3.2.5.

The system model having been explained, in this section, four different terrain types (water, vegetation, asphalt and sand) were analysed. The dataset used in this section is the same as in Section 3.2.5. The following five Figures (Figures 3.56, 3.57, 3.58, 3.59, 3.60) show the GLRLM results for each terrain under study in this section. It is possible to conclude that by implementing the GLRLM algorithm, vegetation and sand terrains were found to have higher variation values than water-type terrains. However, asphalt terrain type has similar variation values to water-type terrain. Therefore an NN is used to help with this issue. The GLRLM algorithm and its results will be studied in depth in Section 4.6.

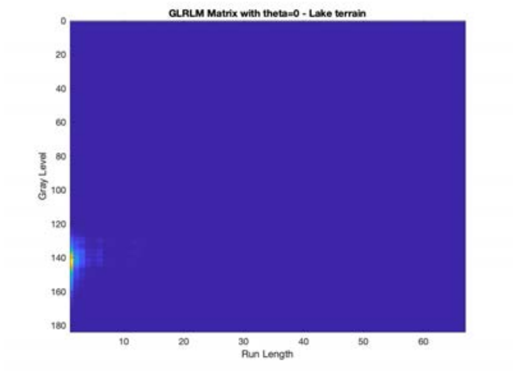


a



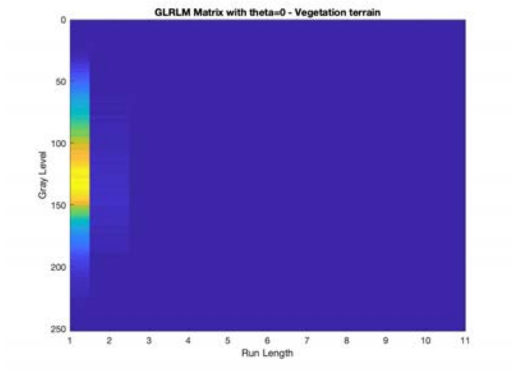
b

Figure 3.56: a) Input image (water terrain A); b) its GLRLM result.



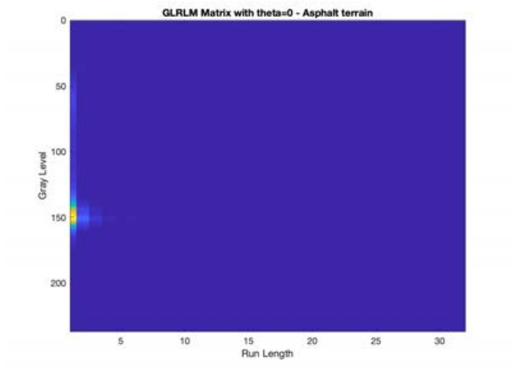
a b

Figure 3.57: a) Input image (water terrain B); b) its GLRLM result.



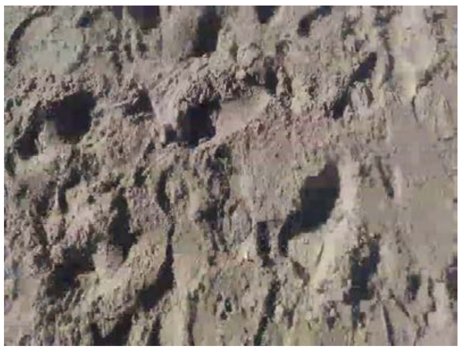
a b

Figure 3.58: a) Input image (vegetation terrain); b) its GLRLM result.

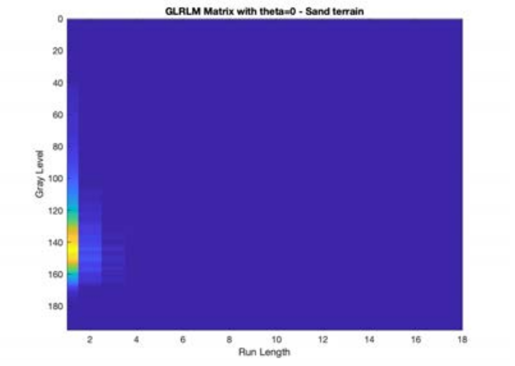


a b

Figure 3.59: a) Input image (asphalt terrain); b) its GLRLM result.



a



b

Figure 3.60: a) Input image (sand terrain); b) its GLRLM result.

3.3 Dynamic Feature Extraction for Terrain Classification

The ability to detect static textures for terrain classification is important. However, the ability to distinguish terrains (water and non-water terrains) using motion analysis increases the accuracy of the system. Therefore, several features of different terrains will be studied to support the terrain classification.

In this section, the downwash effect from the UAV propellers was studied to extract dynamic textures from water terrain (circular movements), which can be used to differentiate it from any other type of terrain being studied (vegetation, asphalt and sand).

Different types of terrain create different behaviors when affected by the UAV's downwash effect. Whereas in water (in this dissertation lake and pool terrains, ocean and sea being excluded) it makes a circular pattern, vegetation terrain produces a linear movement and asphalt/sand are almost static. This movement can be detected using a well known concept called Optical Flow (OF), as mentioned in Section 2.4.

The current section will study the dynamic part (motion analysis) of the terrain, focusing on water-type terrain, comparing it with other non-water terrain types. The OF concept, more specifically in the algorithm developed by Farneback (Farnebäck, 2003), will be the basis of this dynamic texture study. The dynamic study is then divided into two sub-sections (two algorithms are proposed):

1. Travel Distance;
2. Circular Motion.

3.3.1 Travel Distance

Water-type terrain only exhibits dynamic texture when exposed to the downwash effect. Figures 3.61 a), b), c) and d) show the textural information with and without the downwash effect, respectively. However, in spite of having a dynamic texture, the optical flow is never stronger than the dynamic observed for sand, asphalt and vegetation terrain types. As mentioned in Section 2.4, the OF method can calculate the distance travelled by block matching features in a given frame sequence, n, captured by the UAV's downward-looking camera. These images are captured at a 30 frame per second rate. In this section, the Farneback algorithm (Farnebäck, 2003) was used to detect the movement of these features and return a flow matrix, as shown in Listing 3.12. Then the TD will be used to characterize the dynamic texture.

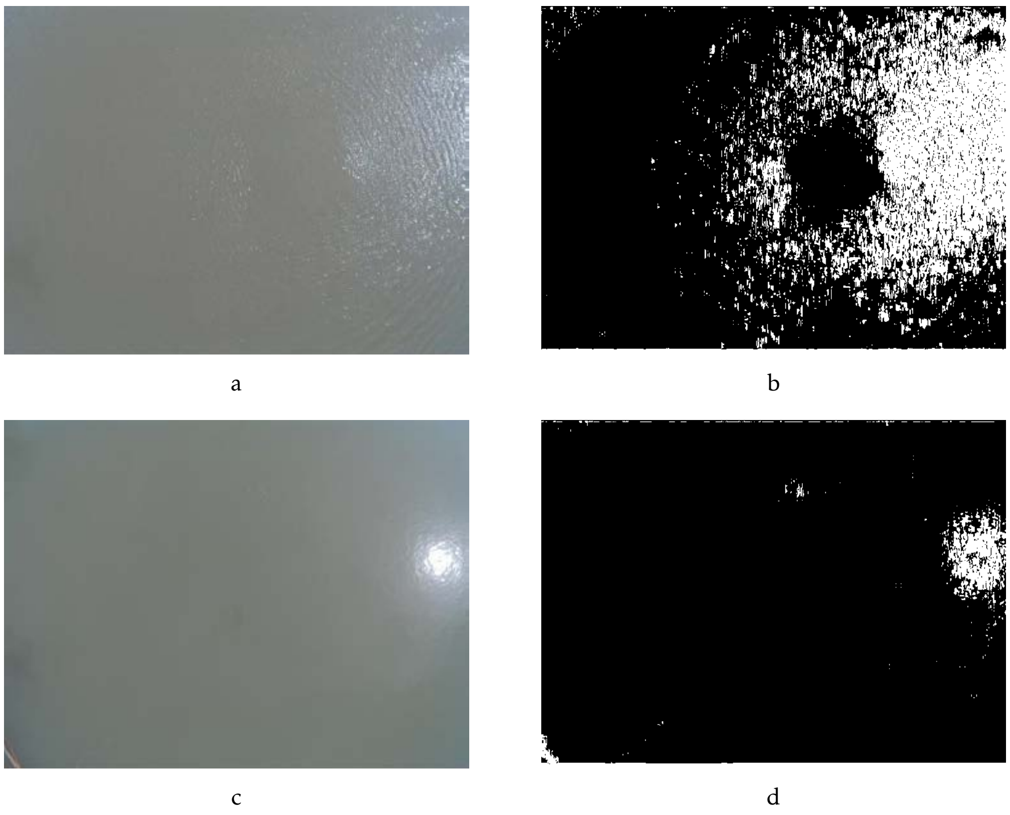


Figure 3.61: Downash effect: a) and b) Textural information with the downwash presented; c) and d) Textural information without the downwash effect.

Listing 3.12: Farneback Algorithm.

```
opticalFlowFarneback(prevImage, nextImage){
    // Farneback algorithm calculation
    flow=calcOpticalFlowFarneback(prevImageGray, nextImageGray, flowUmat);

    // Return the sequence frames' flows
    return flow;
}
```

After obtaining the flow matrix, a set a vectors is used (trackers) in these n frames to show the flows, as shown in Listing 3.13. As can be observed in Listing 3.13, the trackers are initialized over the resulting frame, previously resized to a 640×480 image, with a

15 pixel width and height distance. The resulting matrix and flows can be observed in Figure 3.62 b), forming a uniform grid covering the whole image.

Listing 3.13: Flow matrix visualization.

```
visualizeFarnebackMatrix(resultImage){
    // Define the variables
    featuresStepHist = 15;
    width  = prevImage.cols;
    height = prevImage.rows;
    Matrix selectedFlows;

    // Only the selected flows are sent into a new matrix
    for (y=0; y < height; y += featuresStepHist) {
        for (x = 0; x < width; x += featuresStepHist) {
            Point flowatxy = resultImage(y, x);
            selectedFlows=line((x, y), (x+ flowatxy.x, y+ flowatxy.y), ↩
               blueColor);
            selectedFlows=circle((x, y), blackColor);
        }
    }
}
```

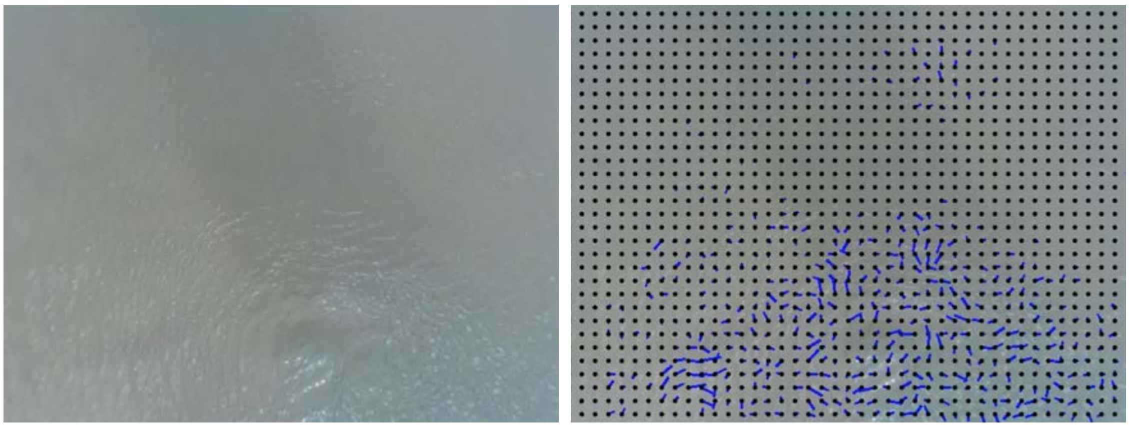


Figure 3.62: Example of an Optical Flow concept using the Farneback algorithm: a) Current frame; b) Optical flow result using the Farneback algorithm (flows in blue).

After having computed the OF, the next step is to calculate the distance travelled (trajectory) by each feature in a sequence of frames (equation 3.40).

$$Travel_{distance} = \sum_{i=2}^{n} \sqrt{A_x(i) + B_y(i)} \tag{3.40}$$

where:

$$A_x(i) = \left[x_1 - x_{i-1} + F_{dx}\right]^2 \tag{3.41}$$

$$B_y(i) = \left[y_1 - y_{i-1} + F_{dy}\right]^2 \tag{3.42}$$

and x_i and y_i are the positions in x and y in the most recent frame (n), x_1 and y_1 are the initial positions ($n = 1$) and F_{dx} and F_{dy} are the flow displacements between frames n and $n-1$. The normalized x and y coordinates were used for the calculations.

To eliminate features that did not move or were almost static in a sequence of frames, a thresholding was imposed:

$$\begin{cases} Red & for\ Trajectory \geq Threshold \\ Nothing & for\ Trajectory \leq Threshold \end{cases} \tag{3.43}$$

From equation 3.43 and knowing the maximum number of features (the number of features in rows and columns in the image is predefined), the percentage of Number of Features (NOF) that appear in the image (equation 3.44) is calculated. An example showing the TD algorithm and its result is illustrated in Figure 3.63.

$$NOF = \frac{filtered\ features}{Total\ features} \cdot 100\% \tag{3.44}$$

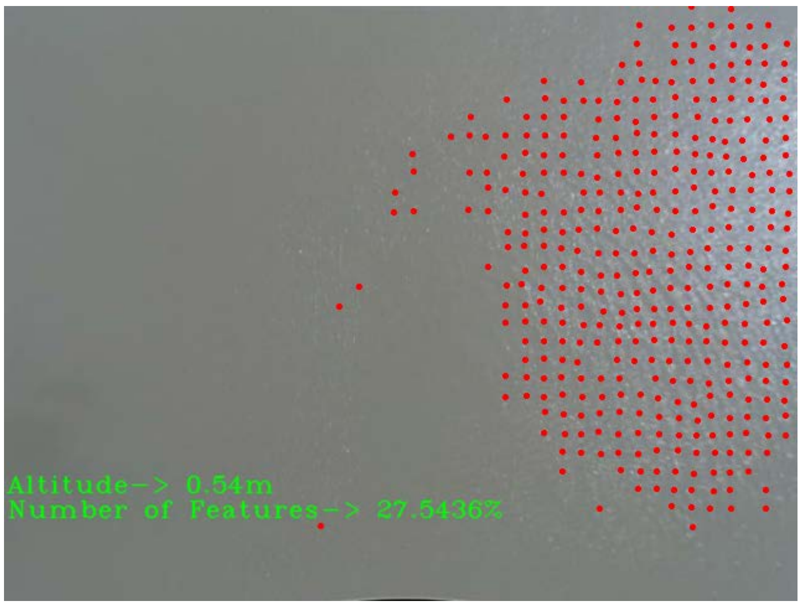


Figure 3.63: Dynamic texture detection by Farneback algorithm and distance travelled/number of features calculation on water-type terrain.

In order to explain how the TD algorithm was applied in this dissertation, the proposed system model, for terrain classification is presented in Figure 3.64. It is possible to observe four main processes in the architecture, namely:

- **Rectified Image:** Before starting to analyze each frame from the RGB cameras, it is necessary to perform the pre-processing step. This step is necessary because all RGB

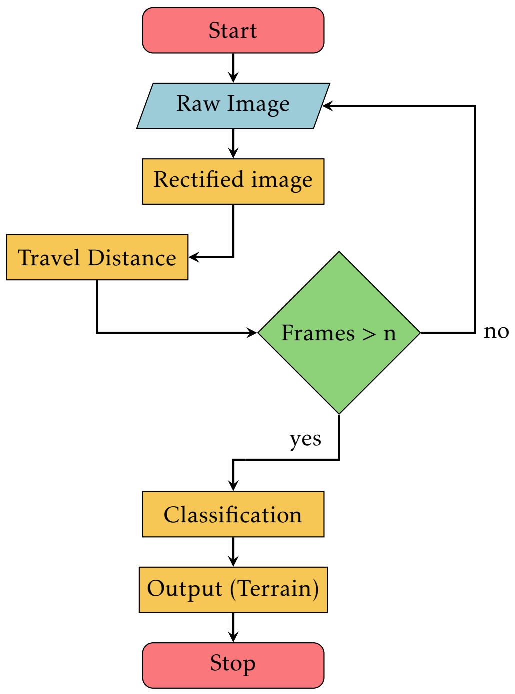


Figure 3.64: TD proposed system model.

cameras need to be compatible with all developed algorithms in this dissertation. Section 3.1.1 describes how to calibrate these RGB cameras;

- **Travel Distance:** To measure the trajectory of each tracker, this phase uses the TD algorithm and outputs the NOF calculation;
- **Frames > n:** As mentioned before, "n" is the total number of frames required to increase the algorithm's accuracy. In this section, this threshold is equal to three;
- **Classification:** The outputs generated by the TD phase are turned into inputs for a Neural Network (NN). In this section, a Multilayer Perceptron (MLP) architecture was used. The design of the NN is the same as in Section 3.2.6.

The system having been explained, three different terrain types (water, vegetation and sand) were analysed in this section. Figures 3.65, 3.66, 3.67 and 3.68 present different texture outputs from different terrain types. It is possible to conclude that by implementing the TD algorithm, asphalt and vegetation-like terrains were found to have a higher NOF than sand and water-type terrains. On the other hand, water-type terrain, also presents lower NOF than sand-type terrain which can be decisive for differentiating it from other terrain types.

3.3.2 Circular Motion

It has already been shown throughout the dissertation that the downwash effect causes a circular effect on water-type terrains, a behavior that differs from other types of terrain

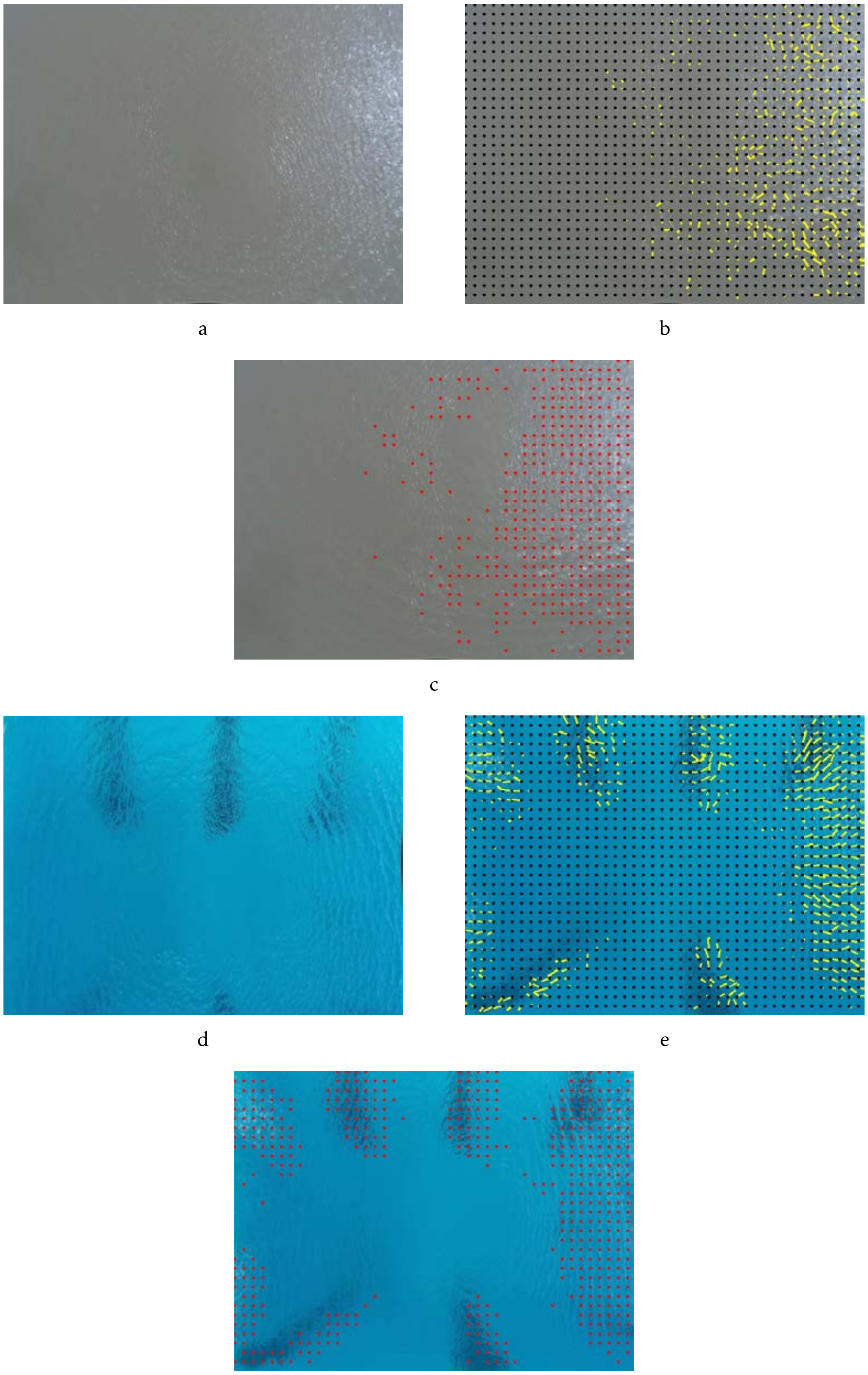
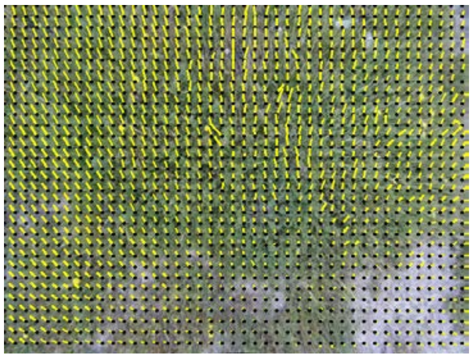
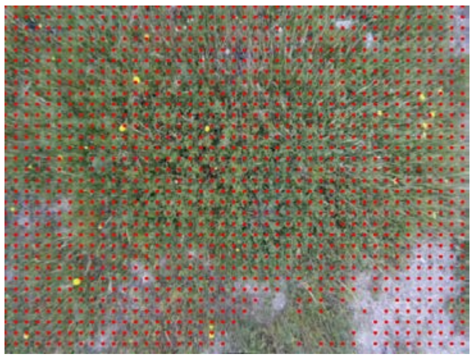


Figure 3.65: Examples of the TD algorithm applied to water-type terrain. (a,d) Input image; (b,e) the optical flow; (c,f) the travel distance calculation.

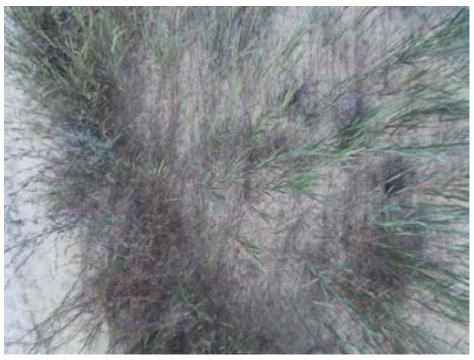
a



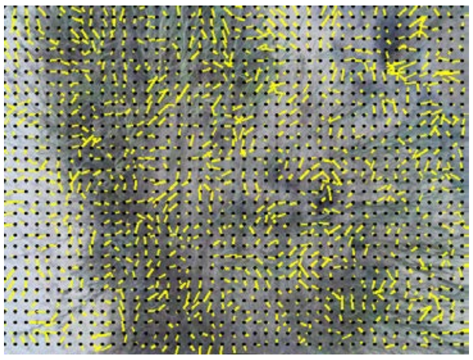
b



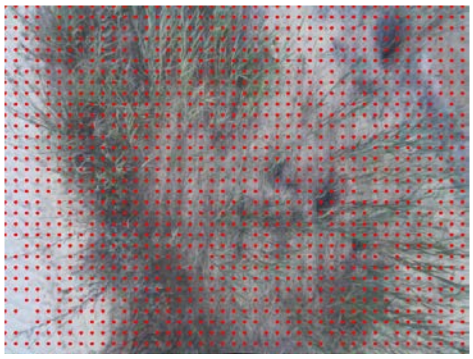
c



d



e



f

Figure 3.66: Examples of the TD algorithm applied to vegetation-type terrain. (a,d) Input image; (b,e) the optical flow; (c,f) the travel distance calculation.

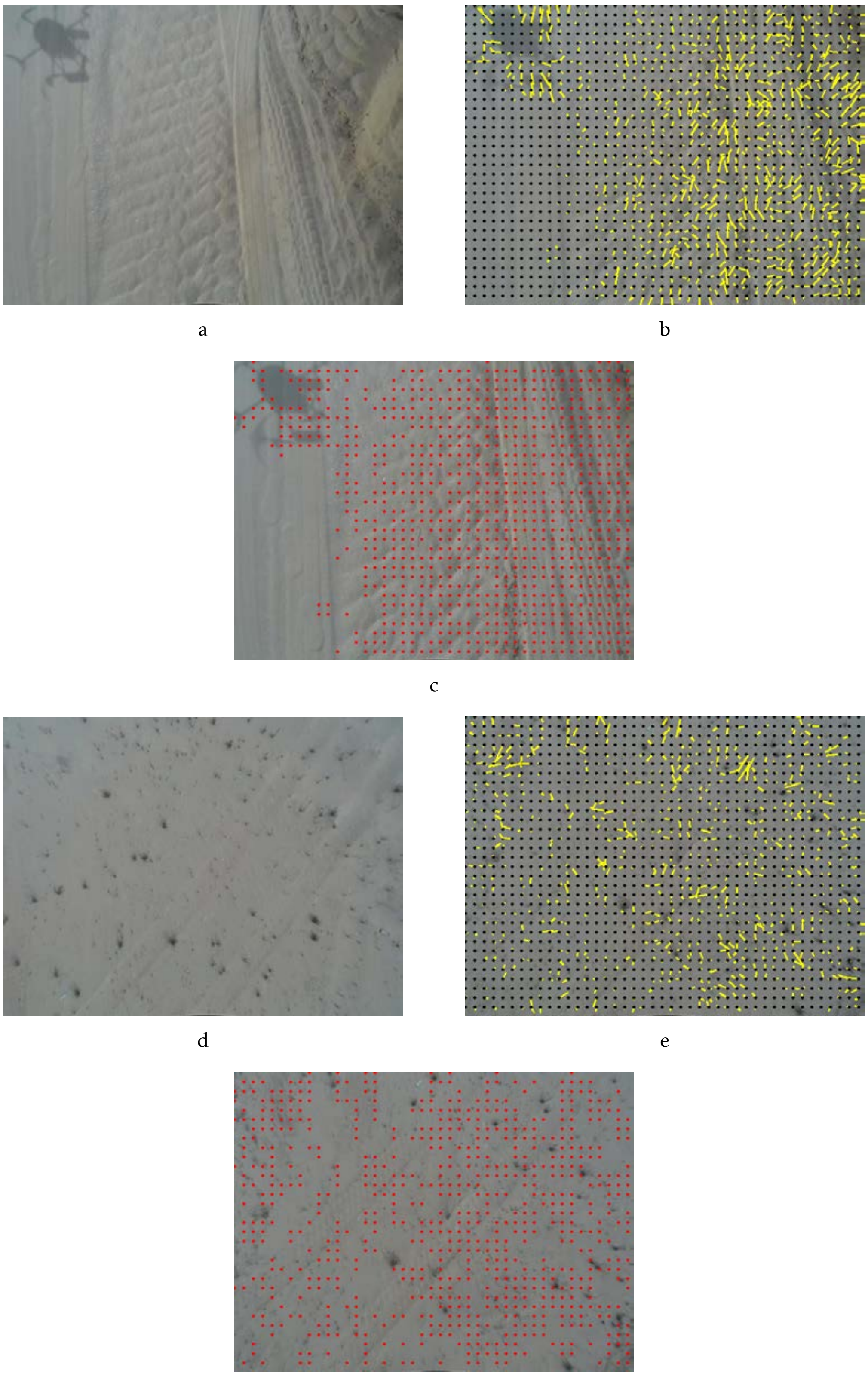


Figure 3.67: Examples of the TD algorithm applied to sand-type terrain. (a,d) Input image; (b,e) the optical flow; (c,f) the travel distance calculation.

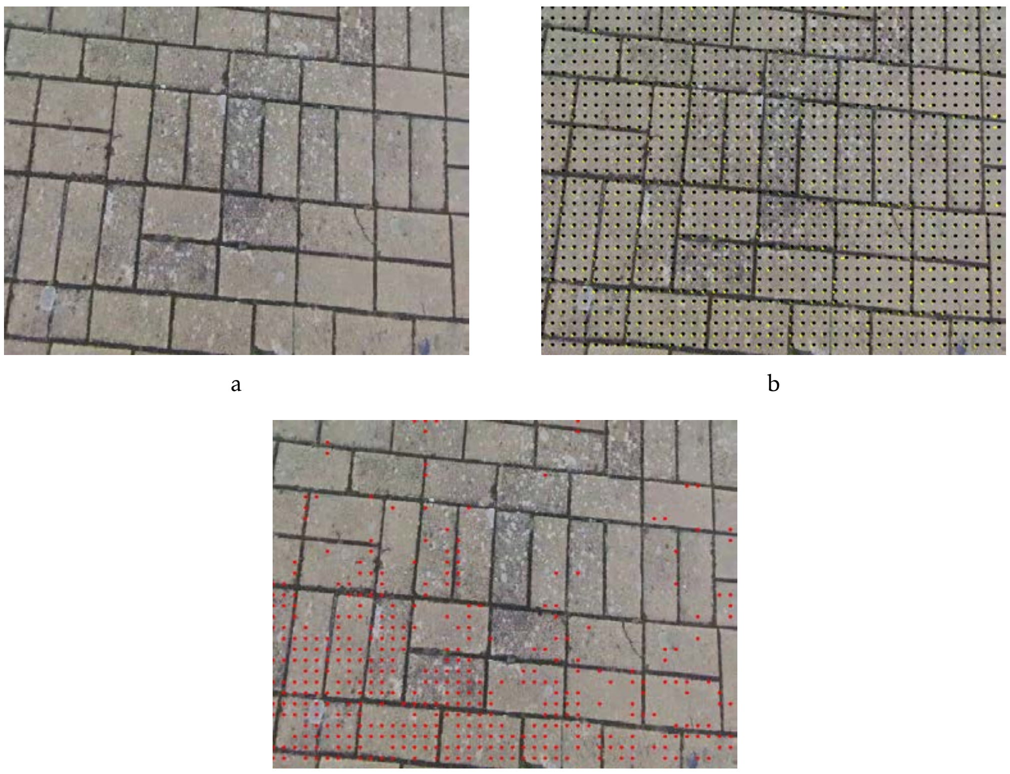


Figure 3.68: Examples of the TD algorithm applied to asphalt-type terrain. (a,d) Input image; (b,e) the optical flow; (c,f) the travel distance calculation.

(vegetation, sand and asphalt). Therefore, this section also analyses the motion of terrains under study (to obtain the circular effect on water-type terrains), where water-type terrain is distinguished from non-water terrain. Thus, the second dynamic algorithm proposed in this dissertation is know as CM.

This algorithm's main steps are the following:

1. Optical Flow - Farneback algorithm;

2. Density Calculation (DC);

3. Flows Orientation (FO).

In this section, the Farneback algorithm (Farnebäck, 2003) was also used to detect textural movement. One of the advantages of using the Farneback algorithm is that it provides the flow displacement, F_d, given by the difference in the features between two frames, while the Lukas Kanade Algorithm needs to have the trackers initialized over the first frame. The flow displacement F_d between features in frame n and $n-1$ can be obtained from Equation 3.45:

$$F_d = S_n - S_{n-1} \tag{3.45}$$

where S_n and S_{n-1} are the sample pixels between two consecutive sequences of frames.

Using the same Listing 3.12 in Section 3.3.1, it is possible to obtain the flow matrix as illustrated in Figure 3.69, where the red arrows are the flows between two consecutive sequences of frames, $n = 1$ (these red arrows are oriented from the center of the downwash to the outside because the terrain under study is water-type terrain).

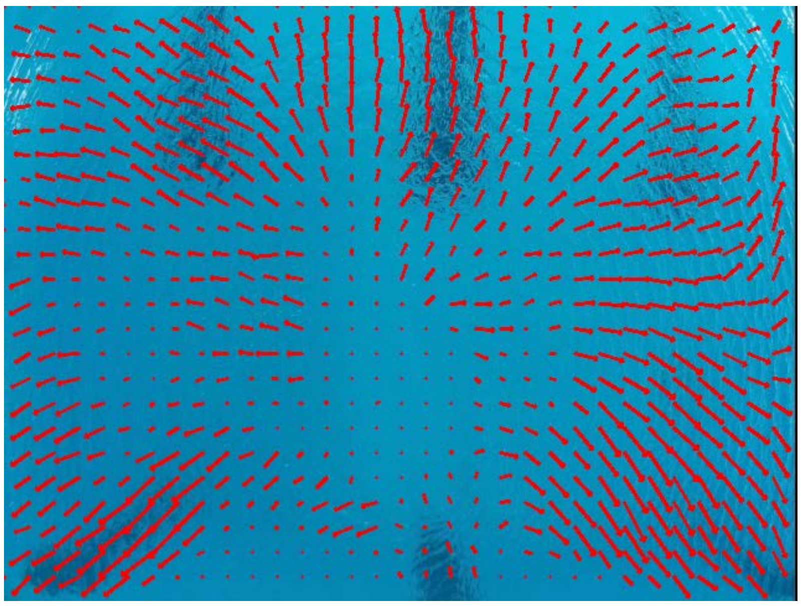


Figure 3.69: Example of an Optical Flow concept using the Farneback algorithm in water-type (pool) terrain.

Then, with the obtained flows, and to determine the density of the obtained flows, the first step is to obtain the linear equation of all flows, as described in Listing 3.14, and display all lines on an image as shown in Figure 3.70.

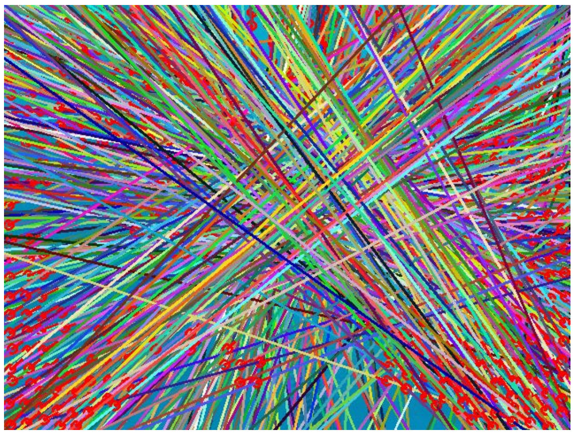


Figure 3.70: Linear equations after flow texture extraction.

After obtaining the linear equations from Listing 3.14, it is now possible to move to the DC step. The goal of this step is to divide each point in each linear equation by a compact value (in this case, the compact value is equal to 20 and it was obtained by a set

Listing 3.14: Linear Equations Calculations.

```
calcLinearEquation(img, xi, yi, xf, yf, blue, green, red){
    // Define the variables
    Vector equationVector;

    // Linear equation color
    Color linearEquationColor = Color( blue, green, red);

    // Slope calculation
    m = (yf-yi)/(xf-xi);

    // Intercept calculation
    b = yf - m*xf;

    // Draw the linear equation line
    img=line((0,b), (width, m*width+b), linearEquationColor);

    // Save slope and intercept into equationVector and return it
    equationVector.add(m,b);

    return  equationVector;
}
```

Listing 3.15: Density Calculation.

```
calcDensity(img, Vector equationVector, compactValue){
    // Define the variables
    width = img.cols;
    height = img.rows;
    desityVector [height][width];

    // Density Calculation by Compact Value
    for (index=0; index < width; ++index) {
        value=round((equationVector.m*index+equationVector.b)/compactValue);
        if(value > 0 && value < height -1)
            desityVector[value][index]++;
    }
}
```

of iterations). Since the slope and intercept values of each linear equation are needed to calculate the density, Listing 3.15 takes Listing 3.14 output as its input argument.

In Figure 3.71 it is possible to observe the influence of dividing the flow density by 20 or when it is not divided.

After computing the flow density diagram, the next step is to find the location of its maximum value (e.g. the highest pixel value), which corresponds to the downwash center

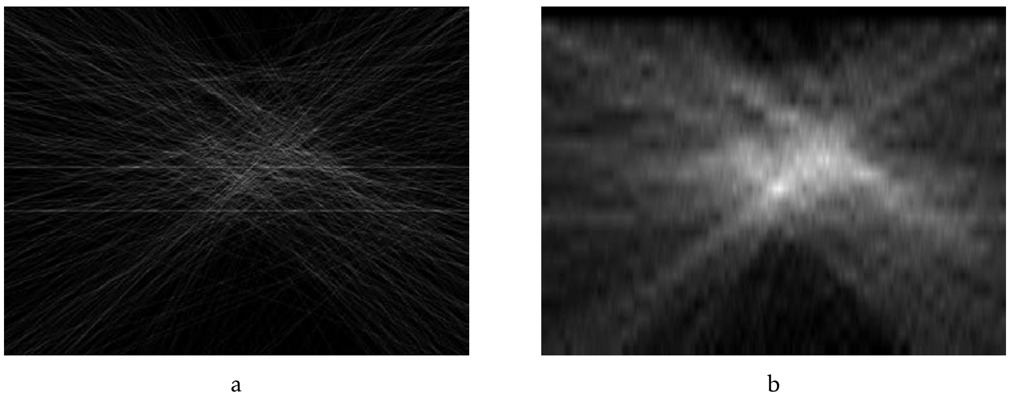


Figure 3.71: Flow density diagram: a) Original; b) Divided by 20.

(Listing 3.16).

Listing 3.16: Downwash Center.

```
calcDownwashCenter(img, desityVector [480][640], compactValue){
    // Define the variables
    width = img.cols;
    height = img.rows;
    maxValue=0;
    centerX=0;
    centerY=0;

    // Downwash Center Calculation
    for (y = 0; y < height; ++y) {
        for (x = 0; x < width; ++x) {
            if(desityVector[y][x] > maxValue){
                maxValue = desityVector[y][x];
                centerX = x;
                centerY = y*compactValue < height-1 ? y*compactValue : ↩
                    height-1;
            }
        }
    }
}
```

Knowing the downwash center and considering Figure 3.69, it will be analysed if the flows (arrows in red) are pointing into or out of the downwash center. In this water-terrain type example, all the flows originated from the Farneback algorithm are expected to point out from the center of the downwash (the circular movement goes from the center of the downwash outwards). Thus, there are 3 possible outputs, when creating a result image:

- **Blue Blocks:** If the flow is pointing out of the downwash center;
- **Green Block:** If the flow is pointing into the downwash center;

- **Gray Block:** If the size of the flow is below a certain given treshold value.

In this section, the images were divided into blocks, to evaluate the flows in groups. Thus, in a 640 by 480 image, each block has a width of 80 (eight blocks) pixels and a height of 96 pixels (five blocks). The result of this example is shown in Figure 3.72.

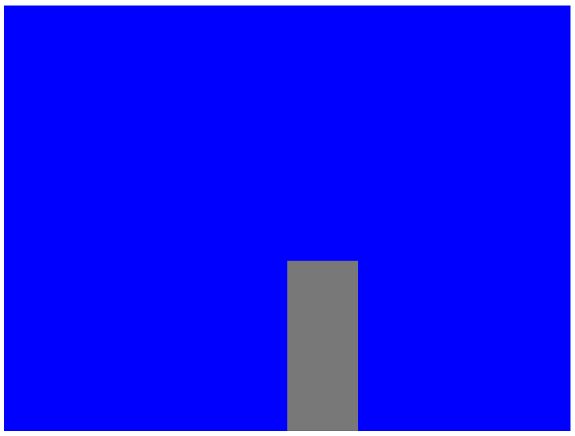


Figure 3.72: Circular motion algorithm result.

Lastly, the output information in Figure 3.72 is sent to the NN to classify the terrain under study.

Thus, in order to explain how the CM algorithm was applied in this dissertation, the proposed system model, for terrain classification, is presented in Figure 3.73. Four main processes were identified in the architecture, namely:

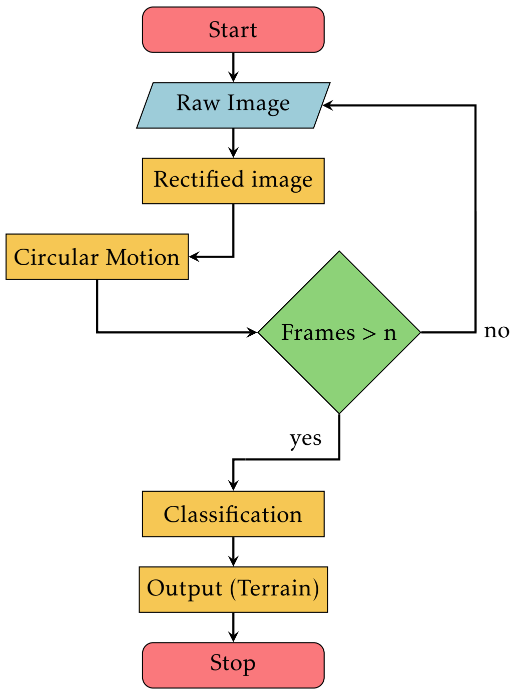


Figure 3.73: TD proposed system model.

- **Rectified Image:** As mentioned in other sections, it is necessary to make the algorithm developed work on any type of RGB camera. Thus, it is necessary to ensure that there is a camera-calibration process. Section 3.1.1 describes in detail about how this calibration occurs;

- **Circular Motion:** In order to measure the circular effect of water-type terrains, this phase uses the concept of optical flow to measure the center of the downwash effect, as well as the flows of each tracker in relation to the center;

- **Frames > n:** As mentioned before, "n" is the total number of frames required to increase the algorithm's accuracy. In this section, this threshold is equal to three;

- **Classification:** The outputs generated by the CM phase are turned into inputs for a Neural Network (NN). In this section, an MLP architecture was used. The design of the NN is the same as in Section 3.2.6: The Neural Network model consists of the hidden layer containing 10 neurons, and the third layer corresponds to the system output and has the number of possible terrain outputs being studied in this dissertation. Each neuron uses a sigmoidal function to calculate its output. They are connected as a Fully Connected Feed Forward Neural Network. During the training stage, 70% of the total data was used for training, 15% for testing and 15% for validation;

Now that the system has been explained, four different terrain types (water, vegetation, sand and asphalt) were analysed in this section. The following three Figures (Figures 3.74, 3.75 and 3.76) illustrate the CM results of each terrain under study in this section. It is possible to conclude that by implementing the CM algorithm, vegetation and asphalt terrains were found to have more gray colored blocks due to the motion of the missing features. Sand terrain has more green than gray colored blocks, due to the sand grains that move more easily than vegetation and asphalt-type terrain. On the other hand, water-type terrain presents more blue colored blocks, which can be decisive for differentiating it from other types of terrain.

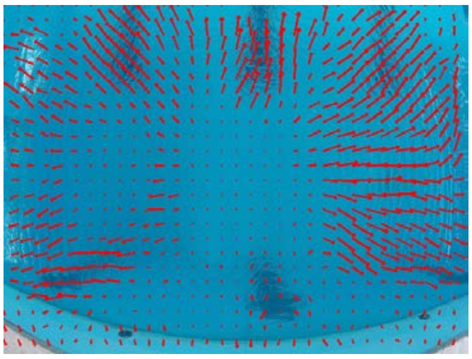
3.4 GPU Acceleration

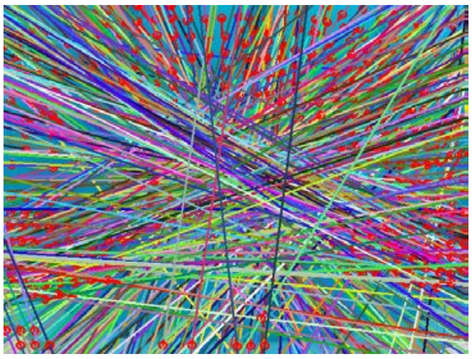
In image processing there are two main problems that are commonly highlighted (Liu et al., 2006):

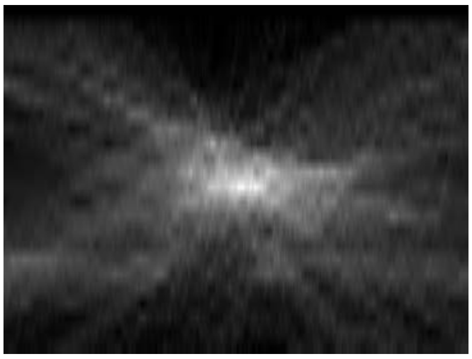
1. The accuracy in decision making;

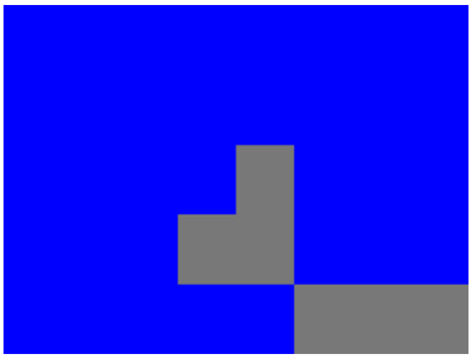
2. The processing time.

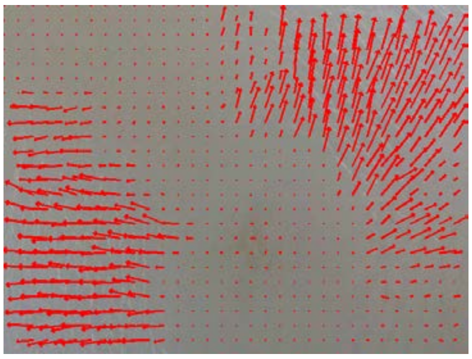
It is important to understand each one's trade-offs and what is most crucial to the problem being examined. Fast response time has become a requirement in the real-time

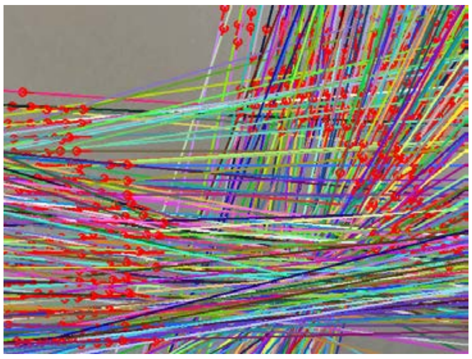

a

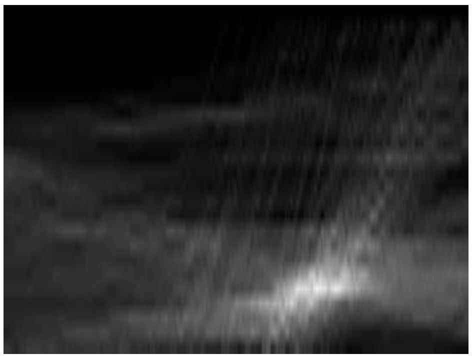

b


c


d


e


f


g

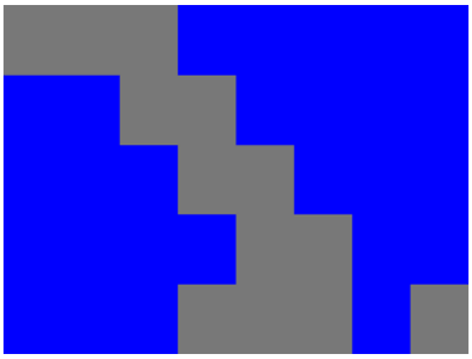

h

Figure 3.74: Circular motion algorithm applied to water-like-terrain (pool and lake): a) and e) Optical Flow concept with the result of the Farneback algorithm; b) and f) Flows interceptions; c) Flow density diagram; d) and h) Classification result.

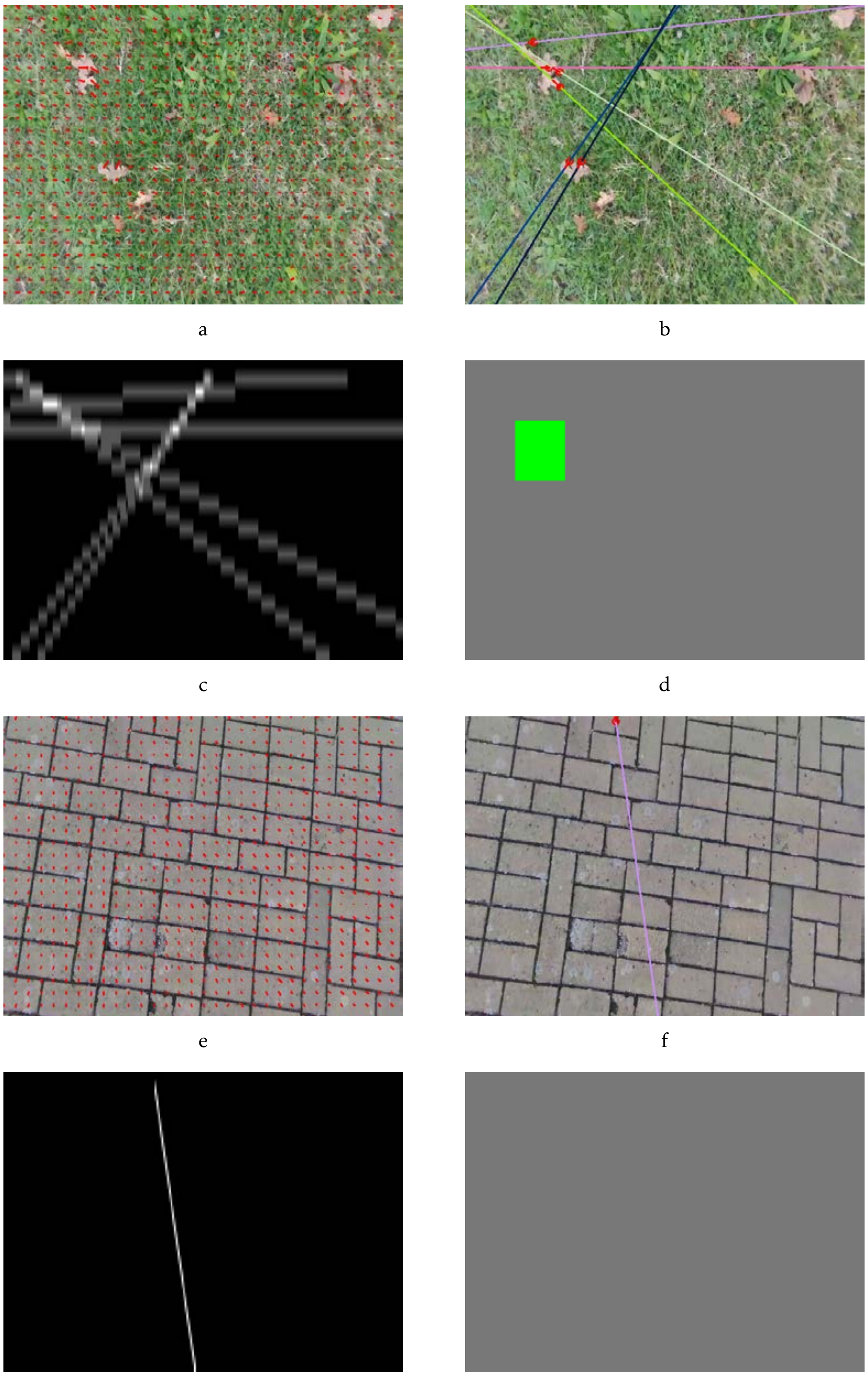


Figure 3.75: Circular motion algorithm applied to non-water-type terrain (vegetation and asphalt): a) and e) Optical Flow concept with the result of the Farneback algorithm; b) and f) Flows interceptions; c) Flow density diagram; d) and h) Classification result.

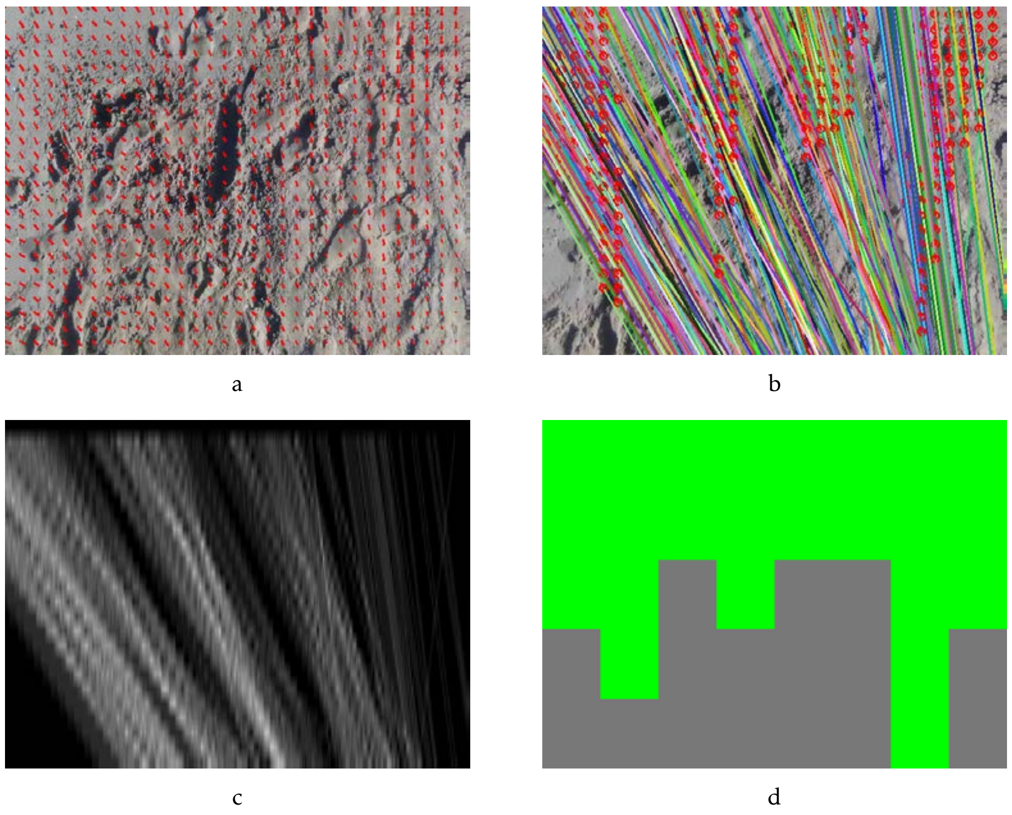


Figure 3.76: Circular motion algorithm applied to non-water-type terrain (sand): a) Optical Flow concept with the result of the Farneback algorithm; b) Flows interceptions; c) Flow density diagram; d) Classification result.

system world, as systems have become larger and more complex over time, with most use being in human and robot interaction and imaging hardware (Jung et al., 1995). In this case study, greater reliability of the algorithms is desired, compromising the execution speed and time response.

In this dissertation, there are two objectives: not only to produce a system capable of classifying and differentiating water-type and non-water-type terrains, but also a fast system for image processing. Thus, the algorithms have been optimized for acceleration on the GPU as well.

GPU, also known as a Visual Processing Unit (VPU), is the name given to a type of microprocessor specialized in processing graphics; its parallel processing architecture makes it more capable of this type of work than a normal CPU. The CPU is responsible for the main operating system and GPU is a coprocessing unit. The CPU consists of an Arithmetic Logic Unit (ALU) used to temporarily store the data and perform calculations, and a Control Unit (CU) which performs instruction sequencing. Table 3.4 shows the main differences between the CPU and the GPU (Singh, 2019).

From Table 3.4 it is necessary to ascertain if the best approach for extracting features from an image is to use the capabilities of a GPU. Section 4.10 will focus on the creation

Table 3.4: CPU vs GPU Comparison (Singh, 2019).

CPU	GPU
CPU stands for Central Processing Unit	GPU stands for Graphics Processing Unit
CPU consumes or needs more memory than GPU	GPU consumes or requires less memory than CPU
CPU contain fewer powerful cores	GPU contain more weak cores
CPU is suitable for serial instruction processing	GPU is suitable for parallel instruction processing
CPU emphasis on low latency	GPU emphasis on high throughput

of a GLCM matrix using the GPU CUDA framework, and displaying the differences in processing time between a CPU and a GPU.

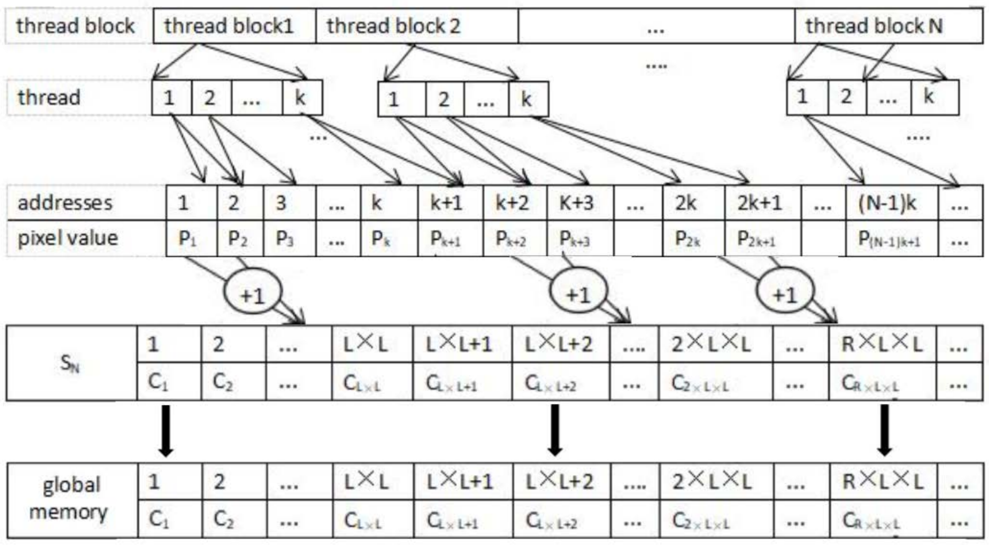


Figure 3.77: GLCM diagram computation using CUDA framework with $\theta = 0°$ and $d = 100$. Adapted from (Hong et al., 2018)

In the diagram in Figure 3.77, it is possible to identify two main steps: the first GLCM matrix phase creation and development; plut the addition of the GLCM transposed matrix. Considering this, along with programming using the CUDA framework, a two-dimensional kernel was created in which each block has 16 threads and the grid size contains $Width/16$ blocks in X and $Height/16$ blocks in Y. The algorithm has a total of $Width * Height$ threads running in parallel (in this dissertation, where $Width = 640$ and

Height = 480, the result is a parallel of 307 200 threads to process the image under study). However, the advantage of working in parallel mode raises other concerns that prove that programming using only the CPU is not necessary: "Multiple CUDA threads' reading and writing of the same position" (Hong et al., 2018). Section 4.10 will study in detail the best approach to design the GLCM matrix, the differences between CPU and GPU using the GLCM matrix, showing the results obtained and their conclusions.

3.5 Mapping

It is possible for an aerial vehicle to be autonomous when its route planning and decision making abilities are accurate enough to enable safe and self-controlled navigation. Furthermore, aerial mapping of the classified terrains using UAVs can be very useful, and can be implemented for many different type os application, such as autonomous navigation, precision agriculture, emergency landings and rescue missions.

The fusion between both of the techniques applied for terrain classification - static textures and dynamic textures; through MLP results in several images where each pixel represents one of two possible outputs:

- Water (pool and lake);
- Non-Water (vegetation, asphalt and sand).

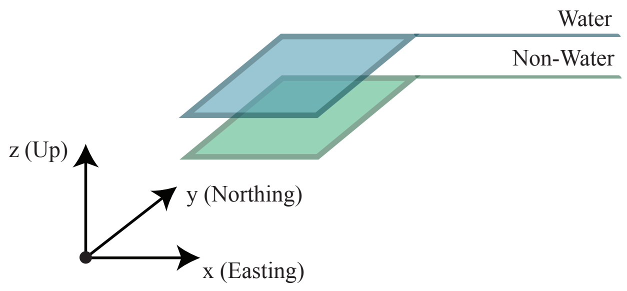


Figure 3.78: Layered mapping of the classified terrain types.

This system's mapping procedure is similar to the Aerial Semantic Mapping algorithm described for a precision agriculture experiment (Salvado, 2018). The framework's pipeline uses a dynamic ROS grid map composed of two layers (one for each type of terrain, Figure 3.78). These layers are converted into the *OccupancyGrid* ROS message type which saves a georeferenced map with all the collected and classified images. The layered grid map cells have a value of 0 (white) when the pixel presents the corresponding layer's terrain type, otherwise the cell value is 1 (black).

Georeferencing data requires information about GPS coordinates (latitude, longitude and altitude), high-precision positioning, IMU attitude (yaw), the camera lense's FOV, the image's resolution and aspect ratio (Figure 3.79).

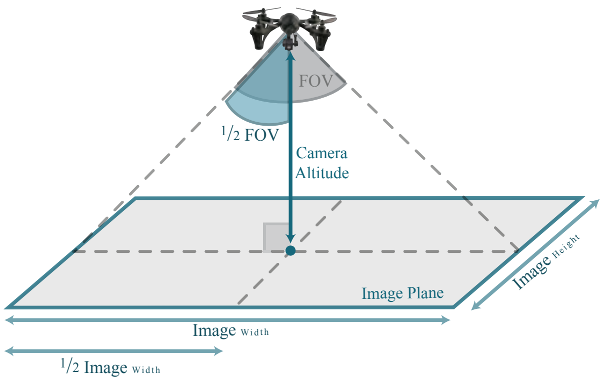


Figure 3.79: Schematic of how to determine the real image plane dimension.

$$
\begin{aligned}
Image_{width} &= 2 \times \tan(FOV/2) \times UAV_{altitude} \\
Image_{height} &= Image_{width} \times \frac{1}{AspectRatio}
\end{aligned}
\tag{3.46}
$$

Equation 3.46 computes the real dimensions of the captured area, where FOV is measured in degrees (^{o}) and the *AspectRatio* is the image's height:width proportion factor.

The Parrot Bebop2 (Base_link) uses a counter-clockwise IMU sensor, which is in line with ROS REP:103 conventions (Tully Foote, 2014) followed by the *World* and *Map* frames. On the other hand, OpenCV uses a clockwise system, which means that when the x-axis points to the right, then the y-axis is pointing down, Figure 3.80.

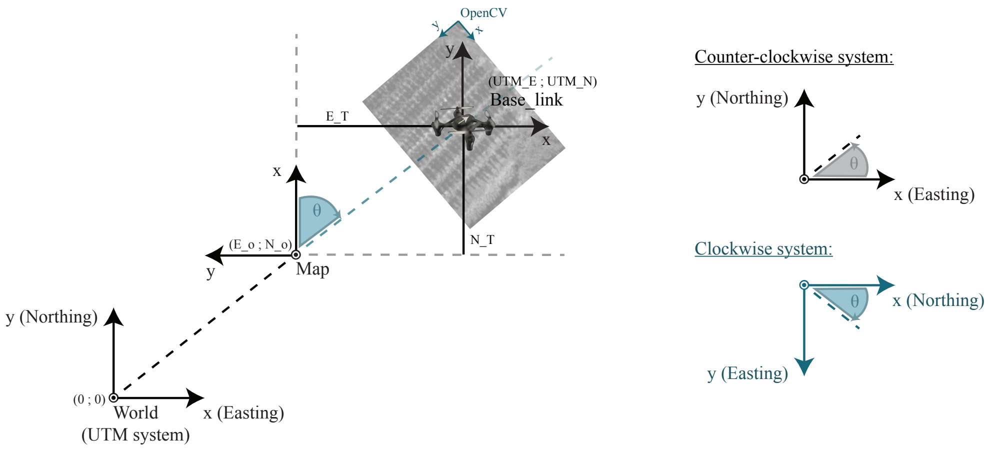


Figure 3.80: Coordinate systems of interfaces: Base_link, World and Map (black counter-clockwise coordinate systems), OpenCV (blue clockwise coordinate systems).

The precision implied by computer vision systems and ROS image mapping also requires caution when it comes to implementing the proper rotation and translation transformations. As illustrated in the schematic of Figure 3.80, the standard 90^o counter-clockwise rotation is considered, to deal with the world→map transformation in equation 3.47.

$$\begin{gathered}
StandardRotation: \\
M(\theta) = \begin{bmatrix} cos(\theta) & -sen(\theta) \\ sen(\theta) & cos(\theta) \end{bmatrix} \\[2em]
90^o Rotated: \\
M(\theta) = \begin{bmatrix} cos(\theta) & -sen(\theta) \\ sen(\theta) & cos(\theta) \end{bmatrix} \cdot \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix} \\[1em]
= \begin{bmatrix} -sen(\theta) & -cos(\theta) \\ cos(\theta) & -sen(\theta) \end{bmatrix}
\end{gathered} \tag{3.47}$$

The following equation determines pixels rotation motion according to a rotation matrix M by θ (equation 3.48).

$$\begin{bmatrix} px' \\ py' \end{bmatrix} = M(\theta) \cdot \begin{bmatrix} px \\ py \end{bmatrix} \tag{3.48}$$

Then, according to Figure 3.80, the first captured image center is positioned at (E_o, N_o). The subsequent images will be mapped in line with a pivot rotation, as explained in the following steps, where P(px,py) is the position of each image pixel:

1. P(px,py) point translation to the map origin (E_o, N_o);
2. P(px,py) point rotation around the map origin (E_o, N_o);
3. P(px,py) point back translation to the Base_link origin (UTM_E, UTM_N);

$$\begin{gathered}
P(px;py) = \begin{bmatrix} px \\ py \\ 1 \end{bmatrix} \qquad T(E_T;N_T) = \begin{bmatrix} 1 & 0 & E_T \\ 0 & 1 & N_T \\ 0 & 0 & 1 \end{bmatrix} \\[1em]
M(\theta) = \begin{bmatrix} -sen(\theta) & -cos(\theta) & 0 \\ cos(\theta) & -sen(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} \\[1em]
P'(px';py') = T(E_T;N_T) \cdot M(\theta) \cdot T(-E_T;-N_T) \cdot P(px;py)
\end{gathered} \tag{3.49}$$

Equation 3.49, as explained in (Salvado, 2018), describes the pivot rotation applied to mapping procedures, where:

- $P'(px';py')$ is the new $P(px;py)$ point pivot rotation result;
- $M(\theta)$ is the counter-clockwise 90^o rotation matrix (Equation 3.47);
- $T(E_T;N_T)$ is the Map-to-Base_link translation matrix, where E_T and N_T are, respectively, the easting and northing translation in Figure 3.80;
- θ is the counter-clockwise Base_link IMU rotation angle (rad).

Based on the previous pivot rotation formula, it was possible to successfully compute a dynamic mapping algorithm (Equation 3.50) that accurately builds an aerial view over the UAV's flown area.

$$\begin{bmatrix} px' \\ py' \end{bmatrix} = \begin{bmatrix} -(px - E_T) \cdot sen(\theta) - (py - N_T) \cdot cos(\theta) + E_T \\ (px - E_T) \cdot cos(\theta) - (py - N_T) \cdot sen(\theta) + N_T \end{bmatrix} \tag{3.50}$$

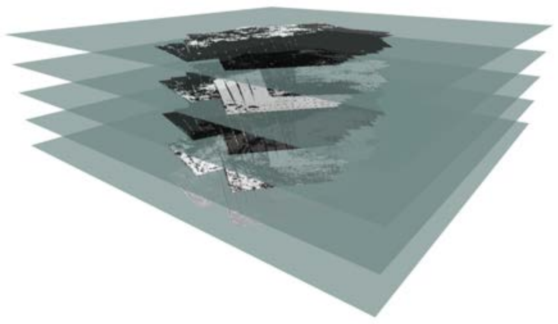


Figure 3.81: ROS map result of the four distinct classified terrain types in layers (1st layer- water; 2nd layer- vegetation; 3rd layer- asphalt; 4th layer- sand; 5th layer - RGB camera images). Georeferenced map visualization using the $RViz$ tool.

The classified terrain information obtained during this experiment was all georeferenced and accurately mapped using the ROS grid map layers, as illustrated in Figure 3.81, and can be visualized using the $RViz$ tool (3D visualization tool for ROS).

Chapter 4

Experimental Results

In order to validate the proposed static and dynamic texture features for terrain classification, in this section, four different terrain types (water, vegetation, asphalt and sand) were analyzed. As mentioned in Section 3.1.2, these terrain types were studied at four locations in Portugal as shown in Figure 4.1.

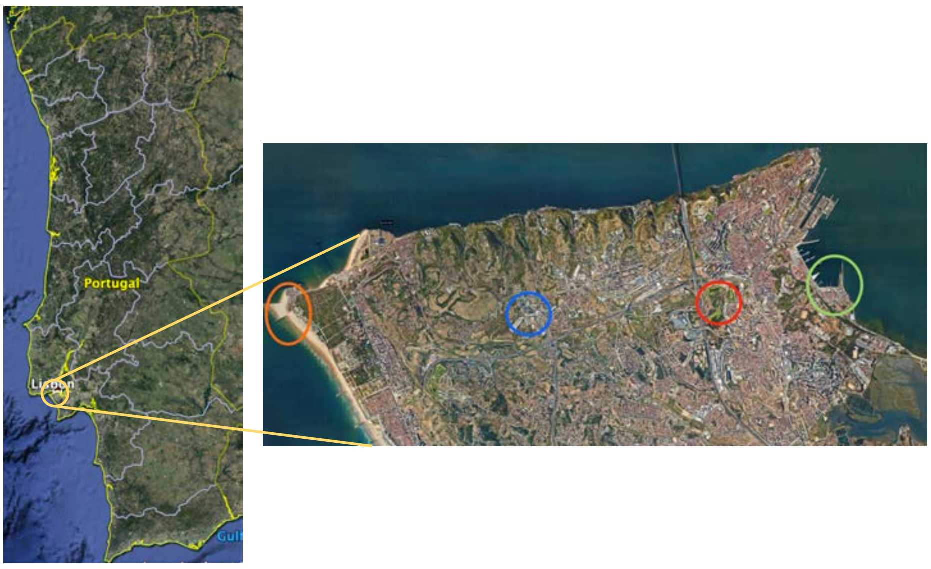


Figure 4.1: Data collection locations in Portugal.

The sample images were acquired in different conditions at these locations: at an altitude of one to two meters and from 8am to 6pm. Regarding environmental conditions, the author collected data during both clear and cloudy skies. To avoid overfitting and in order to make the proposed algorithms more robust, older data (with and without the downwash effect) from other locations (e.g. agriculture field at Spain) and from other UAVs were also used. Furthermore, to improve generalization, a technique called "Early Topping" was used, with two different data sets: the training set, to update the weights and biases, and the validation set, to stop training when the network begins to overfit the data.

For validation, a "k-fold" technique was used, where the data was divided into k randomly chosen subsets of roughly equal size (k=5 was selected for this dissertation).

The complete data set collected is composed of 11121 images, distributed as follows: 2332 water, 2373 vegetation, 834 asphalt and 5582 sand images.

Thus, in the current section each of the proposed texture features (static or dynamic), will be evaluated and validated individually and all possible combinations between all features will also be taken into account, to identify which will have the best behavior for terrain classification.

4.1 Gabor with Lowess Regression

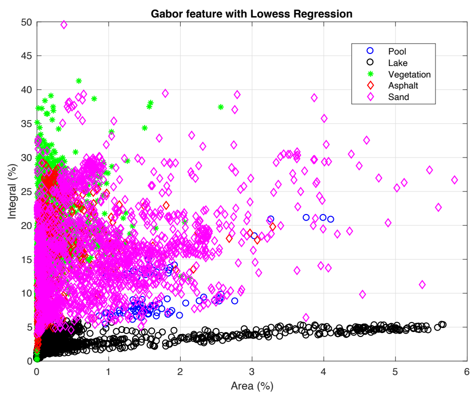
As mentioned in Section 3.2.1, regarding Gabor and Lowess Regression combination, two important features were calculated:

1. Area measured between the local minimum and its respective two local maxima;
2. Integral between the local minimum and its two respective local maxima.

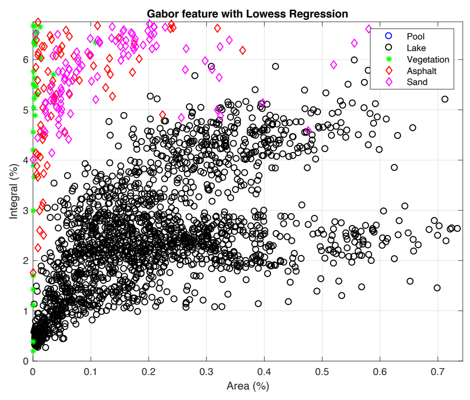
These two features were used as input to an NN, designed the same way as for the other algorithms proposed in this dissertation. Thus, it becomes possible to correctly compare the accuracy of each algorithm. The Neural Network model consists of the hidden layer containing 10 neurons, and the third layer corresponds to the system output with the number of possible terrain outputs under study in this dissertation. Each neuron uses a sigmoidal function to calculate its output. They are connected as a Fully Connected Feed Forward Neural Network. During the training stage, 70% of the total data was used for training, 15% for testing and 15% for validation;

After the initial summary, Figures 4.2 a), b), c) and d) show the results obtained by the Gabor and Lowess regression. For an easier understanding of Figures 4.2 a) and b), Figures 4.2 c) and d) have been added so that it is possible to clearly identify water-type and non-water terrains where Figures 4.2 b) and d) correspond to Figures 4.2 a) and c), respectively, zoomed in.

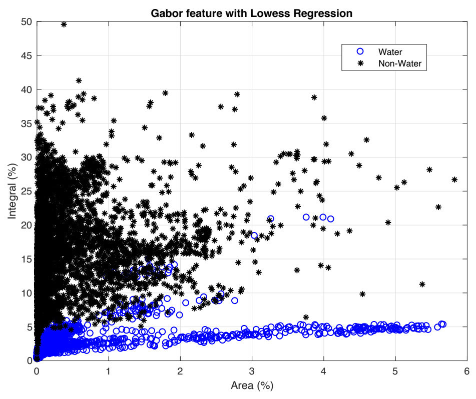
Despite the greater overlap in non-water type terrains (in this case vegetation, sand and asphalt), it is possible to see a separation between water and non-water type terrains.



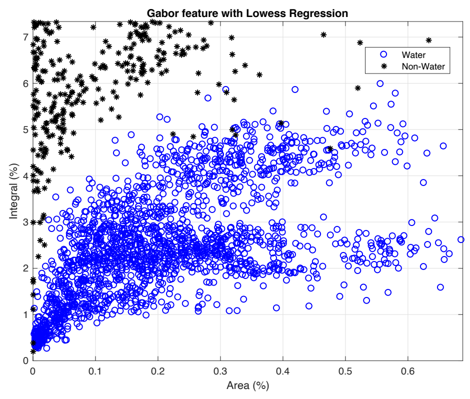
a



b



c



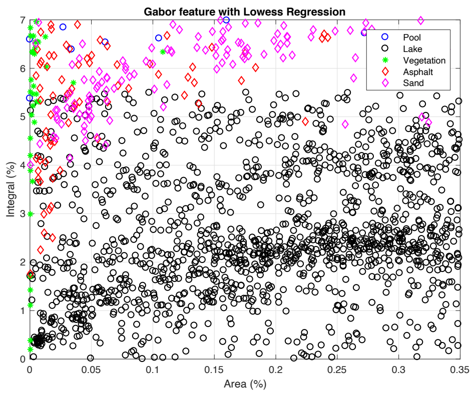
d

Figure 4.2: Gabor Static texture feature with Lowess Regression: a) Area relationship with respect to the integral of minima and maxima locals for four terrain types; b) and d) are a zoom of a) and c), respectively.

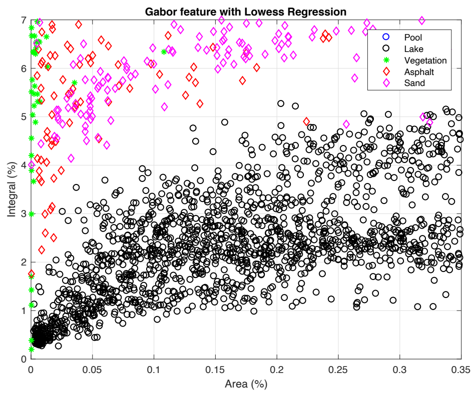
Regarding the integral percentage values present in the water-type terrain, low values are presented due to the fact that the only textures shown in the image under study come from the downwash effect.

In addition to the terrain classification between water and non-water type terrains, it is important to know the influence that the downwash effect has on these same terrains. New data was taken in the same locations without the downwash effect. Then, the same algorithm was applied, producing the results that can be seen in Figures 4.3 a), b), c) and d).

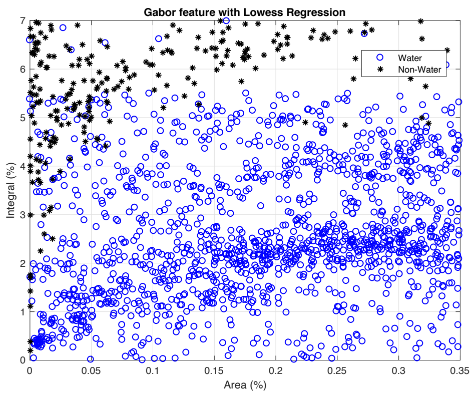
Figures 4.3 a) and b) illustrate that, without the downwash effect caused by the UAV, water-type terrains behave in a more random way, in which their shape depends only on the luminosity and natural wind (not provoked by the UAV). Thus, it was also concluded that the effect of the downwash caused by the UAV, makes the behavior of water-type terrains more constant and, as can be seen in Figures 4.3 b) and d); it makes the results more compact and easier to correctly classify the terrain under study.



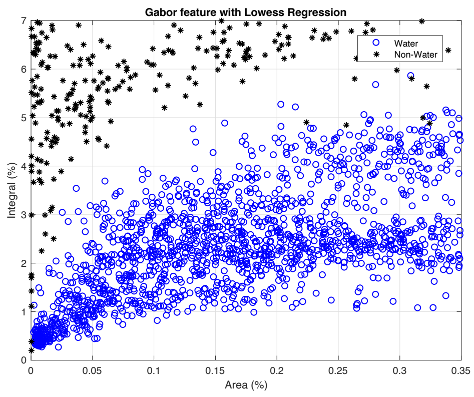
a



b



c



d

Figure 4.3: Gabor and Lowess Regression algorithms in relation to the downwash effect: a) and c) without downwash effect; b) and d) with downwash effect.

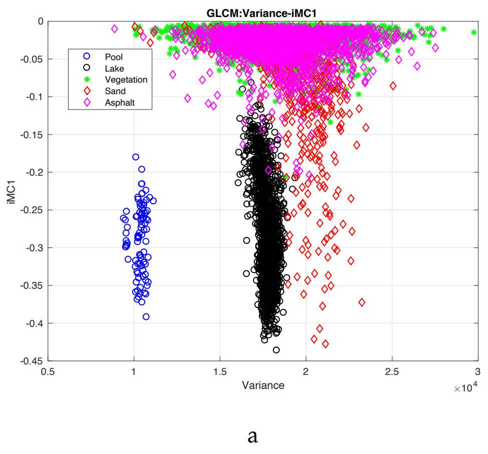
4.2 Particle Swarm Optimization

To test the combination of PSO and GLCM, the 27 features studied in Section 3.2.5 were calculated and computed, and applied to each dataset image(see Table 3.3). The results of the whole set of frames under study were analysed and 12 of 27 combinations of features were selected by the author. The selected combination of features were: 1- Variance with IMC1; 2- Correlation with Joint Average; 3- Variance with Difference Entropy; 4- Homogeneity with Variance; 5- Entropy with Variance; 6- Variance with IMC2; 7- Sum Average with Difference Entropy; 8- Sum Average with IMC1; 9- Sum Average with IMC2; 10- Auto-Correlation with IMC1; 11- Auto-Correlation with IMC2 and 12- Joint Average with IMC2.

As can be seen in the selected feature enumeration, it is possible to conclude that the impact of the features was:

- Variance: 41.67% of impact;
- IMC2: 33.33% of impact;
- IMC1: 25.00% of impact;
- Sum Average: 25.00% of impact;
- Joint Average: 16.67% of impact;
- Difference Entropy: 16.67% of impact;
- Auto-Correlation: 16.67% of impact;
- Correlation: 8.33% of impact;
- Entropy: 8.33% of impact;
- Homogeneity: 8.33% of impact;

Since the Variance feature was used in more than 41% of cases, Figure 4.4, Figure 4.5, Figure 4.6, Figure 4.7 and Figure 4.8 show the results of comparing Variance to IMC1, IMC2, Entropy, Difference Entropy, Correlation and Homogeneity, for the whole dataset.



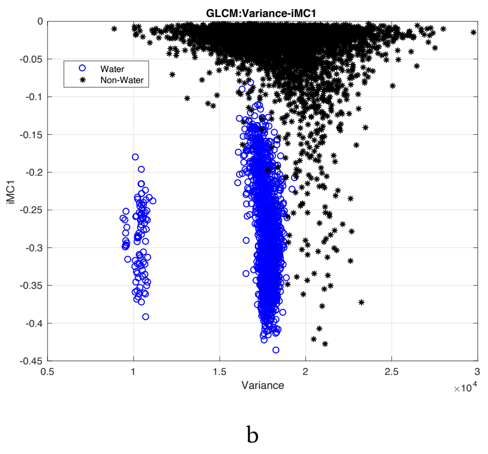


Figure 4.4: PSO results for IMC1 with respect to Variance. a) five terrain types; b) water vs non-water.

In Figure a), water terrain type is represented by black (pool) and blue (lake), green represents the vegetation-type terrain, sand-type terrain is represented by pink and therefore, red represents the asphalt-type terrain, while in Figure b) the water-type terrain is represented by blue (pool and lake) and black represents the non-water type terrain. It is possible to conclude that the combination of PSO and GLCM cannot easily differentiate water-type terrains from non-water type terrains.

Even using the downwash effect caused by the UAV, a higher variation in the output results is observed, as shown in Figure 4.9. This problem arises from the fact that the PSO removes the high frequencies in the images, which to an extent causes the removal of the downwash effect.

Figures 4.10 shows the comparison between using the PSO and GLCM combination and using the GLCM algorithm only. It can be concluded that using GLCM only, instead of the combination with PSO, makes the system much more stable. For this reason, the author concluded that it is not feasible to proceed with the PSO algorithm.

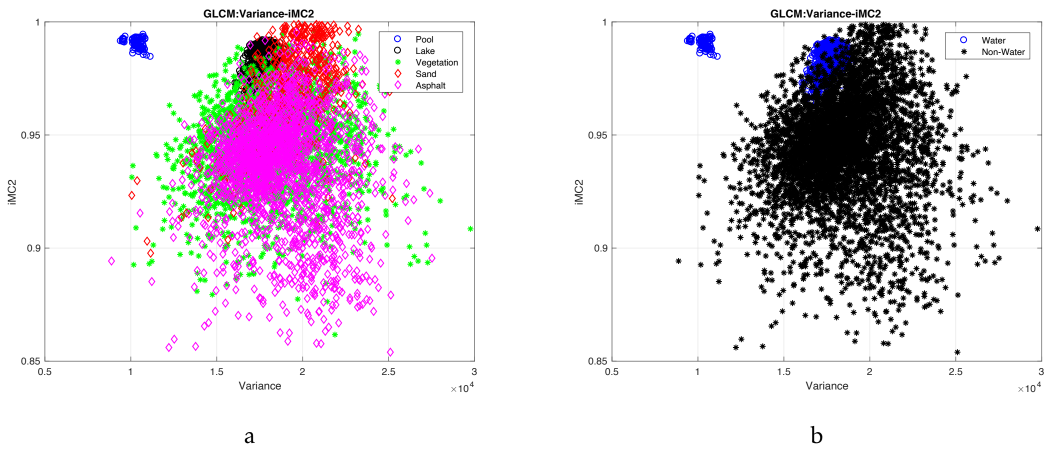


Figure 4.5: PSO results for IMC1 with respect to Variance. a) five terrain types; b) water vs non-water.

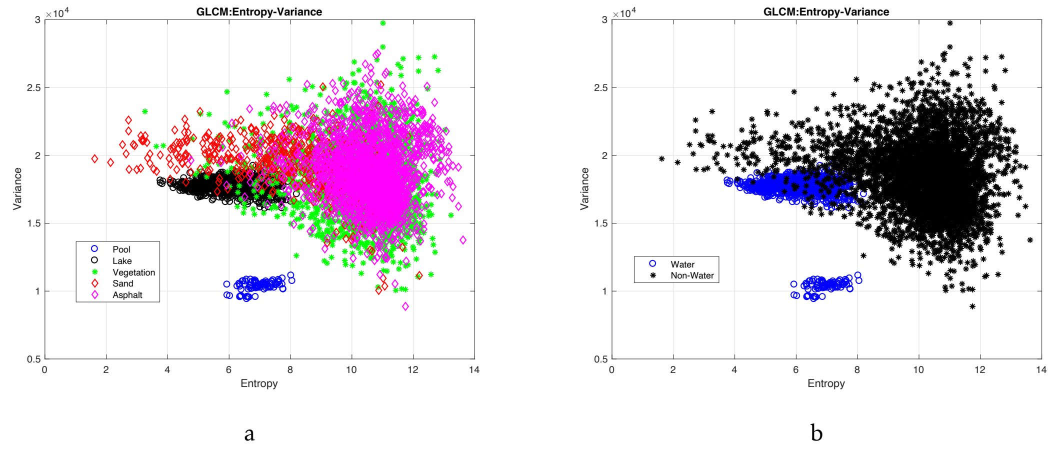


Figure 4.6: PSO results for Variance with respect to Entropy. a) five terrain types; b) water vs non-water.

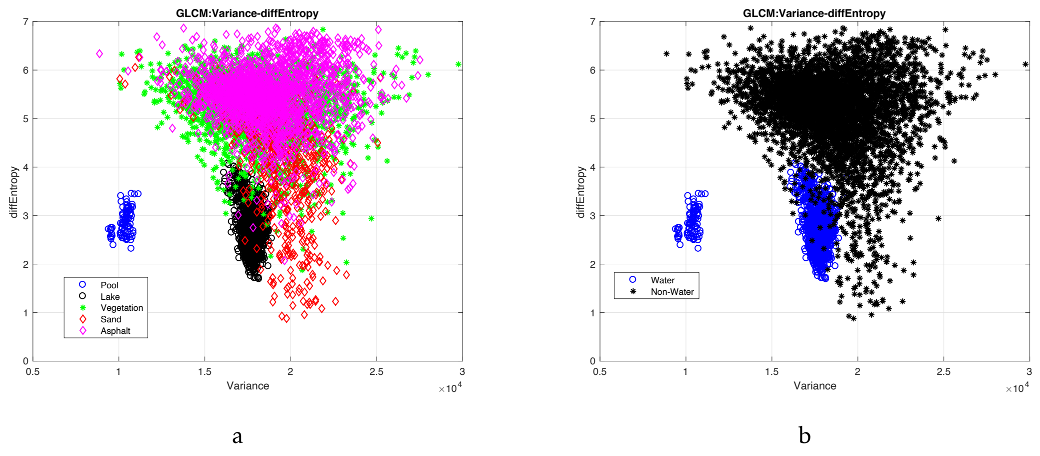


Figure 4.7: PSO results for Difference Entropy with respect to Variance. a) five terrain types; b) water vs non-water.

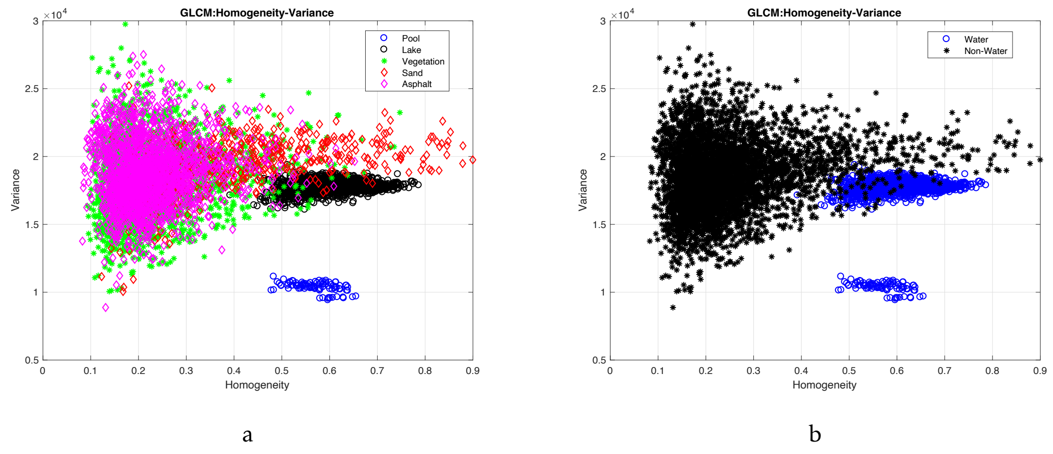


Figure 4.8: PSO results for Variance with respect to Homogeneity. a) five terrain types; b) water vs non-water.

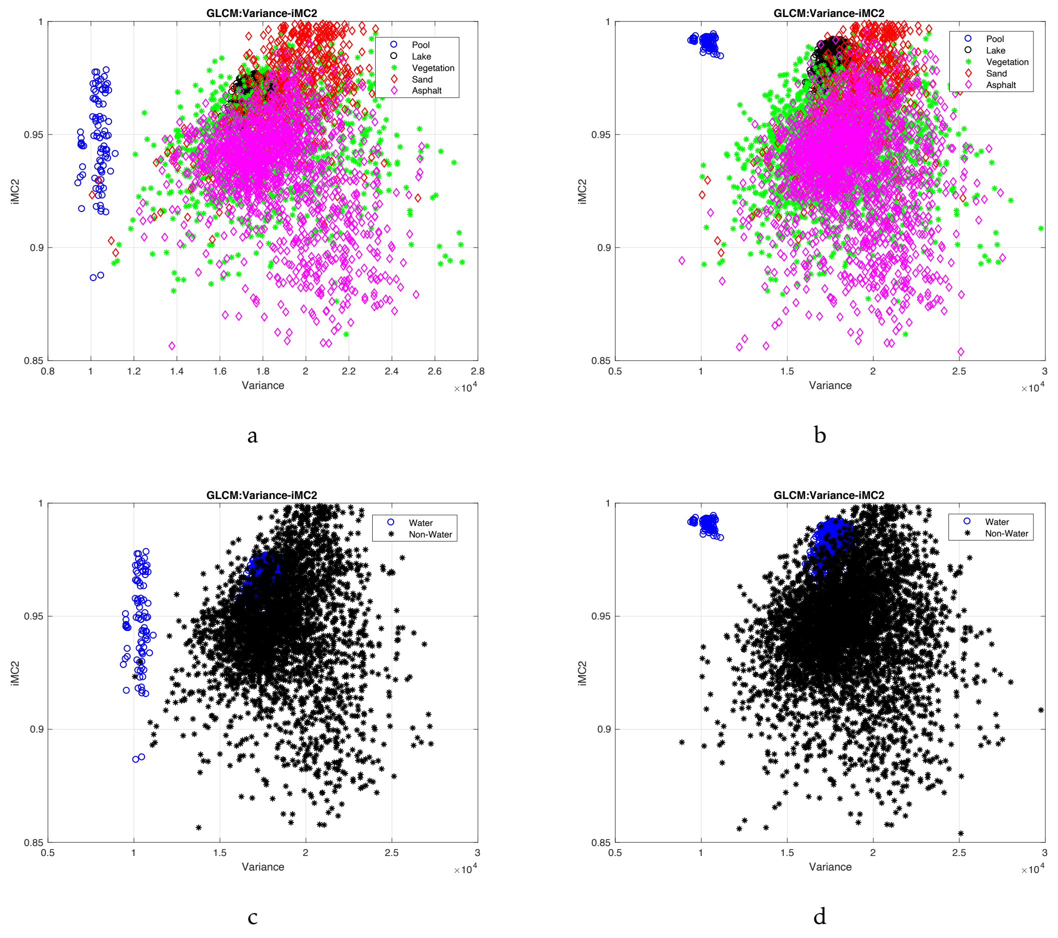


Figure 4.9: PSO and GLCM algorithms: a) and c) without downwash effect; b) and d) with downwash effect.

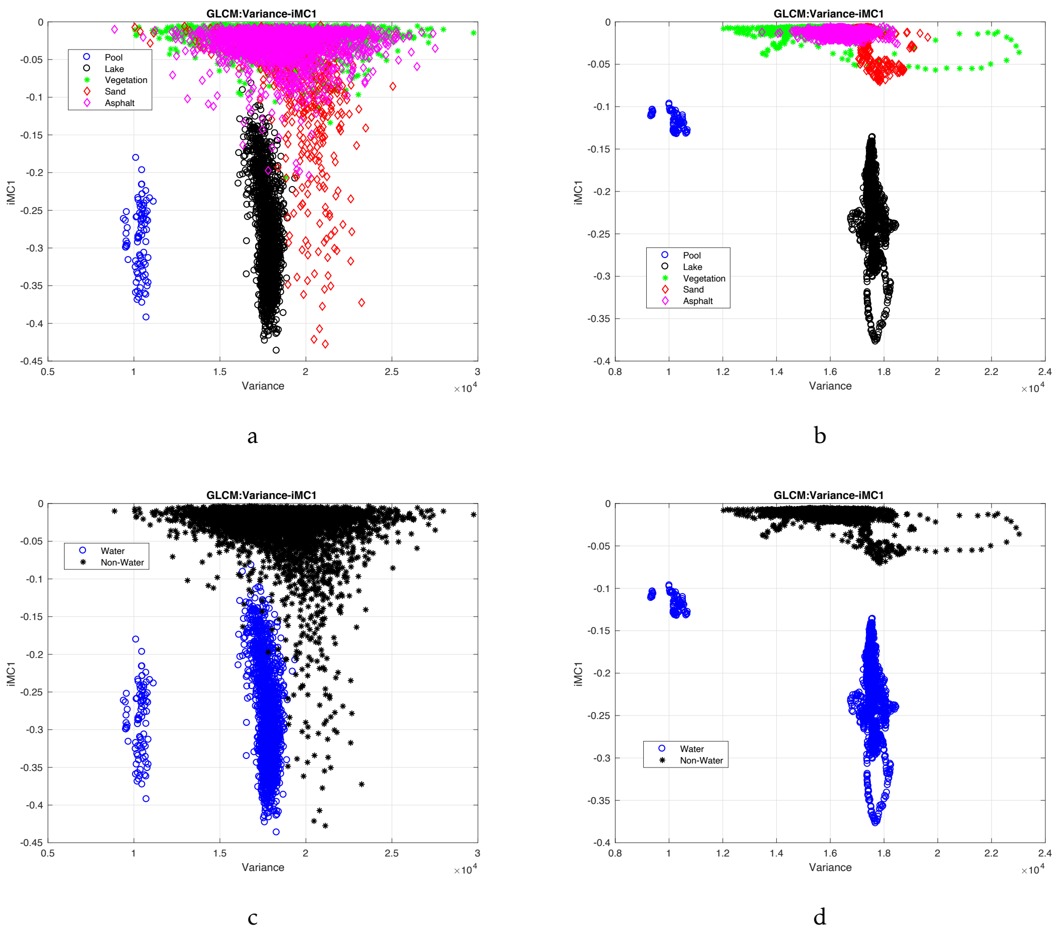


Figure 4.10: Comparison between: a) and c) PSO and GLCM combination and; b) and d) only using the GLCM algorithm.

4.3 Empirical Mode Decomposition

After obtaining the IMFs from the Empirical Mode Decomposition, which is applied in the spatial domain, two features are extracted and used by an NN to classify the terrain under study as water or non-water type terrain. The following features take into consideration each terrain's IMF:

1. Number of local maxima given by IMF output;
2. Number of times the IMF result has crossed zero along the vector.

Figure 4.11 and Figure 4.12 show the results of the first three IMFs (IMF-A, IMF-B and IMF-C) with the dataset of this dissertation (regarding these two features). Two important aspects can be concluded:

1. It is possible to separate water-type terrain from non-water-type terrain with good precision via IMF-A where, in Figure 4.11 a) and b), it goes up to 47% and 24%

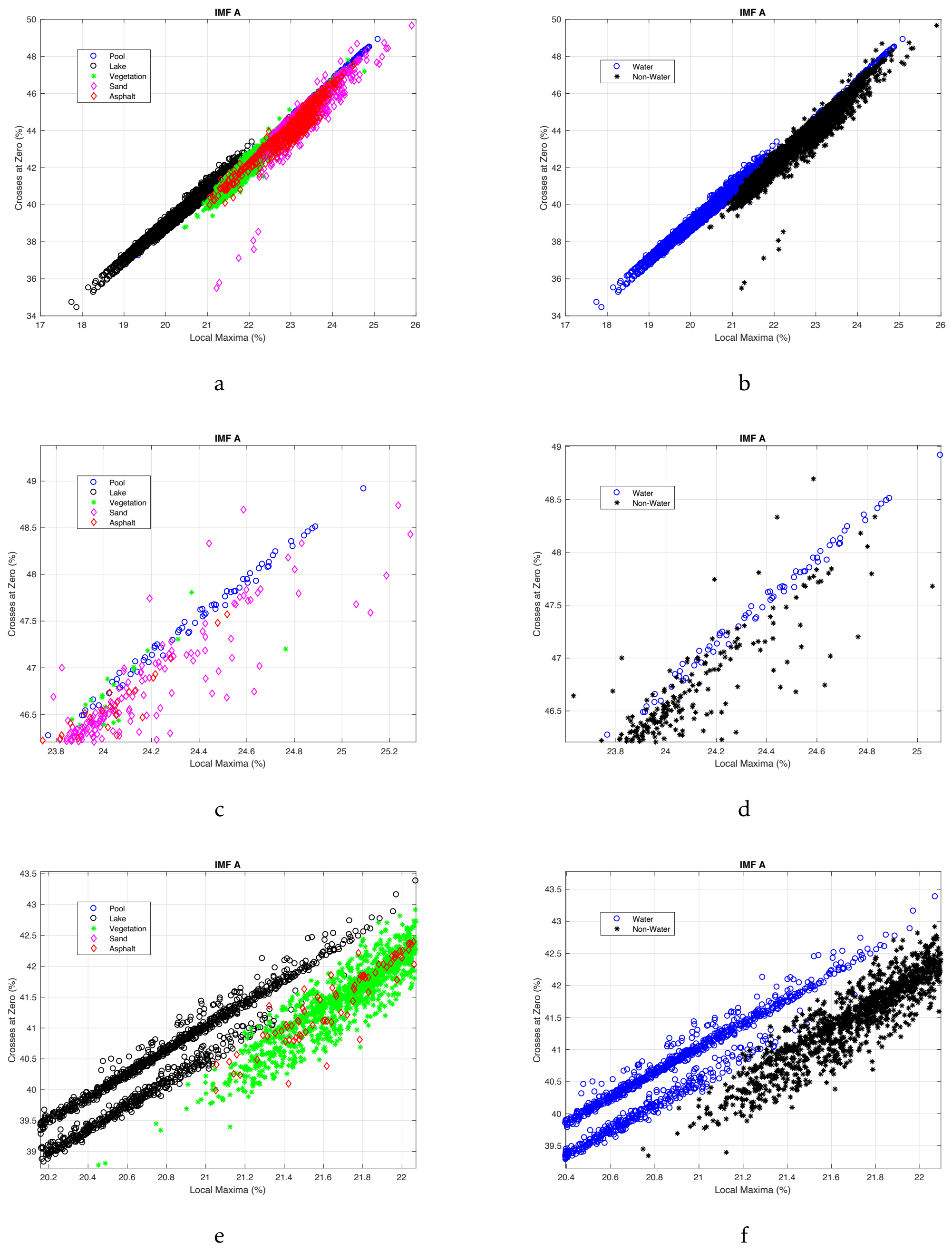
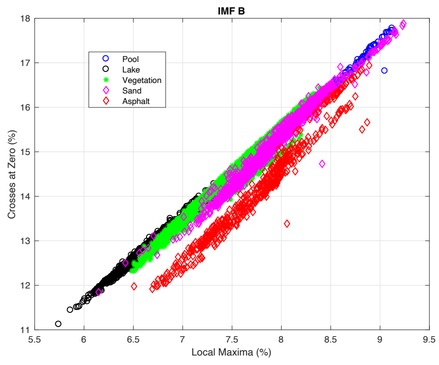
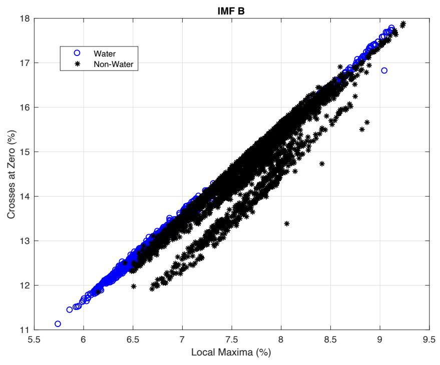


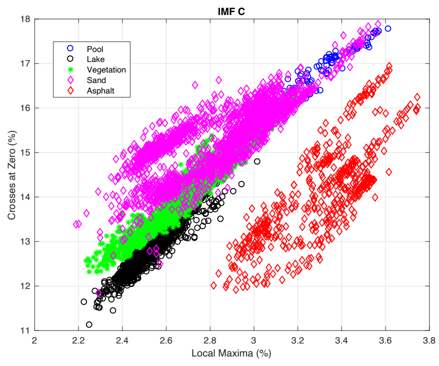
Figure 4.11: Intrinsic Mode Function – The zero crossings in relation to the total number of local maxima. a) and b) IMF-A result; c) and d) Zoom of pool result; e) and f) Zoom of lake result.



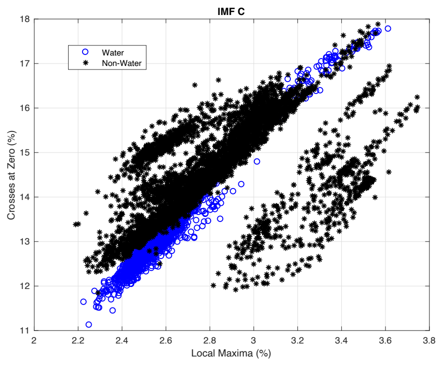
a



b



c



d

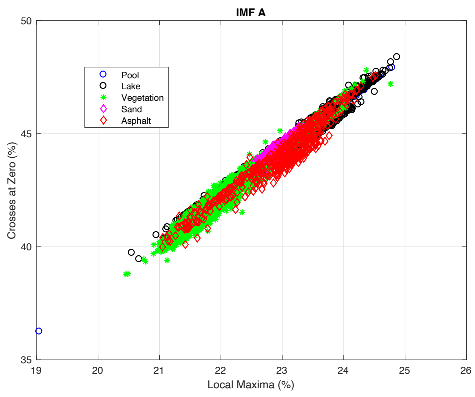
Figure 4.12: Intrinsic Mode Function – The zero crossings in relation to the total number of local maxima. a) and b) IMF-B; c) and d) IMF-C.

at crossing at zero and local maximum, respectively; in general the pool (water) terrain type is represented. Between 40% - 46% and 21% - 24% at crossing at zero and local maximum respectively, represents the non-water terrain type. It is also observed, in Figure 4.11 a) and b), that lake (water) terrain type presents crossing at zero values from 38% to 42% and local maximum values from 20% to 21.4%;

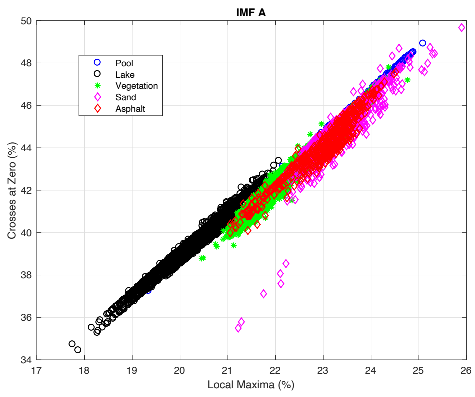
2. It is important to mention that as the IMF's iteration increases, the detection of the different clusters becomes less clear, and therefore, the error of classifying the terrain type increases. For this reason, in this dissertation, only IMF-A was used to extract terrain features to send to the NN.

As in the case of the two previous sections (Section 4.1 and Section 4.2), in addition to the terrain classification into water-type terrain and non-water type terrain, it is important to know the influence that the downwash effect has on these same terrains. New data was taken in the same locations without the downwash effect. Then, the EMD algorithm was applied and the results can be seen in Figures 4.13 a), b), c) and d).

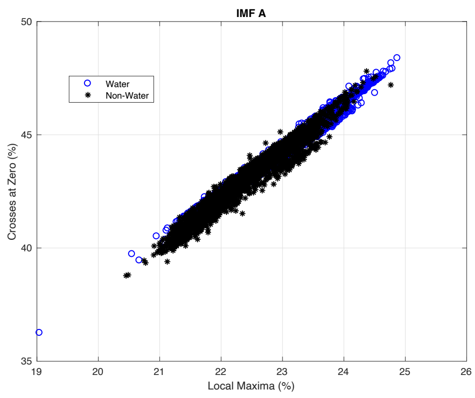
Figures 4.13 a) and c) illustrate how, without the downwash effect caused by the UAV, water-type terrain (pool and lake) have local maximum and zero crossings values



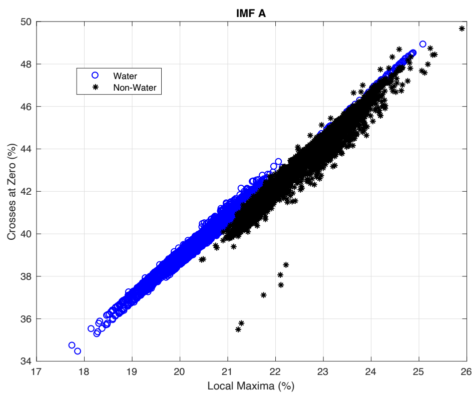
a



b



c



d

Figure 4.13: EMD algorithm: a) and c) without downwash effect; b) and d) with downwash effect.

close to non-water type terrain, which will cause overlap between these two types of terrains. Consequently, it will be very difficult to differentiate between water and non-water terrains. However, it is also possible to conclude through Figures 4.13 b) and d) that with the downwash effect, this overlap, between different types of terrain, decreases and therefore their classification is improved.

4.4 Wiener-Khinchin

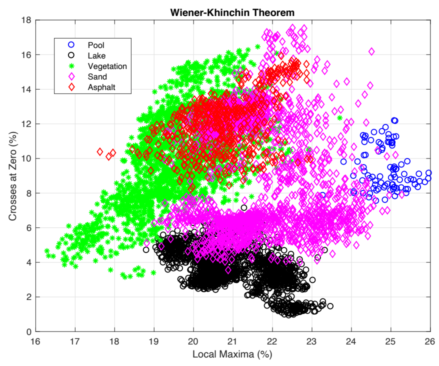
The Wiener-Kinchin algorithm, as mentioned in Section 3.2.4, works in the frequency domain. As in Section 4.3, this section also proposes three features in order to send these three features to the NN and classify the terrain under study as water or non-water type. The following features take into consideration each terrain's W-K Filter:

1. Number of local maxima given by W-K Filter output;

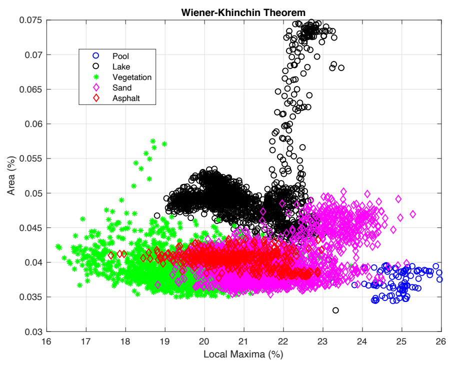
2. Number of times the W-K Filter result has crossed zero along the vector;

3. Area ratio of the external area to the internal area.

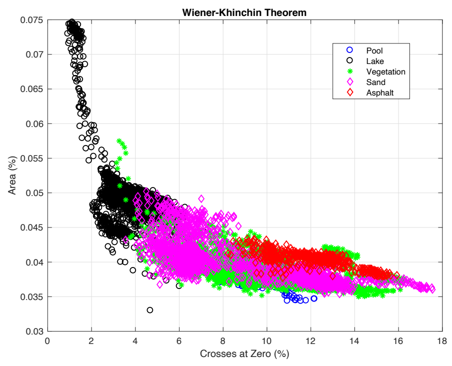
The following figures (Figures 4.14 a) b) and c)) show the possible combinations between these three features, with the dataset of this dissertation, in order to identify which will be used in the terrain classification.



a



b



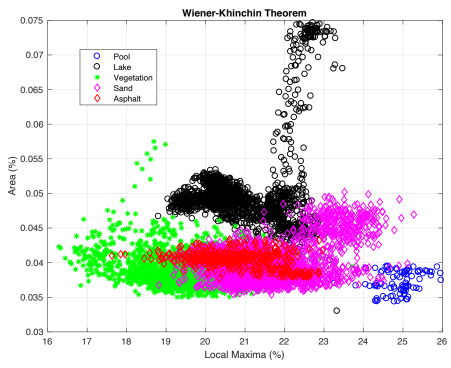
c

Figure 4.14: Wiener Khinchin results with three features – a) The zero crossings in relation to the total number of local maxima; b) The area in relation to the total number of local maximum; c) The area in relation to the zero crossings.

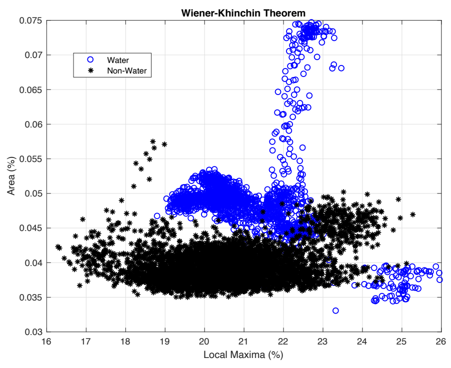
From Figures 4.14 it is possible to visually conclude that the area in relation to the total number of local maxima can more accurately classify water-type and non-water-type terrains (mathematical values that validate this conclusion, defined in Section 4.9). Thus, with regard to terrain classification of the three features, only the number of local maxima and the area will be used as input for the MLP. Thus, Figure 4.15 shows more detailed information regarding these two features.

As can be seen in Figures 4.15, with the W-K Filter, it is possible to identify and separate the water terrains (pool and lake) from non-water terrains where:

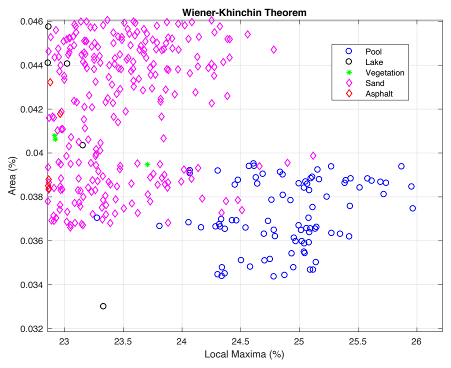
- **Pool (water):** Up to 24.5% at local maximum and area between 0.035% and 0.04%, the pool terrain type is represented.



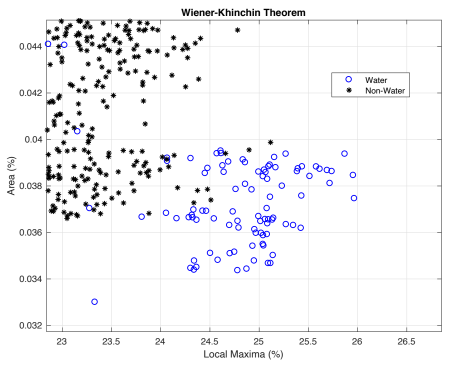
a



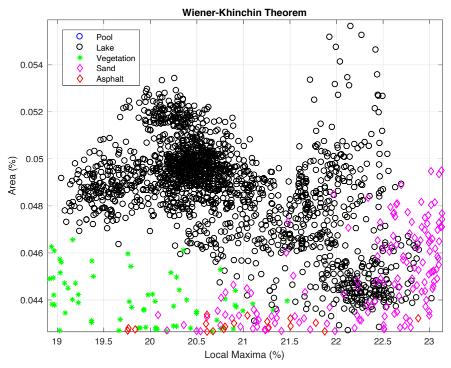
b



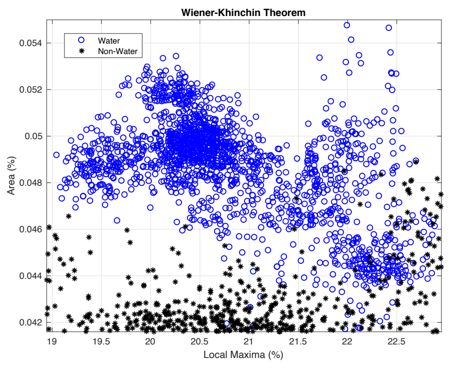
c



d



e

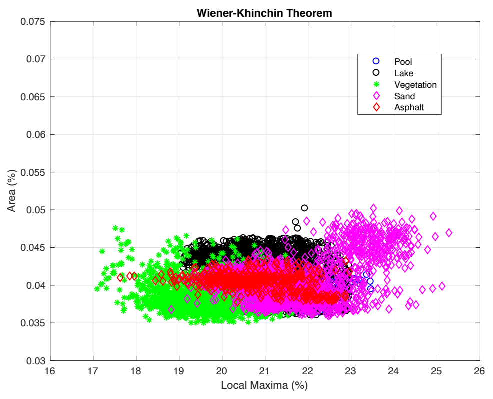


f

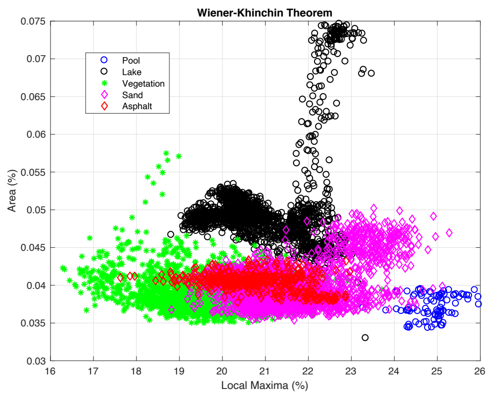
Figure 4.15: Wiener Khinchin results – Area in relation to the total number of local maxima. a) 5 terrain types; b) water vs non-water; c) and d) Zoom of pool result; e) and f) Zoom of lake result.

- **Lake (water):** Between 19% and 22.5% at local maximum and area between 0.045% and 0.075% represents the lake type terrain.

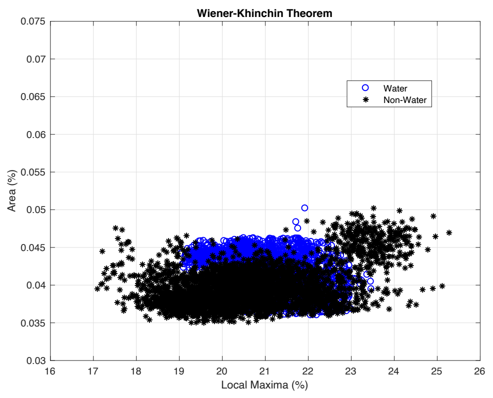
Just as in the three previous sections (Section 4.1, Section 4.2 and Section 4.3), in addition to the terrain classification between water-type terrain and non-water type terrain, the influence of the downwash effect by UAV rotors was also studied. New data were taken in the same locations without the effect of downwash. Then, the same algorithm in this section was applied and the results can be seen in Figure 4.16.



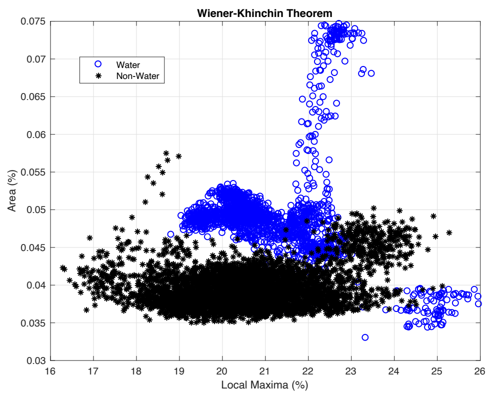
a



b



c



d

Figure 4.16: W-K Filter algorithm: a) and c) without downwash effect; b) and d) with downwash effect.

Figures 4.16 a) and c) illustrate that, without the downwash effect caused by the UAV, water-type terrain (pool and lake) will have local maximum and area values close to non-water type terrain, which will cause overlap between these two terrain types. It is also possible to conclude from Figures 4.16 b) and d) that, using the downwash effect, the overlap between water and non-water terrain types decreases due to the fact that the area values undergo a major change when affected by the downwash effect. Therefore, the probability of classifying the terrain increases. Section 4.9 will show the percentage of improvement using the downwash effect provoked by UAV rotors in water-type and non-water-type terrain classification.

Once again, it is important to note that the purpose of this dissertation is only to classify water-type terrain and non-water type terrain (while studying the influence of the downwash effect in terrain classification). The overlap between vegetation, sand and asphalt terrains does not present any problem for this thesis (as long as the overlap is not with water-type terrains - in this case, pool and lake terrains).

4.5 GLCM

This section will study the GLCM algorithm behaviour using the validation dataset. The 27 features presented in Table 3.3 were applied after each terrain GLCM matrix. The results of the whole set of frames were analysed and 12 of the 27 features were selected by the author. The selected combination of features were:

1- Entropy with IMC1; 2- Sum Average with Cluster Tendency; 3- Cluster Tendency with Auto-Correlation; 4- Entropy with Difference Average; 5- Entropy with Sum Average; 6- Homogeneity with Cluster Shade; 7- Homogeneity with Entropy; 8- Difference Average with Cluster Shade; 9- Difference Entropy with Cluster Shade; 10- Contrast with Entropy; 11- Correlation with Entropy and 12- Entropy with Sum Variance.

As the enumeration of combination of the features by the author shows, it is possible to calculate the impact of these features:

- Entropy: 58.33% of impact;
- Cluster Shade: 25% of impact;
- Difference Average: 16.67% of impact;
- Cluster Tendency: 16.67% of impact;
- Homogeneity: 16.67% of impact;
- Sum Average: 16.67% of impact;
- IMC1: 8.33% of impact;
- Auto-Correlation: 8.33% of impact;
- Contrast: 8.33% of impact;
- Correlation: 8.33% of impact;
- Difference Entropy: 8.33% of impact;
- Sum Variance: 8.33% of impact.

Since the Entropy feature has been used in more than 55% of cases, the Figures that depend on it will be shown. The following Figures (Figure 4.17, Figure 4.18, Figure 4.19, Figure 4.20, Figure 4.21, Figure 4.22 and Figure 4.23) show the results of the whole set of frames regarding Entropy with IMC1, Contrast, Correlation, Difference Average, Sum Average, Sum Variance and Homogeneity. In these figures: water-type terrain is represented by black (pool) and blue (lake), green represents vegetation-type terrain, sand-type terrain is represented by pink and red represents the asphalt-type terrain.

It is possible to conclude from the Figures that the GLCM algorithm can differentiate water-type terrain from non-water-type terrain (this is the focus of this dissertation).

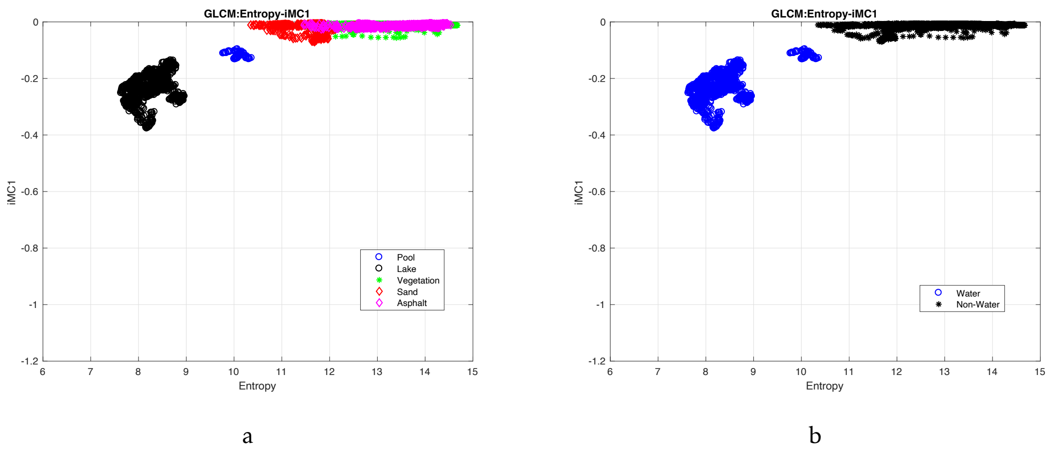


Figure 4.17: Entropy with respect to IMC1: a) five terrain-type classes and b) water vs non-water.

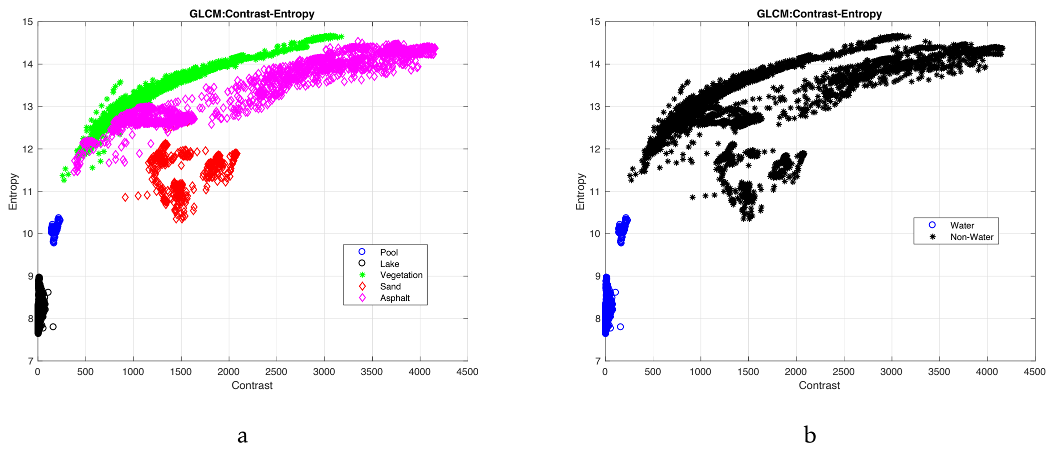


Figure 4.18: Contrast with respect to Entropy: a) five terrain-type classes and b) water vs non-water.

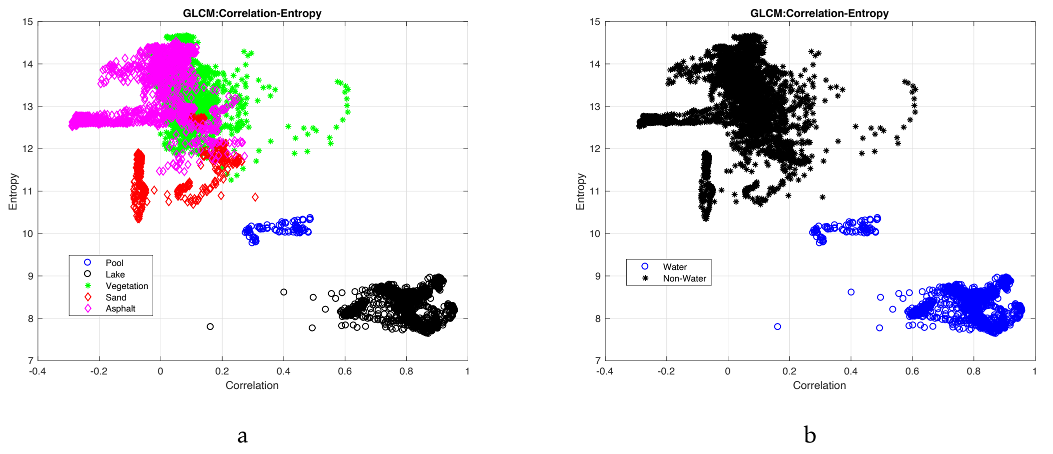
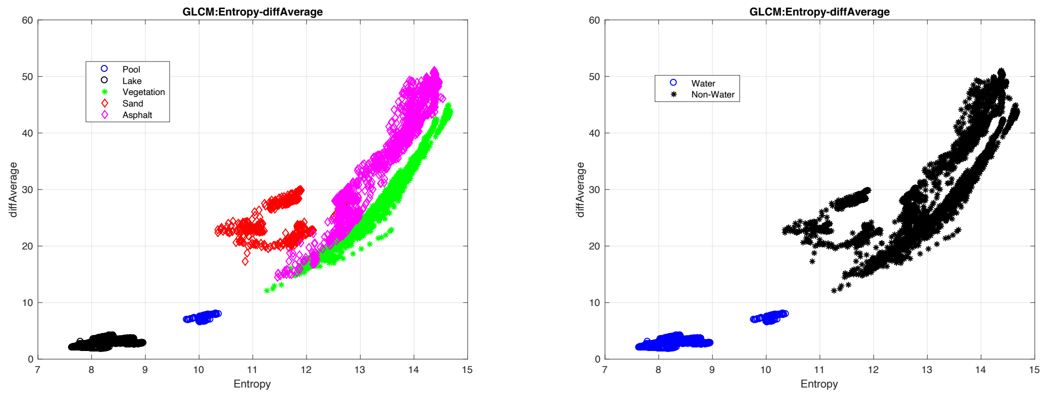
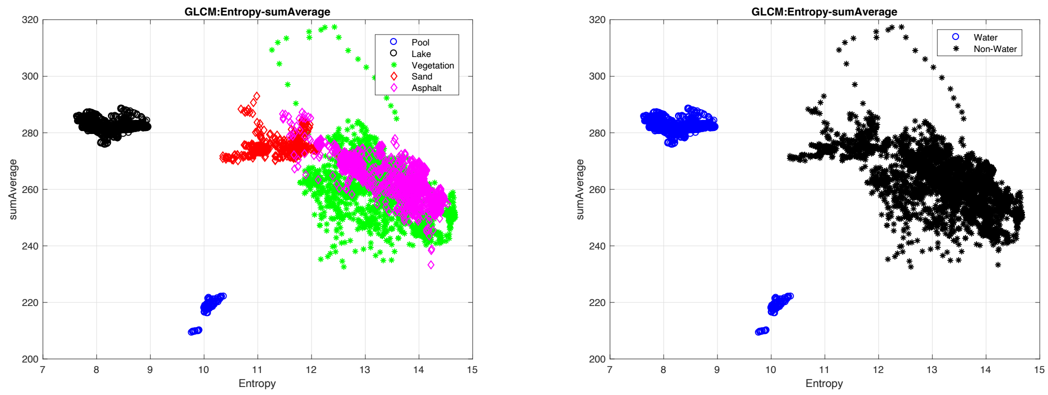


Figure 4.19: Correlation with respect to Entropy: a) five terrain-type classes and b) water vs non-water.



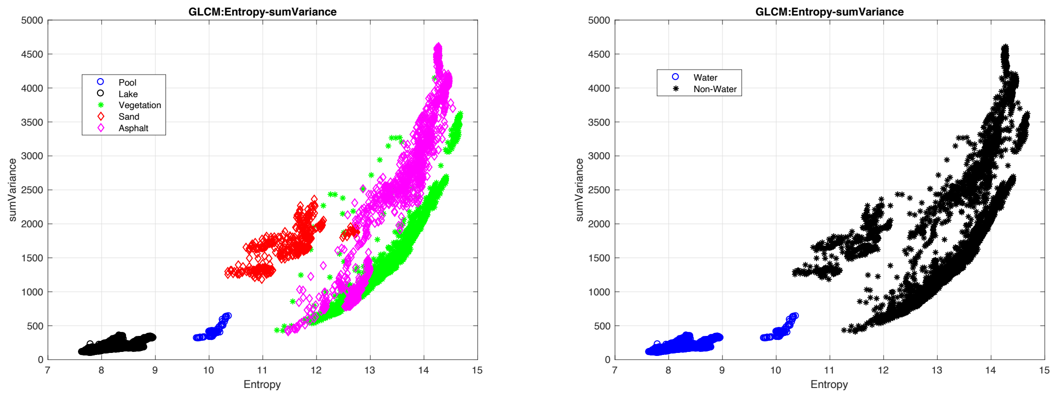
a b

Figure 4.20: Entropy with respect to Difference Average: a) five terrain-type classes and b) water vs non-water.



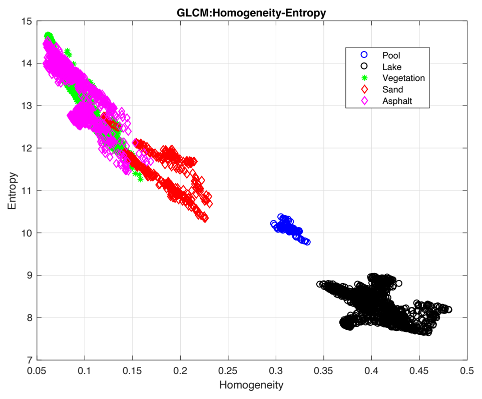
a b

Figure 4.21: Entropy with respect to Sum Average: a) five terrain-type classes and b) water vs non-water.

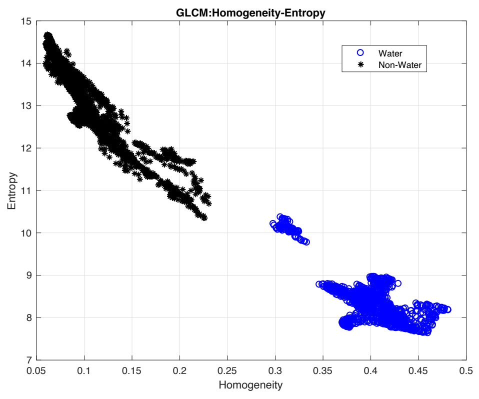


a b

Figure 4.22: Entropy with respect to Sum Variance: a) five terrain-type classes and b) water vs non-water.



a



b

Figure 4.23: Homogeneity with respect to Entropy: a) five terrain-type classes and b) water vs non-water.

However, the instability is higher in non-water-type terrains, due to the greater dispersion of the points in each Figure. In order to merge the data, while increasing certainty and automating the classification of terrain type, a machine-learning technique was used, named Feed-Forward Neural Network with MLP architecture.

As the focus of this dissertation is also to study the impact of the downwash effect in water-type terrain (pool, lake) and non-water type terrain (vegetation, asphalt and sand), data was taken in the same locations with and without the effect of downwash. Then, the GLCM algorithm was applied and the results are represented in Figure 4.24.

Figures 4.24 a) and c) show it is possible to observe that, without the downwash effect caused by the UAV, water-type terrain (pool and lake) will have entropy and IMC1 values close to non-water type terrain, which could cause overlap between these two terrain types. It is also possible to conclude from Figures 4.24 b) and d) that, using the downwash effect, the probability of overlap between water and non-water-type terrains decreases due to the fact that the entropy and IMC1 values of water-type terrain are more distant from the values of non-water-type terrain. Another advantage of using the downwash effect (as shown in Figures 4.24 b) and d)), is that water-type terrain clusters are more compact. Section 4.9 will present the percentage of improvement using the downwash effect provoked by UAV rotors in water-type and non-water-type terrain classification.

4.6 GLRLM

In GLCM the size of the GLCM matrix is already known (256x256 due to the pixel values: 2^8-1). On the other hand, the GLRLM matrix number of columns is not constant, despite the number of lines (2^8-1). For this reason, tests were carried out to choose the best value for the columns:

- **Minimum possible columns:** The highest pixel number length was calculated along

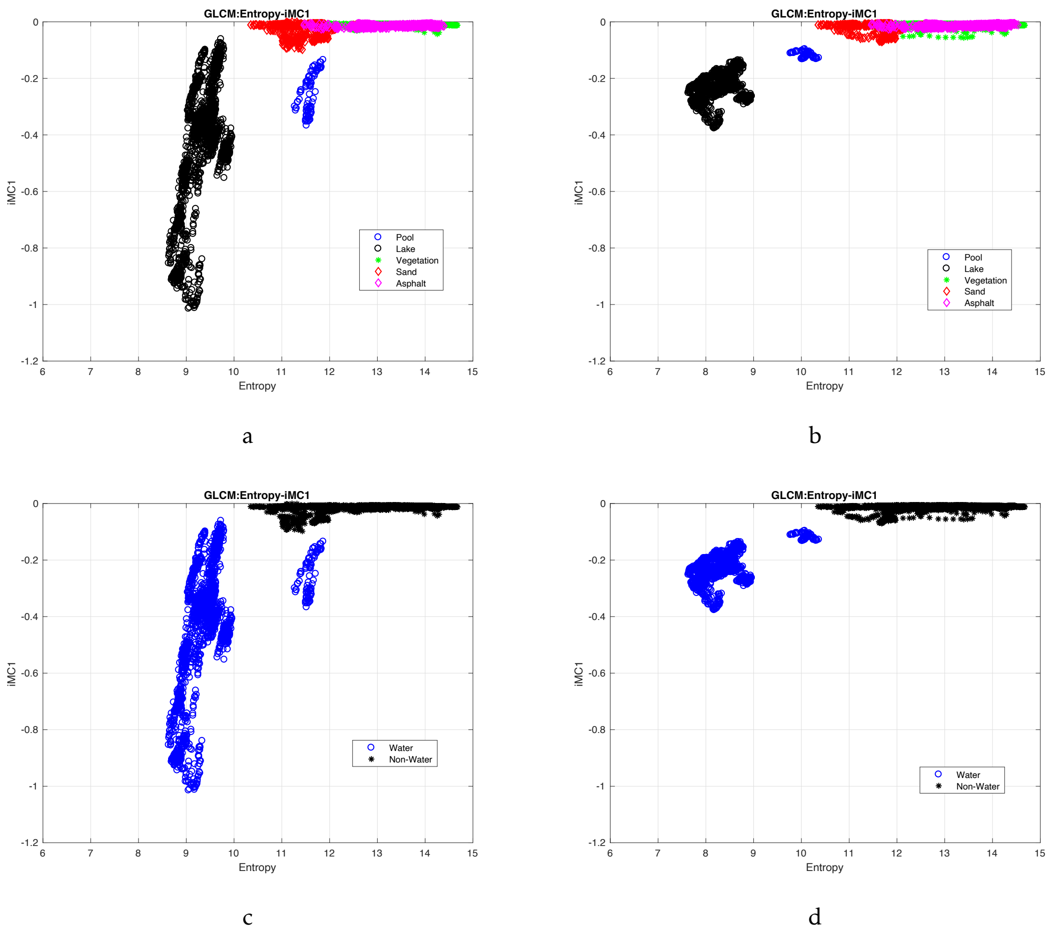


Figure 4.24: GLCM algorithm: a) and c) without downwash effect; b) and d) with downwash effect.

the image under study and was limited to that value;

- **Equals to rows number:** Since the number of pixels varies between 0 and 255, the number of columns was limited to 256 (all values over 255 are neglected);
- **Frame size:** As the run length can exceed the size of 255, the number of columns equal to the image size under study was also tested (in this case, the columns number was equal to 640).

Figure 4.25 shows the differences between the three possible choices for the number of columns in the GLRLM matrix.

As illustrated in Figure 4.25, it is possible to conclude that using the highest pixel number length as columns' number in the GLRLM matrix is not an option to follow because, with this number of columns, it cannot distinguish water from non-water terrains. Between the option of the frame window size being equal to 255, Figures 4.25 b) and c) allowed the author to establish that the GLRLM matrix must have a number of columns equal to the size of the input image (in this case 640), since the goal of this dissertation

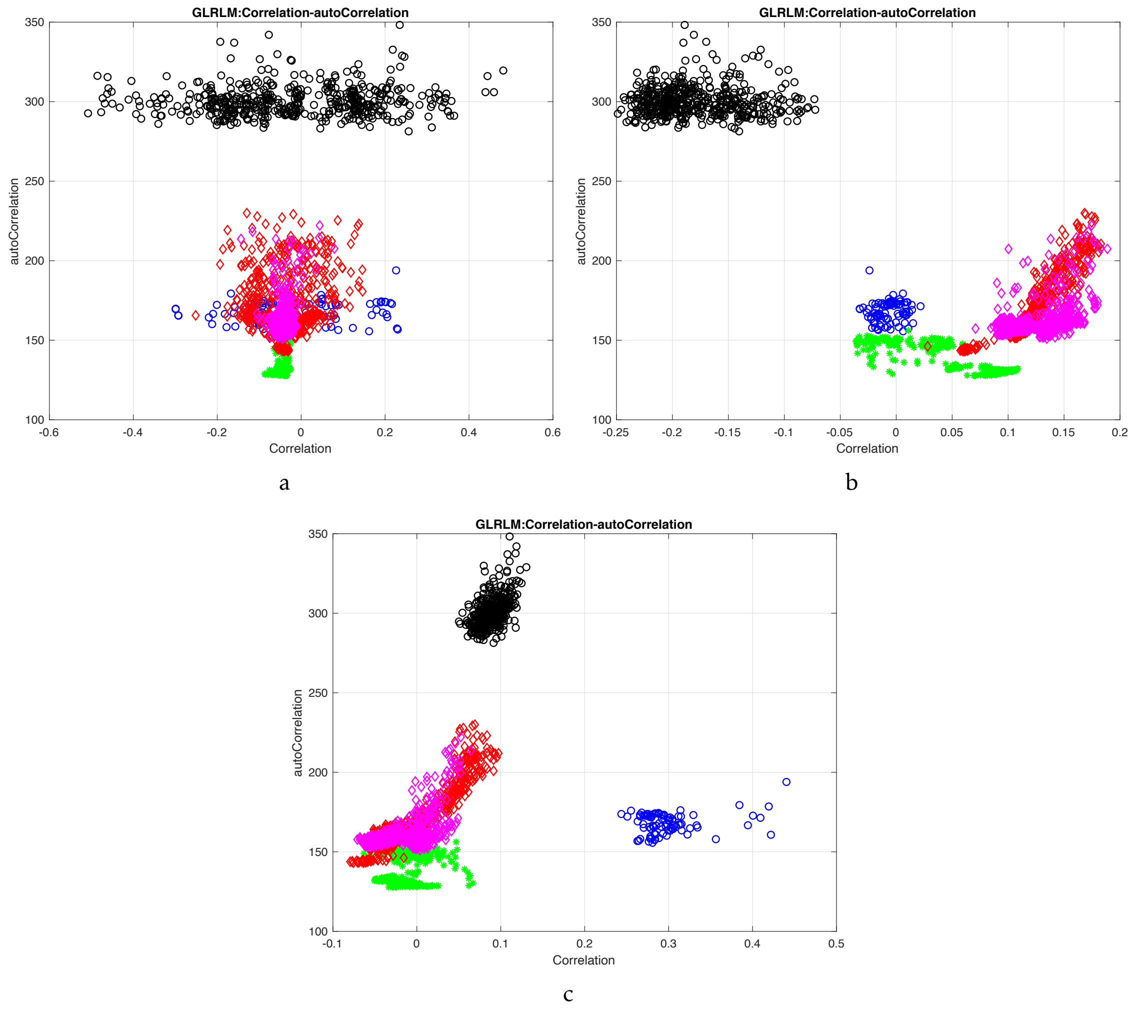


Figure 4.25: GLRLM result: Auto-Correlation with respect to the Correlation. a) GLRLM columns equal to the highest pixel number length; b) GLRLM columns equal to GLRLM rows (255); c) GLRLM columns equal to image width.

is to be able to classify water and non-water-type terrain (water-type terrain is better distinguished from non-water type).

After having the GLRLM matrix's column size defined, it is, it is possible to test the matrix on the terrain under study. The complete data set is composed of 11.121 images where 2.332 are of water, 2.373 are of vegetation, 834 are of asphalt and 5.582 are sand. Then, as carried out previously in Section 4.5, the 27 features presented in Table 3.3 were also used for the calculations in this dataset. The whole set of studied frames' results were analysed and consequently 11 of the 27 combination features were selected by the author. The selected combination features are: 1- Correlation with Contrast; 2- Variance with Sum Variance; 3- Variance with Difference Variance; 4- Sum Average with Sum Variance; 5- Correlation with Homogeneity; 6- Sum Average with Difference Variance; 7- Correlation with Auto-Correlation; 8- Sum Variance with Difference Average; 9- Correlation with Difference Entropy; 10- Difference Variance with Joint Average.

As can be seen in the authors' enumeration of "good" features, it is possible to conclude the impact of features, where:

- Correlation: 40% of impact;
- Sum Variance: 30% of impact;
- Difference Variance: 30% of impact;
- Variance: 20% of impact;
- Sum Average 20% of impact;
- Joint Average 10% of impact;
- Contrast: 10% of impact;
- Difference Entropy: 10% of impact;
- Auto-Correlation: 10% of impact;
- Homogeneity: 10% of impact;
- Difference Average 10% of impact.

Since the Sum Variance feature has been used in more than 50% of cases, the Figures that depend on it will be shown. The Figures (Figure 4.26, Figure 4.27, Figure 4.28 and Figure 4.29) illustrate the results of the whole set of frames concerning the Sum Variance with Sum Average and Difference Average. Water-type terrain is represented by black (pool) and blue (lake), green represents the vegetation-type terrain, sand-type terrain is pink and red represents the asphalt-type terrain, while in Figure b), the water-type terrain is represented by blue (pool and lake) and black represents the non-water-type terrain.

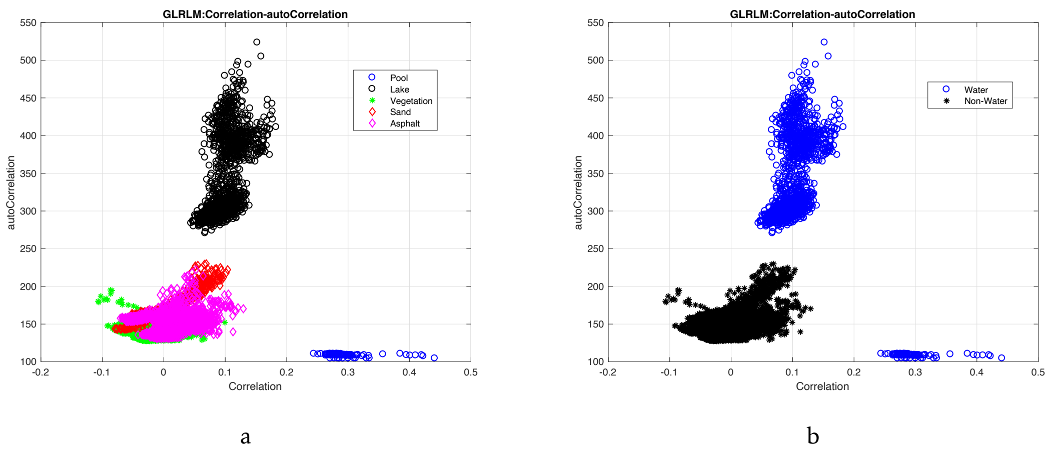
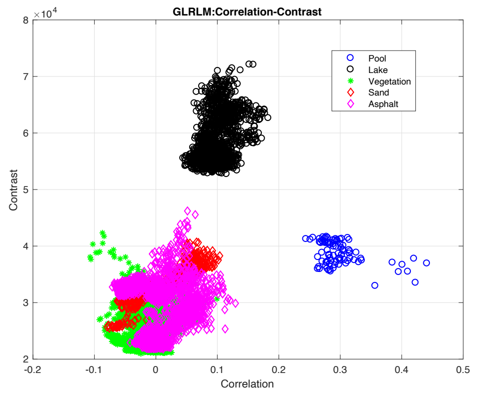
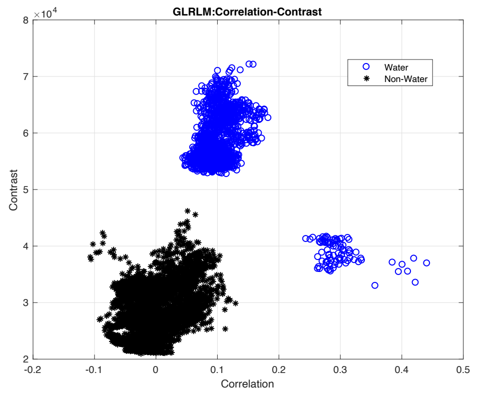


Figure 4.26: Correlation with respect to Auto-Correlation: a) five terrain-type classes and b) water vs non-water.

It is possible to visualize a good identification of each cluster in Figure 4.26, Figure 4.27, Figure 4.28 and Figure 4.29: water (pool and lake) and non-water (vegetation, asphalt and sand) terrain types. Since one of the goals of this dissertation is to identify water-type terrains (distinguish water-type from non-water terrain), it is concluded that the GLRLM algorithm has good results due to the fact that water-type terrain (pool and lake) does not disperse much in the calculated values.

As already mentioned, another goal of this dissertation is to study the impact of the downwash effect on water-type terrain (pool, lake) and non-water type terrain (vegetation,

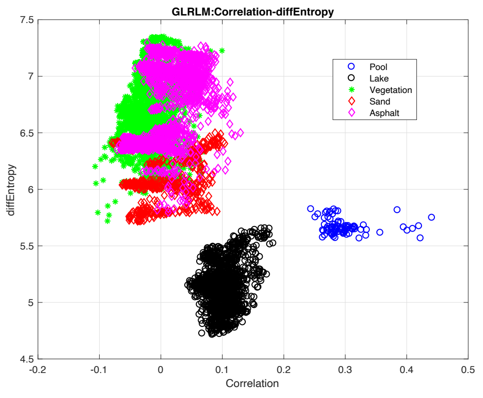


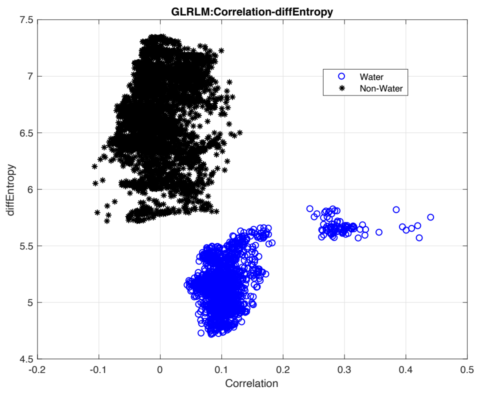


a

b

Figure 4.27: Correlation with respect to Contrast: a) five terrain-type classes and b) water vs non-water.

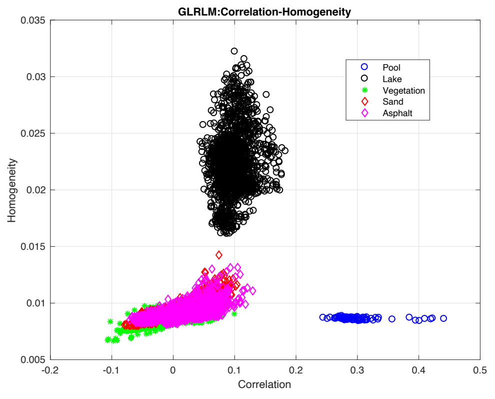


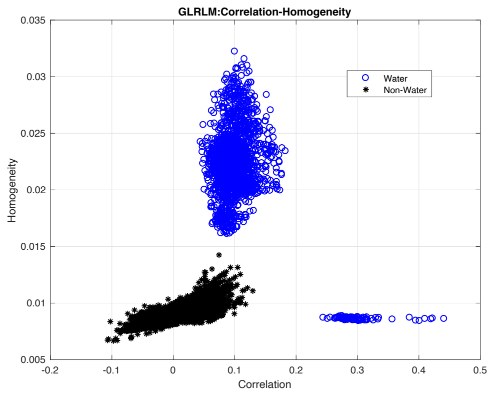


a

b

Figure 4.28: Correlation with respect to Difference Entropy: a) five terrain-type classes and b) water vs non-water.

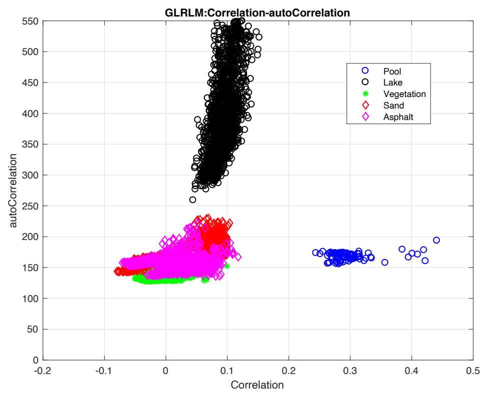




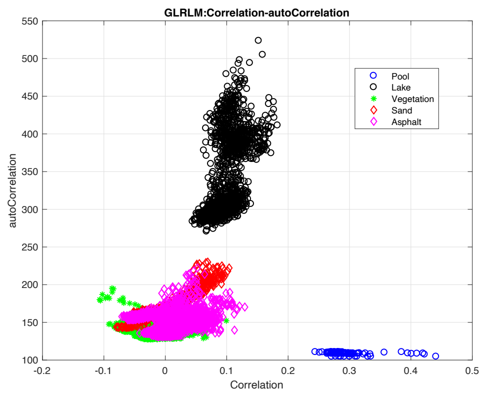
a

b

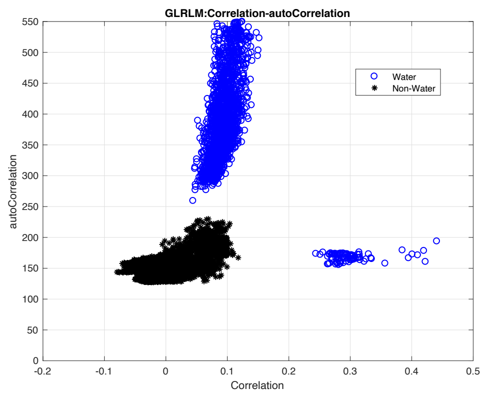
Figure 4.29: Correlation with respect to Homogeneity: a) five terrain-type classes and b) water vs non-water.



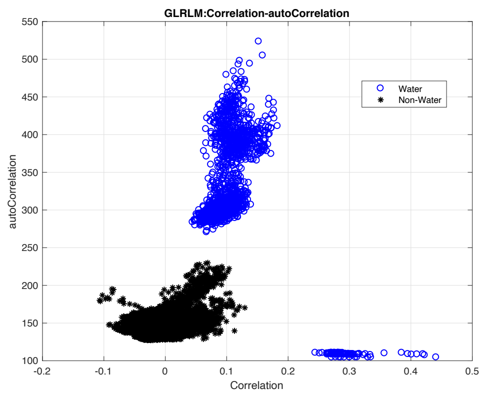
a



b



c



d

Figure 4.30: GLRLM algorithm: a) and c) without downwash effect; b) and d) with downwash effect.

asphalt and sand). Thus, data was taken in the same locations with and without the effect of downwash. Then, the GLRLM algorithm was applied and the results can be seen in Figure 4.30.

In Figures 4.30 a) and c) it can be observed that, without the downwash effect caused by the UAV, in water-type terrain (pool and lake) the values are more dispersed which causes a certain instability to the obtained results. Regarding the downwash effect provoked by UAV rotors in the water-type terrain, the results become more stable and therefore there is greater compression in the outputs generated in water-type terrain, as shown in Figures 4.30 b) and d). Section 4.9 will present the percentage of improvement using the downwash effect in the water-type and non-water-type terrain classification.

4.7 Travel Distance

As mentioned and explained in Section 3.3.1, water-type terrain only exhibits dynamic texture when exposed to the downwash effect. Thus, in this case, when visualizing the effect of downwash on the terrains under study (water-type terrains and non-water-type terrains), the travel distance algorithm is calculated in each flow obtained by a sequence of frames. If the travel distance result is higher than a threshold defined by the author of this thesis, then the counting variable increases. At the end, the variable is divided by the size of the image in order to obtain a percentage in which it is sent to an NN, where the terrain under study will be classified (the NN will classify the terrain under study as water or non-water-type terrain).

After explaining this algorithm very briefly, the quality of algorithms presented in Section 3.3.1, will be studied in more detail in this section and Section 4.9. Figure 4.31 shows the TD results with the dataset of this dissertation.

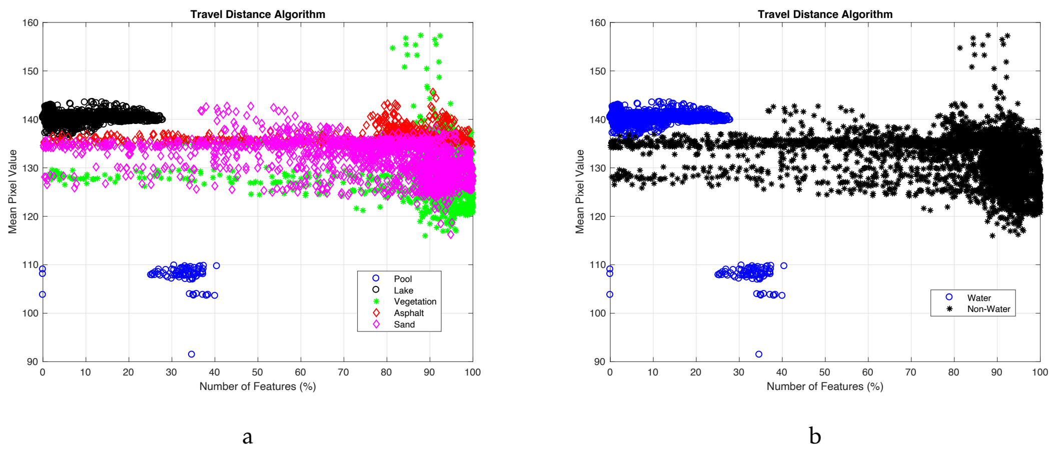


Figure 4.31: Dynamic Texture - Relationship in number of features with respect to mean pixel value: a) five terrain-type classes and b) water vs non-water.

Figure 4.31 illustrates that, with the TD algorithm it is possible to identify and separate the water terrains (pool and lake) from non-water terrains (vegetation, asphalt and sand) where:

- **Pool (water):** Up to 25% at number of features and mean pixel value between 105% and 112%, in general, the pool-type terrain is represented.
- **Lake (water):** Between 0% and 23% at number of features and mean pixel value between 138% and 145% represents the lake-type terrain.

Finally, these features were extracted from the raw terrain frames and shown to the neural network classifier, which outputted the automated terrain classification. The extracted features and the classification result is shown in Table 4.1.

As expected, the proposed features and classification method obtained good results by correctly classifying all seven examples, reinforcing the idea the dynamic texture (i.e. travel distance) can be used to automatically classify between water-type terrain (pool and lake) and non-water-type terrain (vegetation, asphalt and sand) using RGB images.

Table 4.1: Experimental Results.

Figure	Type of Terrain	Number of Features (%)	Mean Pixel Value	Classification
3.65 a)	water	5.10	142	water
3.65 d)	water	30.00	106	water
3.66 a)	vegetation	98.32	126	vegetation
3.66 d)	vegetation	98.69	123	vegetation
3.67 a)	sand	63.74	134	sand
3.67 d)	sand	59.30	130	sand
3.68 d)	asphalt	91.00	138	asphalt

Besides terrain classification, it is also important to study the impact of the downwash effect on water-type terrain and non-water type terrain. Thus, data were taken in the same locations with and without the effect of downwash. Then, the TD algorithm was applied and the results can be seen in Figure 4.32. Two important conclusions can be drawn:

- **Terrain Classification:** Regarding terrain classification between water-type terrain (pool and lake) and non-water-type terrain (vegetation, asphalt and sand), it is observed that without the downwash effect, a greater approximation between these two types of terrains will occur (it is noticed mainly in the lake approaching the sand values on the y-mean pixel value axis). This is due to the fact that, without the downwash effect produced by UAVs, there will no longer be pixels with high intensity values (close to 255). Section 4.9 will discuss the accuracy of this algorithm in greater detail;

- **Downwash Effect:** Being one of the most important topics in this dissertation, it is important to know the impact of the downwash effect on water and non-water terrains. Figure 4.32 shows that, with the downwash effect, a greater dispersion between the values of a number of features occurs. This clearly shows a greater instability in the values.

4.8 Circular Motion

The CM algorithm presents a single feature between the image flows and the center of the intersection. This single feature will count the number of blue, green and gray blocks on

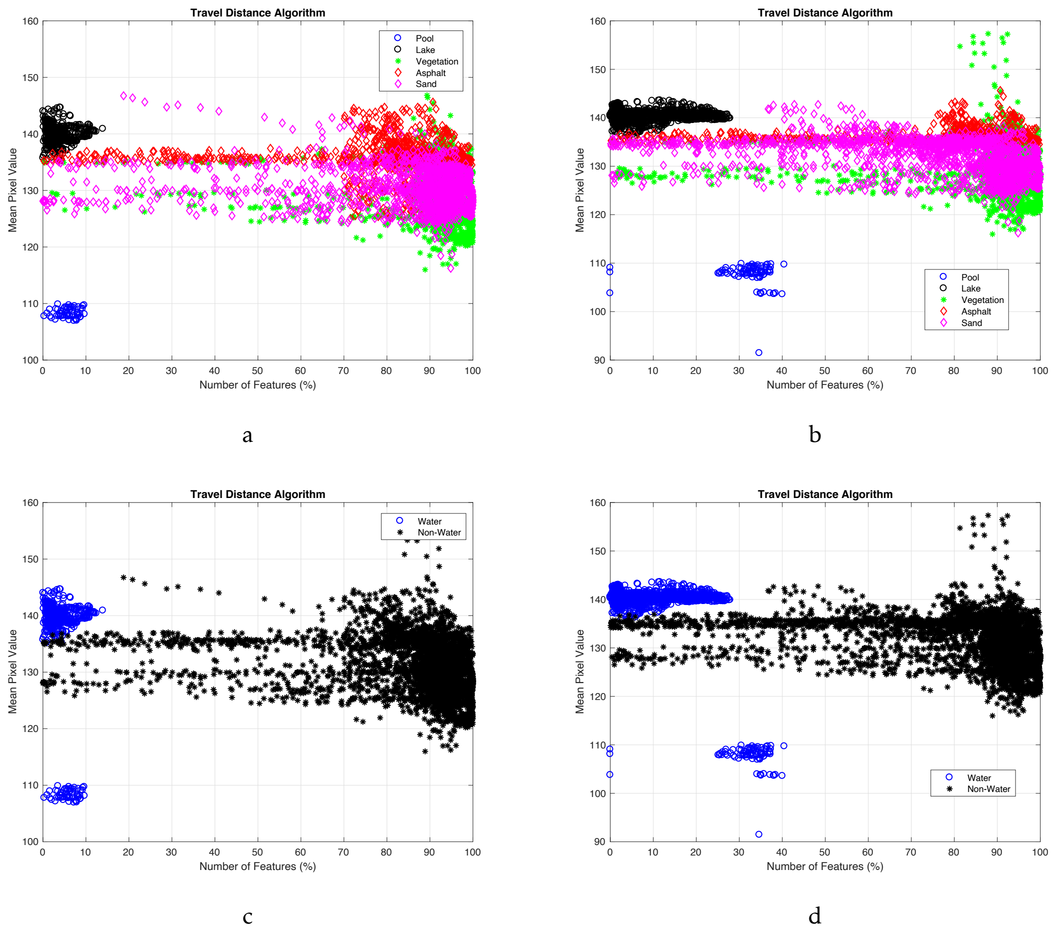


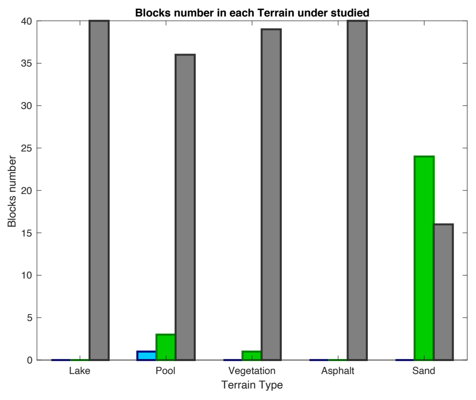
Figure 4.32: Downwash Effect: a) and c) Using TD algorithm without downwash effect; b) and d) Using TD algorithm with downwash effect.

the image result and the system will send this information to the NN as input to classify the terrain under study.

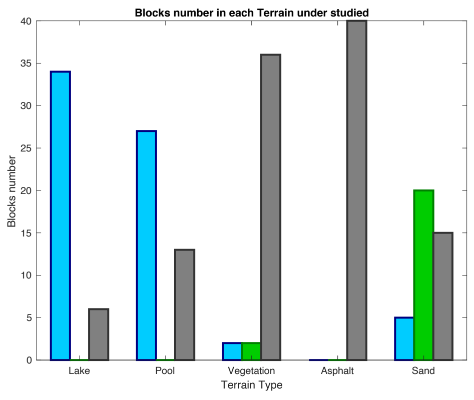
Very briefly, the output generated that will represent the chosen color (blue, green or gray) will be originated from the group of pixels with higher intensity. In water-type terrains, this group of pixels is generated from the effect of the downwash caused by the UAV propellers. Thus, the downwash effect will create a circumference (in water-type terrains) in which the flow tends to point outside the downwash center and therefore will give rise to the blue color.

As mentioned in Section 3.3.2, the images were divided into blocks to ensure a less sensitive analysis of the flows. Thus, in a 640 by 480 image, each block has a width of 80 pixels (eight blocks) and a height of 96 pixels (five blocks) with a total of 40 blocks.

Since this algorithm only returns one type of output (1D information), Tables 4.2 and 4.3 and Figure 4.33 show the experimental results for each of the images (each type of terrain) presented in Section 3.3.2. It should be noted that two images were taken from Section 3.3.2, in which the former was taken without the downwash effect and the latter considered this effect.



a



b

Figure 4.33: Circular Motion Algorithm output: Blue - flows point out from the center of the downwash; Green - flows point towards the center of the downwash; Gray - indicates that the flow size is below a certain given threshold value: a) without downwash effect; b) with downwash effect.

Table 4.2: Experimental Results from Figure 4.33 a) without downwash effect.

Figure	Type of Terrain	Blue Block	Green Blocks	Gray Blocks	Classification
3.74 a)	water	0	0	40	water (lake)
3.74 e)	water	1	3	36	water (pool)
3.75 a)	vegetation	0	1	39	non-water (vegetation)
3.75 e)	asphalt	0	0	40	non-water (asphalt)
3.76 a)	sand	0	24	16	non-water (sand)

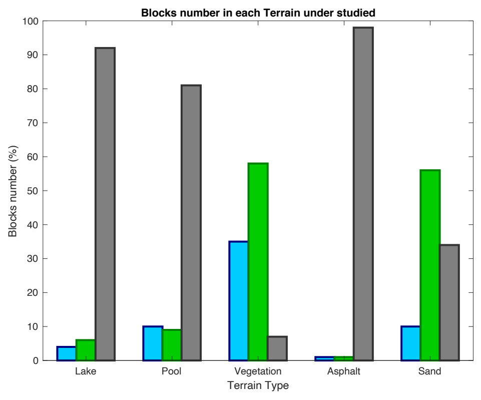
As Tables 4.3 and Figure 4.33 b) demonstrate, water-type terrains present, on average, blue blocks corresponding to the circumference of the downwash effect and, at the same time, gray blocks more noticeable in the center of the downwash effect.

As in the two previous sections on static and dynamic features (Sections 4.1, 4.2, 4.3 4.4, 4.5, 4.6 and 4.7), in addition to the terrain classification between water-type terrain and non-water-type terrains, it is important to know the influence that the downwash effect has on these same terrains. Thus, new data was taken in the same locations without the downwash effect. Then, the CM algorithm was applied and the results presented in Figure 4.34.

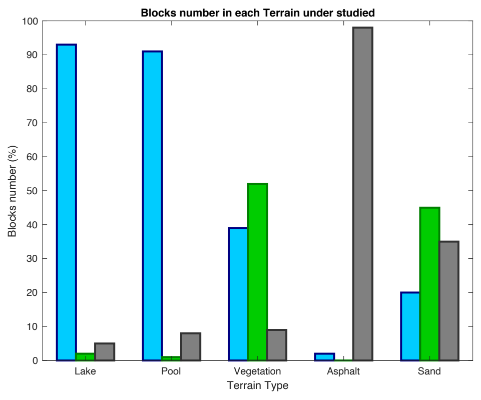
Figure 4.34 concludes that, without the downwash effect, water-type terrains do not

Table 4.3: Experimental Results from Figure 4.33 b) with downwash effect.

Figure	Type of Terrain	Blue Block	Green Blocks	Gray Blocks	Classification
3.74 a)	water	34	0	6	water (lake)
3.74 e)	water	27	0	13	water (pool)
3.75 a)	vegetation	2	2	36	non-water (vegetation)
3.75 e)	asphalt	0	0	40	non-water (asphalt)
3.76 a)	sand	5	20	15	non-water (sand)



a



b

Figure 4.34: Downwash Effect: a) Using the CM algorithm without downwash effect; b) Using the CM algorithm with downwash effect.

present any type of texture due to the fact that, in general, the terrain is always homogeneous (for example, neglecting any external wind effect) either in a lake or in a pool terrain. The probability that the algorithm differentiates water and non-water types is lower (in this case, pool and lake would be relatively equal to the asphalt). However, with the downwash effect it is possible to create dynamic texture in water-type terrains and therefore the probability of correctly classifying the terrain under study increases.

4.9 Combined Results

After explaining the experimental results in Chapter 4, it is in this section that the impact of the downwash effect on water-type and non-water-type terrains will be described mathematically. In addition to showing this impact, the results obtained with the work presented in Section 2.8 Table 2.3 will also be validated.

To evaluate the robustness of the algorithms proposed in this dissertation, a classifier of discriminant analysis was needed: Linear Discriminant Analysis (LDA).

Linear discrimination is a classic classifier of discriminant analysis (Hastie et al., 2009) that seeks to separate classes using linear decision surfaces. This classifier has also, as its main advantages, a low computational demand and the possibility of classifying multiple classes.

The classification of a sample in a class is given by:

$$\begin{bmatrix} A_1 & A_2 & A_3 & \cdots & A_n \\ B_1 & B_2 & B_3 & & \\ C_1 & C_2 & C_3 & & \\ \vdots & & & \ddots & \\ N_1 & & & & N_n \end{bmatrix} = \begin{bmatrix} A\mu_{I_1} & A\mu_{I_2} & A\mu_{I_3} & \cdots & A\mu_{I_n} \\ B\mu_{I_1} & B\mu_{I_2} & B\mu_{I_3} & & \\ C\mu_{I_1} & C\mu_{I_2} & C\mu_{I_3} & & \\ \vdots & & & \ddots & \\ N\mu_{I_1} & & & & N\mu_{I_n} \end{bmatrix} \div \begin{bmatrix} Apc_1 & Apc_2 & Apc_3 & \cdots & Apc_n \\ Bpc_1 & Bpc_2 & Bpc_3 & & \\ Cpc_1 & Cpc_2 & Cpc_3 & & \\ \vdots & & & \ddots & \\ Npc_1 & & & & Npc_n \end{bmatrix} \tag{4.1}$$

Where $N\mu_n$ represents the group mean matrix and Npc_n is the accumulate pooled covariance information matrix. Next, the result of this division will be multiplied by an identity matrix that has one more column $(N_n + 1)$ and the first element is equal to zero, in order to have the result matrix first column equal to zero too.

$$\begin{bmatrix} 0 & A_1 & A_2 & A_3 & \cdots & A_n \\ 0 & B_1 & B_2 & B_3 & & \\ 0 & C_1 & C_2 & C_3 & & \\ \vdots & \vdots & & & \ddots & \\ 0 & N_1 & & & & N_n \end{bmatrix} = \begin{bmatrix} A_1 & A_2 & A_3 & \cdots & A_n \\ B_1 & B_2 & B_3 & & \\ C_1 & C_2 & C_3 & & \\ \vdots & & & \ddots & \\ N_1 & & & & N_n \end{bmatrix} \cdot \begin{bmatrix} 0 & 1 & 0 & 0 & \cdots & 0 \\ 0 & 0 & 1 & 0 & & \\ 0 & 0 & 0 & 1 & & \\ \vdots & & & & \ddots & \\ 0 & & & & & 1 \end{bmatrix} \tag{4.2}$$

After that, with the equation 4.2 result, it is possible to add it to an $N_n x N_n$ matrix where the constants different from zero are in the first column. According to equation 4.3, these constants are placed in the first column of the equation 4.2 result matrix.

$$\begin{bmatrix} A_c & A_1 & A_2 & A_3 & \cdots & A_n \\ B_c & B_1 & B_2 & B_3 & & \\ C_c & C_1 & C_2 & C_3 & & \\ \vdots & \vdots & & & \ddots & \\ N_c & N_1 & & & & N_n \end{bmatrix} = \begin{bmatrix} 0 & A_1 & A_2 & A_3 & \cdots & A_n \\ 0 & B_1 & B_2 & B_3 & & \\ 0 & C_1 & C_2 & C_3 & & \\ \vdots & \vdots & & & \ddots & \\ 0 & N_1 & & & & N_n \end{bmatrix} + \begin{bmatrix} A_c & 0 & 0 & 0 & \cdots & 0 \\ B_c & 0 & 0 & 0 & & \\ C_c & 0 & 0 & 0 & & \\ \vdots & & & & \ddots & \\ N_c & & & & & 0 \end{bmatrix} \tag{4.3}$$

where the constants of equation 4.3 are the result of a multiplication between the matrix resulting from equation 4.1 and the transposed group mean matrix. This multiplication will be also multiplied by a constant -0.5 and added to the logarithm of the probability of each class in the training sample.

Along with the developed LDA algorithm, it is possible to evaluate the robustness of the algorithms proposed in this dissertation. Table 4.4 shows the results of each algorithm to classify water-type and non-water-type terrains and it also shows the impact of using the downwash effect.

Table 4.4: LDA of the seven proposed algorithms in this dissertation with and without downwash effect (DW means the presence of downwash effect and PI means Performance Improvement).

	Classification (%)											
	Water (%)			Non-Water (%)			Normal Average (%)			Weighted Average (%)		
Feature	no DW	DW	PI	no DW	DW	PI	no DW	DW	PI	no DW	DW	PI
Gabor	72.22	84.44	12.22	89.79	90.96	1.17	81.01	87.70	6.69	84.42	88.97	4.55
EMD	32.36	49.88	17.52	95.44	99.76	4.32	63.90	74.83	10.93	76.15	84.52	8.37
W-K Filter	16.97	48.95	31.98	95.46	97.00	1.54	56.22	72.98	16.76	71.47	82.31	10.84
GLCM	65.00	97.90	32.9	98.00	99.56	1.56	81.50	98.73	17.23	87.91	99.05	11.14
GLRLM	98.56	98.89	0.33	98.20	99.00	0.8	98.38	98.95	0.57	98.31	98.97	0.66
TD	70.81	98.33	27.52	90.51	90.78	0.27	80.66	94.56	13.9	84.49	93.09	8.6
CM	22.12	99.30	77.18	97.90	99.10	1.2	60.01	99.20	39.19	74.73	99.16	24.43

Where DW means the presence of downwash effect and PI means Performance Improvement. Since the number of images for each terrain type is not balanced, two types of calculations were made: Normal Average - calculates the probability of correctly classifying without considering the images number portion of each type of terrain; Weighted Average - calculates the probability of correctly classifying considering the images number portion of each type of terrain;

As displayed in Table 4.4, it is possible to reach two important conclusions:

1. **Downwash Effect:** From Table 4.4 in the "Water" column, it is possible to see that the downwash effect caused by UAVs has a greater impact on dynamic (TD and CM) than static (Gabor, EMD, W-K Filter, GLCM and GLRLM) texture algorithms, with the greatest impact occurring in CM algorithm with an improvement of 77.18% for classifying water-type terrain. It is possible to conclude that, in general, the effect of downwash helps to classify water-type and non-water-type terrains, proving the veracity of the hypothesis proposed in Section 1.3;

2. **Terrain Classification:** It is also possible to conclude from Table 4.4 that from the seven proposed algorithms, the one that present the best results, both in the classification of water and non-water types, is the CM algorithm with an accuracy of 99.16% Weighted Average. It reinforces the idea that the dynamic texture algorithms are more reliable than the static ones due to the fact that the static ones do not have the movement perception that help in distinguishing terrains.

In addition to assessing the individual robustness of each algorithm combinations were also made between these algorithms in order to know which would be the best combinations to help in the classification of water and non-water terrain types. The LDA was once again applied in the several possible combinations. The four algorithms with the highest success rate in the classification were chosen. Table 4.5 shows the combination between these algorithms and their respective accuracy results.

Table 4.5: LDA combination between GLCM, GLRLM, TD and CM algorithms proposed and developed in this dissertation using the downwash effect.

	Features				**Classification (%)**			
LDA	**GLCM**	**GLRLM**	**TD**	**CM**	**Water (%)**	**Non-Water (%)**	**Average (%)**	**Weighted Average (%)**
1	X	X			98.15	**91.42**	94.79	93.48
2	X		X		99.06	98.71	98.89	98.82
3	X			X	91.27	**99.59**	95.43	97.05
4		X	X		99.13	94.85	96.99	96.16
5		X		X	96.32	98.00	97.16	97.49
6			X	X	**90.98**	91.58	**91.28**	**91.40**
7	X	X	X		92.78	94.22	93.50	93.78
8	X	X		X	95.47	99.16	97.32	98.03
9		X	X	X	99.58	97.92	98.75	98.43
10	X	X	X	X	**99.65**	99.57	**99.61**	**99.59**

From Table 4.5, it is possible to conclude that using only the static features (GLCM and GLRLM) or only the dynamic ones (TD and CM), are not good combinations for classifying the terrain types being studied:

- It appears that if one only uses the static features, the system has the worst accuracy in classifying non-water-type terrains (in this case vegetation, asphalt and sand) with 91.42% accuracy;

- It is also concluded, from Table 4.5, that if one only uses dynamic features, the system is less likely to correctly classify water-type terrains, obtaining a 90.98% of accuracy.

It is necessary to have a mixture of these two features in order to improve the classification of the terrain type over which the UAV is flying. In this case, the best solution is to combine these four features, obtaining a percentage of 99.65% 99.57% in the classification of water-type and non-water-type terrain, respectively.

After showing the study of each algorithms developed in this dissertation with the presence or non-presence of the downwash effect caused by the UAV, as well as the study of the best combination for the classification of water and non-water types, the next step will be to compare the results obtained in this section with the results of other similar works presented in Table 2.3 of Section 2.8. Once again, each work compares the respective success rates for each terrain type as well providing an overall system average (in the column on the right). The average does not always consider the four terrain types under study, but each work's terrain types instead, since in some of the related works the proposed algorithms do not test all four terrain types.

Comparing Table 4.5 in this section to Table 2.3 of Section 2.8, it is possible to conclude that the system proposed in this dissertation offers the best accuracy for classifying water and non-water terrain types.

One of the main advantages of this work, when compared to (Sofman et al., 2006), (Salvado, 2018) and (Wang et al., 2019), is the ability to obtain a better classification with the aid of the downwash effect, when the UAV is only one to two meters above the ground. In (Sofman et al., 2006), (Salvado, 2018) and (Wang et al., 2019), images are taken by satellites or at a distance of more than 60 meters above the ground, which increases the error in classifying the terrain types being studied.

One of the strengths of the current work is the use of the terrain's dynamic texture for terrain classification. It clearly shows improved system accuracy, since each terrain behaves differently when affected by the UAV's downwash effect. According to the works cited in Table 2.8, only (Pombeiro et al., 2015) used the dynamic texture feature, observing that the proposed system's accuracy is greater than most articles presented in Table 2.3. Compared to (Pombeiro et al., 2015), this dissertation proposes different static and dynamic texture features, in order to classify water and non-water terrain types with increased accuracy.

4.10 GPU Acceleration

Since processing time is an important factor in terrain classification, the author of this dissertation also studied the GPU capabilities in the GLCM matrix design using the CUDA framework.

As explained in Section 3.2.5, to build the GLCM matrix, the pixel values are used as coordinates of the matrix itself. Therefore, if one or more threads try to write in the same coordinates of the GLCM matrix, an undefined result will occur. To solve this, atomic operations are needed to serialize operations and ensure that the latest value is read. Hence, whenever there is more than one thread writing at the same position in the GLCM matrix, priority will be given to the first thread and so on. This solution has an advantage and a disadvantage:

- **Advantage:** The atomic operation ensures that there are no undefined values and the final result is correct;
- **Disadvantage:** The disadvantage centers on the fact that the parallelism gets lost in these situations of having to write in the same memory. In this way the processing time is reduced.

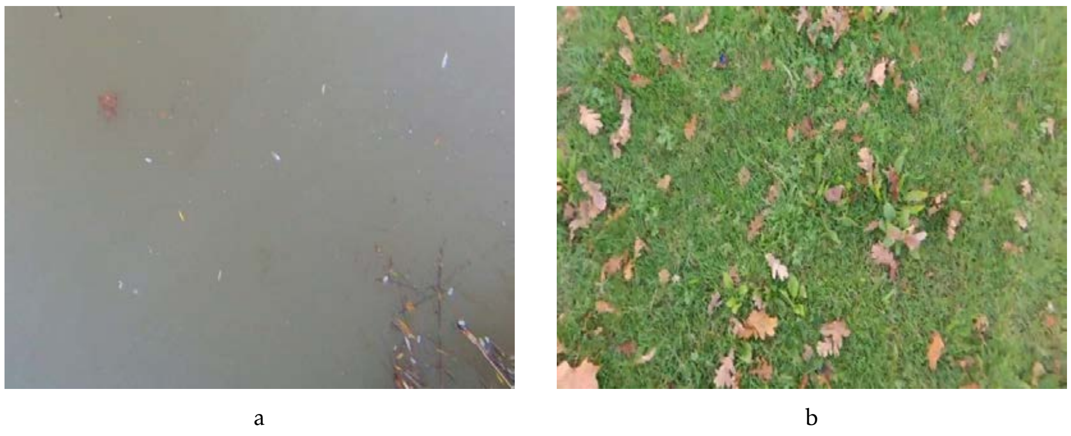


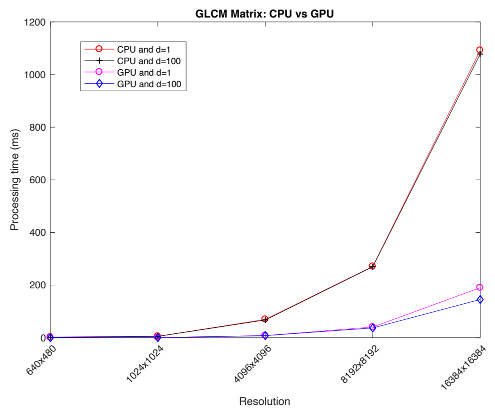
Figure 4.35: Examples of terrain types: water (a); vegetation (b).

To verify this effect in detail (several threads writing in the same memory at the same time), the CUDA algorithm ran on two images, as shown in Figures 4.35 a) and b), and the GLCM matrix processing time in both figures was registered. With the same image dimensions ($Width = 640$ and $Height = 480$), and the same number of blocks and threads in each block, the following couple statements were concluded: the processing using CUDA programming in Figure 4.35 a) is 0.230 ms, while in Figure 4.35 b) it is 0.195 ms; therefore, as in Figure 4.35 b), the processing time is 1.18 times faster than in Figure 4.35 a). These results are due to the fact that in water-type terrains, the pixel values are close to each other (much more homogeneous image), than vegetation-type terrains where there is a greater diversity of pixel colors. There is a greater probability that the threads will write in the same memory position when studying water-type terrains and consequently suffer from poorer processing-time performance.

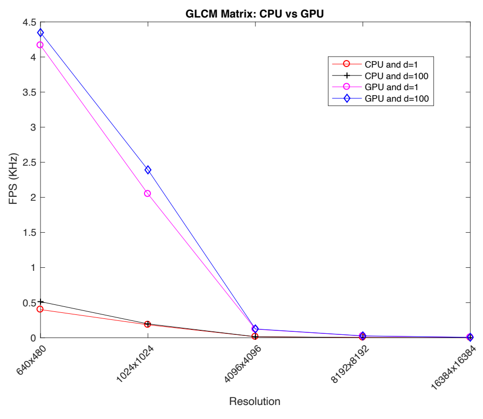
The difference in processing time when programmed in CPU and GPU concerning the GLCM matrix was also studied in detail. Figure 4.35 a) was used as a case study and the difference was analyzed for different resolutions, maintaining the same number of threads in each block and a ratio of grid blocks of $Width/threads$ in X and $Height/thread$ in Y. Table 4.6 and Figure 4.36 show the results obtained.

From Table 4.6 and Figures 4.36 a) and b) it is possible to conclude that using the capacities of the GPU always results in a better processing time regardless the image size and the distance parameter for the developed GLCM matrix.

As illustrated in Figures 4.36 a) and b), it is possible to make some important observations. Regarding resolution, as it increases, the processing time will also increase when



a



b

Figure 4.36: CPU vs GPU: Processing time in relation to resolution (a); FPS in relation to the resolution (b).

Table 4.6: Processing time for different resolutions and GLCM distance parameter.

Width	Height	Distance	CPU (ms)	GPU (ms)	Speed up
640	480	1	2.491	0.240	10.379
		100	1.944	0.230	8.452
1024	1024	1	5.408	0.488	11.082
		100	5.067	0.418	12.122
4096	4096	1	69.617	8.194	8.496
		100	67.392	7.965	8.461
8192	8192	1	271.039	40.576	6.680
		100	269.634	36.995	7.288
16384	16384	1	1092.070	190.093	5.745
		100	1078.060	145.195	7.425

programming using the CPU and GPU, mainly in the 16384x16384 resolution where the processing time using CPU is approximately 1 second, and 0.190 seconds with GPU (with the distance parameter equal to 1). It is also possible to observe that as the resolution of the image increases, the processing time increases exponentially both in the CPU and GPU. It is also known that generally the higher the resolution of the image, the greater the accuracy of the terrain classification (it is possible to obtain much more data in the same image). It is necessary to have a trade-off between the system accuracy and the algorithm processing time.

4.11 Mapping

After the classification of terrain types (water and non-water), it is necessary to georeference the results obtained by the NN. As already mentioned in Section 3.5, using the ROS framework it is possible to build a dynamic map where all the information obtained by the neural network is allocated to the map. Therefore, in order to validate the dynamic mapping algorithm, a flight test was carried out in Parque da Paz, Portugal. In this test, the UAV performed a mission where it would fly over water-type terrain (lake) and non-water-type terrain (vegetation) as shown in Figure 4.37.

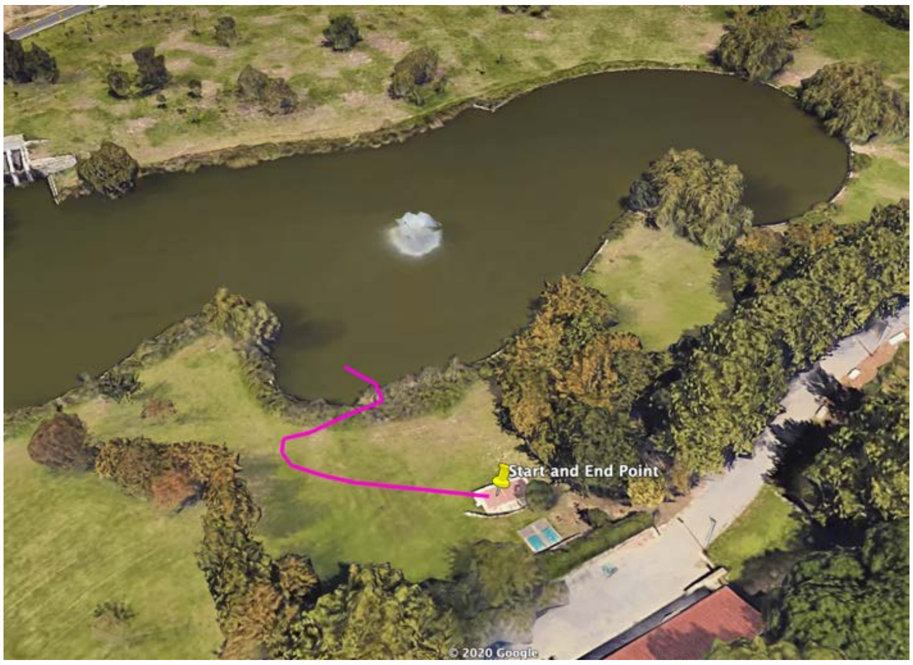


Figure 4.37: Mission test in Parque da Paz, Portugal.

As the UAV flies over different types of terrain, it sends the images via wi-fi (using the ROS framework) to the machine responsible for running the algorithms for the terrain classification. When the result of each classification is generated, a dynamic map is created as follows:

1. **Black:** represents the water-type terrain;

2. **White:** represents the non-water-type terrain;
3. **Gray:** represents the unknown-type terrain.

Figure 4.38 shows the georeferenced dynamic map result. It is possible to conclude that the dynamic mapping algorithm was able to georeference the outputs generated from each image from the UAV. It should be noted that there are different sizes in the output generated due to the different altitudes at which the UAV flew during the mission. It should also be noted that there are certain gaps in each output generated due to the lower UAV GPS resolution.

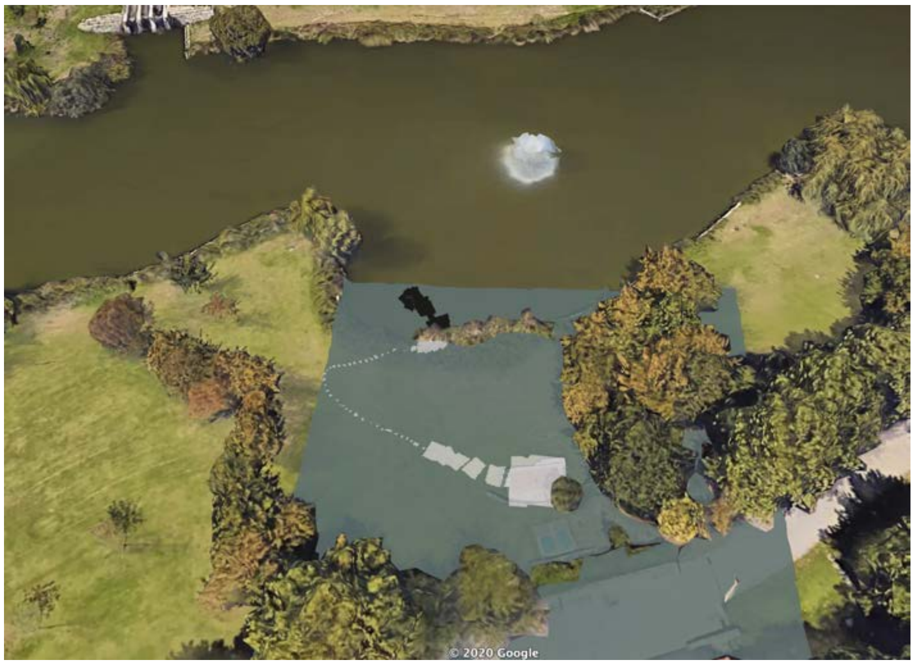


Figure 4.38: Georeferenced dynamic map result in Parque da Paz, Portugal.

Chapter 5

Conclusions and Future Work

This chapter summarizes the main contributions obtained through the dissertation results and proposes some guidelines for future research on the theme, highlighting in Section 5.2 the points where the developed application can be improved.

5.1 Conclusions

The main goal of the work presented in this dissertation was to study and develop textural features that would allow the classification of water and non-water terrain types using the downwash effect caused by the UAV's rotors.

In an initial phase, an intensive literature review was carried out, focused on features extraction that could be used to classify the terrain under study, such as color, texture, dynamism, frequency and spatial domains and deep learning. Chapter 2 provided an overview of the existing terrain classification methods. The pros and cons of design's were also discussed in detail.

Concerning the downwash effect, the main goal was to study and analyse how it could be used to classify water terrains and differentiate this terrain from other non-water terrains. Thus, static and dynamic texture features were developed in order to extract water characteristics and their natural behaviors, such as the circular effect caused by the UAV when it is flying over this terrain type, namely:

- 1^{st} **Static Algorithm:** Gabor with Lowess Regression;
- 2^{nd} **Static Algorithm:** PSO;
- 3^{rd} **Static Algorithm:** EMD;
- 4^{th} **Static Algorithm:** W-K Filter;

- 1st **Dynamic Algorithm:** TD algorithm;
- 2nd **Dynamic Algorithm:** CM algorithm.

Throughout each section, a detailed presentation of each algorithm was provided, and, in the experimental results chapter, a comparative analysis of the algorithms' performance when terrain exposed or not to the downwash effect, was carried out. The tests were conducted on simulated and real environments. The obtained results suggest that the proposed system is able to identify water and non-water terrain types.

However, it is important to mention that the proposed system has limitations. There are several factors that may influence the terrain type classification, namely:

- **Image acquisition time:** the image dataset used for testing the proposed algorithms did not contain different times of the day, what can originate different lightning condition that influence the classification results;
- **UAV flying altitude:** the UAV altitude estimator (sensors) may be prone to errors and if the UAV is above the necessary altitude to generate the downwash effect, the terrain type classification system may fail;
- **UAV motors:** depending on the UAV motor types, the downwash effect may be almost non-existent; for this reason, it should be guaranteed that the UAV setup is adequate;
- **Sea-Type terrains:** In sea-type terrain, as illustrated in Figure 5.1, it is possible to observe that the UAV propellers do not have enough strength to cause the circular effect that is expected to be created in water-type terrains. This happens when the existing waves and wind have more impact than the UAV air flow.

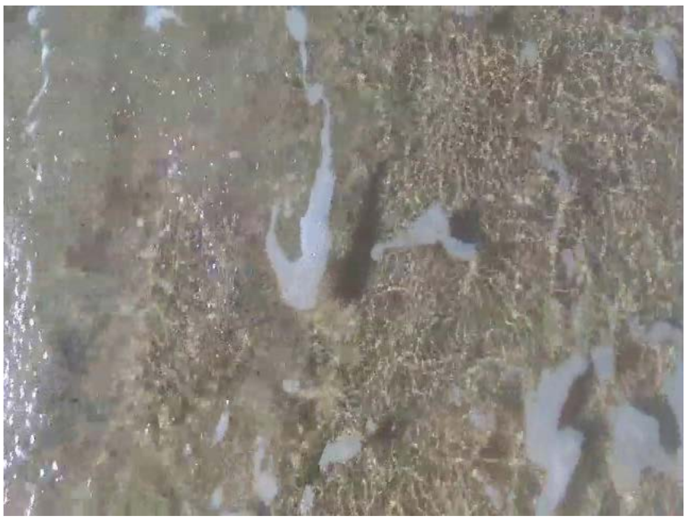


Figure 5.1: Water Terrain: Sea-type terrain.

Apart from the environmental and material factors, the algorithms proposed in this dissertation may find some limitations, such as:

- **Gabor with Lowess Regression:** this algorithm is very sensitive to the downwash effect. If this effect is not strong, the Gabor filter will not be able to extract many features and, therefore, the Lowess Regression may not work correctly, resulting in wrong classifications;

- **W-K Filter:** this algorithm is rotation invariant, as it does not change with the UAV rotation; however, it is still sensitive to the UAV angular movement that may (wrongly) influence the terrain type classification. To minimize this problem it was decided to use a gimbal;

- **GLCM and GLRLM:** despite being second level statistical algorithms, they are sensitive to the image intensity; if the images are acquired in a low-light environment, the GLCM and GLRLM may (wrongly) classify the studied terrain;

- **TD and CM:** these are algorithms that extract dynamically features from the terrain, so it is essential to capture only the studied terrains movement while neglecting the UAV movement. If the former UAV movement is considered, the features' dynamic extraction algorithms will not work correctly. At this critical moment, the next couple approaches were followed: gimbal usage for image stabilization; and having the UAV in hold mode on, during at least 0.5 seconds, for each waypoint from the UAV mission, in order to not have interference between the terrains' movement with its own.

Knowing each algorithm limitations, it is also important to know which algorithms are more robust when used with the downwash effect. The second level statistical algorithms, GLCM and GLRLM, were the ones that extracting static texture features and in the presence of the downwash effect presented better results. However, dynamic texture features shown more robustness in discriminating both water and non-water terrains.

The behavior of the combination of these algorithms was also considered and studied. It was concluded that using only static features or only dynamic ones, are not the best combinations for classifying the terrain type. Using only the static texture features, the system has the worst accuracy in classifying non-water-type terrain (in this case vegetation, asphalt and sand) with 91.42% accuracy; on the other hand, using only dynamic texture features, the system is less likely to correctly classify water-type terrains, obtaining a 90.98% of accuracy. Thus, it was concluded that a mixture of static and dynamic texture features was necessary to improve the terrain-type classification. In this dissertation, the best combination was obtained by merging the GLCM; GLRLM; TD and CM algorithms, resulting in a accuracy of 99.65% and 99.57% in the classification of water and non-water type terrain, respectively.

In order for the UAV to take advantage of the onboard processing it is necessary to have a system with good accuracy to classify the terrain types, and at the same time it must be capable of running in real time. To speed up the image processing GLCM algorithm was redesigned to be parallelized and consequently be able to run in the onboard GPU. The CUDA framework was used to make the sequential image process into parallel processing mode. Using this CUDA framework, the GLCM algorithm was studied in greater depth, achieving speedups up to 10x. It was also validated that using the GLCM algorithm the processing time with GPU resources varies with the image homogeneity. This means that in a more homogeneous image, the processing time will increase due to the attempt to write in the same memory position from several threads at the same time. In a more heterogeneous image, there are no longer so many pixels with the same value and therefore it will not be written in the same memory position by several threads, which speeds up the image processing.

The last described topic in this dissertation is georeferenced mapping. This topic was able to dynamically georeference the terrain classified by the system (determine if it is water or non-water-type terrain) and send it to an end user who needs it. As such, autonomous cooperation between robots is promoted, providing the main motivation behind this dissertation. However, it is important to emphasize that building an accurate georeferenced dynamic map depends on the GPS sensors, IMU for altitude and on the intrinsic and extrinsic camera characteristics. These parameters must be carefully calibrated so that the map matches the actual environment.

5.2 Future Work

Despite the good results obtained so far, there are several areas in which improvements are foreseen. The author of this dissertation plans to further investigate the following topics in the near future:

- Perform a more in-depth study on changing the environment colors, and improving the robustness of the algorithm. Color variation could influence the results with false positives/negatives. Although the dynamic texture does not suffer much from these changes, due to the similarity in the way the terrain moves, this could highly affect the static texture;

- Since the algorithm was designed for a UAV flying at an altitude of between one and two meters, it is likely that at this altitude the UAV may collide with objects in the environment. The author of this thesis has developed an algorithm (Carvalho et al., 2020) to avoid obstacles, from images taken from a depth camera. However, currently it is only working with static objects. An improved version of this algorithm could be used to support the autonomous navigation of the proposed system;

- It is also important to make a more in-depth study into different camera resolutions in order to improve the robustness of the proposed system. However, the system speed may be affected by higher resolutions, as explained in Section 4.10.

Bibliography

Aaron (2016). *ROS Answers: What is a nodelet?* url: http://wiki.ros.org/ros_comm (visited on 03/12/2019).

Acharya, U. R., P. Chowriappa, H. Fujita, S. Bhat, S. Dua, J. E. W. Koh, L. Eugene, P. Kongmebhol, and N. Kh (June 2016). "Thyroid Lesion Classification in 242 Patient Population Using Gabor Transform Features from High Resolution Ultrasound Images". In: *Knowledge-Based Systems* 107. doi: 10.1016/j.knosys.2016.06.010.

Ahonen, T., A. Hadid, and M. Pietikäinen (Jan. 2007). "Face Description with Local Binary Patterns: Application to Face Recognition". In: *IEEE transactions on pattern analysis and machine intelligence* 28, pp. 2037–41. doi: 10.1109/TPAMI.2006.244.

Ahuja, N. (1982). "Dot Pattern Processing Using Voronoi Neighborhoods". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* PAMI-4.3, pp. 336–343. issn: 0162-8828. doi: 10.1109/TPAMI.1982.4767255.

Akhloufi, M. A. and A. Bendada (2010). "Locally adaptive texture features for multispectral face recognition". In: *2010 IEEE International Conference on Systems, Man and Cybernetics*, pp. 3308–3314. doi: 10.1109/ICSMC.2010.5642391.

Andrearczyk, V. (2017). "Deep learning for texture and dynamic texture analysis". In:

AUVSI (2015). *The Benefits of Unmanned Aircraft Systems.* url: https://epic.org/events/UAS-Uses-Saving-Time-Saving-Money-Saving-Lives.pdf (visited on 03/12/2019).

Bai, C., J. Guo, L. Guo, and J. Song (2019). "Deep Multi-Layer Perception Based Terrain Classification for Planetary Exploration Rovers". In: *Sensors* 19.14, p. 3102. issn: 1424-8220. doi: 10.3390/s19143102. url: http://dx.doi.org/10.3390/s19143102.

Barron, J., D. Fleet, and S. Beauchemin (Feb. 1994). "Performance Of Optical Flow Techniques". In: *International Journal of Computer Vision* 12, pp. 43–77. doi: 10.1007/BF01420984.

B.WilsonJr., H. and D. S.Farrior (Oct. 1976). "Computation of geometrical and inertial properties for general areas and volumes of revolution". In: *Science Direct* Volume 8, pp. 257–263.

Camarinha-Matos, L. (2000). "The handouts of the Scientific Research Methodologies and Technologies course of the PhD program in Electrical and Computer Engineering".

Campos, I. S. G., E. R. Nascimento, and L. Chaimowicz (2015). "Terrain Classification from UAV Flights Using Monocular Vision". In: *2015 12th Latin American Robotics Symposium and 2015 3rd Brazilian Symposium on Robotics (LARS-SBR)*, pp. 271–276.

Canny, J. (1986). "A Computational Approach to Edge Detection". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* PAMI-8.6, pp. 679–698. ISSN: 0162-8828. DOI: 10.1109/TPAMI.1986.4767851.

Cargyrak (2016). *Attribution-ShareAlike 4.0 International (CC BY-SA 4.0).* URL: https://upload.wikimedia.org/wikipedia/commons/1/10/Onyxstar_HYDRA-12_UAV_with_embedded_hyperspectral_camera_for_agricultural_research.jpg (visited on 03/12/2019).

Caridade, C., A. Marçal, and T. Mendonça (Jan. 2008). "The use of texture for image classification of black & white air photographs". In: *International Journal of Remote Sensing - INT J REMOTE SENS* 29, pp. 593–607. DOI: 10.1080/01431160701281015.

Carvalho, J., D. F. Pedro, L. Campos, J. Fonseca, and A. Mora (Jan. 2020). "Terrain Classification Using W-K Filter and 3D Navigation with Static Collision Avoidance". In: pp. 1122–1137. ISBN: 978-3-030-29512-7. DOI: 10.1007/978-3-030-29513-4_81.

Castellano, G., L Bonilha, L. Min, and F. Cendes (Jan. 2005). "Texture analysis of medical images". In: *Clinical radiology* 59, pp. 1061–9. DOI: 10.1016/j.crad.2004.07.008.

Chapman, A. (2016). *Types of Drones: Multi-Rotor vs Fixed-Wing vs Single Rotor vs Hybrid VTOL.* URL: https://www.auav.com.au/articles/drone-types/ (visited on 03/12/2019).

Chen, K., V. Kvasnicka, P. Kanen, and S. Haykin (June 2001). "Multi-Valued and Universal Binary Neurons: Theory, Learning, and Applications [Book Review]". In: *Neural Networks, IEEE Transactions on* 12, pp. 647–647. DOI: 10.1109/TNN.2001.925572.

Dechter, R. (1986). "Learning While Searching in Constraint-Satisfaction-Problems". In: *AAAI*.

Dias, L. (2015). *Study and analysis of different camera calibration methods.* URL: http://repositorio.roca.utfpr.edu.br/jspui/handle/1/6454 (visited on 03/12/2020).

Ebadi, F. and M. Norouzi (2017). "Road Terrain detection and Classification algorithm based on the Color Feature extraction". In: *2017 Artificial Intelligence and Robotics (IRANOPEN)*, pp. 139–146. DOI: 10.1109/RIOS.2017.7956457.

FAA (2018). *Office of the Secretary of Transportation, Federal Aviation Administration, Department of Transportation. Unmanned Aircraft Systems.* URL: https://www.faa.gov/data_research/aviation/aerospace_forecasts/media/FY2018-38_FAA_Aerospace_Forecast.pdf (visited on 03/12/2019).

Farnebäck, G. (2003). "Two-Frame Motion Estimation Based on Polynomial Expansion". In: *Image Analysis*. Ed. by J. Bigun and T. Gustavsson. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 363–370. ISBN: 978-3-540-45103-7.

Fleming, J. (2014). *MQ-8B Fire Scout.* URL: https://web.archive.org/web/20140420235117/http://www.northropgrumman.com/Capabilities/FireScout/Documents/pageDocuments/MQ-8B_Fire_Scout_Data_Sheet.pdf (visited on 03/12/2019).

Fraczek, P., A. Mora, and T. Kryjak (2018). “Embedded Vision System for Automated Drone Landing Site Detection”. In: *Computer Vision and Graphics.* Ed. by L. J. Chmielewski, R. Kozera, A. Orłowski, K. Wojciechowski, A. M. Bruckstein, and N. Petkov. Cham: Springer International Publishing, pp. 397–409. ISBN: 978-3-030-00692-1.

Galloway, M. M. (1975). “Texture analysis using gray level run lengths”. In: *Computer Graphics and Image Processing* 4.2, pp. 172 –179. ISSN: 0146-664X. DOI: https://doi.org/10.1016/S0146-664X(75)80008-6. URL: http://www.sciencedirect.com/science/article/pii/S0146664X75800086.

Garra, B., B. Krasner, S. Horii, S. Ascher, S. Mun, and R. Zeman (Nov. 1993). “Improving the Distinction between Benign and Malignant Breast Lesions: The Value of Sonographic Texture Analysis”. In: *Ultrasonic imaging* 15, pp. 267–85. DOI: 10.1006/uimg.1993.1017.

Ghosh, H. and M. Sharma (Sept. 2015). “Histogram of gradient magnitudes: A rotation invariant texture-descriptor”. In: DOI: 10.1109/ICIP.2015.7351681.

Giusti, A., J. Guzzi, D. C. Cireşan, F. He, J. P. Rodríguez, F. Fontana, M. Faessler, C. Forster, J. Schmidhuber, G. D. Caro, D. Scaramuzza, and L. M. Gambardella (2016). “A Machine Learning Approach to Visual Perception of Forest Trails for Mobile Robots”. In: *IEEE Robotics and Automation Letters* 1.2, pp. 661–667. DOI: 10.1109/LRA.2015.2509024.

Glaser, A. (2018). *The Air Traffic Control System of Our Low-Altitude Future.* URL: https://slate.com/technology/2018/01/how-air-traffic-control-could-work-when-we-have-drones-and-flying-cars.html (visited on 03/12/2019).

Gomez, F. and J. Schmidhuber (Jan. 2005). “Co-evolving recurrent neurons learn deep memory POMDPs”. In: pp. 491–498. DOI: 10.1145/1068009.1068092.

González, D., J. Becker, E. Torres, J. Albistur, M. Escudero, R. Fuentes, H. Hinostroza, and F. Donoso (Jan. 2008). “Using LiDAR technology in forestry harvest planning”. In:

Gracia, C., R. S. Tavares, A. Mora, J. Fonseca, H. Oliveira, and L. O. Oliveira (2020). “FPGA-based Satellite Image Classification for Water Bodies Detection”. In: *IEEE International Young Engineers Forum on Electrical and Computer Engineering YEF-ECE*, pp. –.

Gruszczyński, W., W. Matwij, and P. Ćwiąkała (Apr. 2017). “Comparison of low-altitude UAV photogrammetry with terrestrial laser scanning as data-source methods for terrain covered in low vegetation”. In: *ISPRS Journal of Photogrammetry and Remote Sensing* 126, pp. 168–179. DOI: 10.1016/j.isprsjprs.2017.02.015.

Halterman, R. and M. Bruch (Apr. 2010). “Velodyne HDL-64E LIDAR for Unmanned Surface Vehicle Obstacle Detection”. In: *Proceedings of SPIE - The International Society for Optical Engineering*, p. 9. DOI: 10.1117/12.850611.

Haralick, R. M., K. Shanmugam, and I. Dinstein (1973). “Textural Features for Image Classification”. In: *IEEE Transactions on Systems, Man, and Cybernetics* SMC-3.6, pp. 610–621. ISSN: 0018-9472. DOI: 10.1109/TSMC.1973.4309314.

Haralick, R., K Shanmugam, and I. Dinstein (Jan. 1973). “Textural Features for Image Classification”. In: *IEEE Trans Syst Man Cybern* SMC-3, pp. 610–621.

Hastie, T., R. Tibshirani, J. Friedman, and J. Franklin (Nov. 2009). “The Elements of Statistical Learning: Data Mining, Inference, and Prediction”. In: *Math. Intell.* 27, pp. 83–85. DOI: 10.1007/BF02985802.

Heung, B., H. C. D. Ho, J. Zhang, A. Knudby, C. Bulmer, and M. Schmidt (Mar. 2016). “An overview and comparison of machine-learning techniques for classification purposes in digital soil mapping”. In: *Geoderma* 265, pp. 62–77. DOI: 10.1016/j.geoderma.2015.11.014.

Hofmann, T., J. Puzicha, and J. Buhmann (Sept. 1998). “Unsupervised texture segmentation in a deterministic annealing framework”. In: *Pattern Analysis and Machine Intelligence, IEEE Transactions on* 20, pp. 803 –818. DOI: 10.1109/34.709593.

Hong, H., L. Zheng, and S. Pan (2018). “Computation of Gray Level Co-Occurrence Matrix Based on CUDA and Optimization for Medical Computer Vision Application”. In: *IEEE Access* 6, pp. 67762–67770.

Horn, Z., L. Auret, J. McCoy, C. Aldrich, and B. Herbst (Dec. 2017). “Performance of Convolutional Neural Networks for Feature Extraction in Froth Flotation Sensing”. In: *IFAC-PapersOnLine* 50, pp. 13–18. DOI: 10.1016/j.ifacol.2017.12.003.

Hu, M.-K. (1962). “Visual pattern recognition by moment invariants”. In: *IRE Transactions on Information Theory* 8.2, pp. 179–187. ISSN: 0096-1000. DOI: 10.1109/TIT.1962.1057692.

Huang, d., C. Shan, M. Ardabilian, and L. Chen (Nov. 2011). “Local Binary Patterns and Its Application to Facial Image Analysis: A Survey”. In: *IEEE Transactions on Systems, Man, and Cybernetics, Part C* 41, pp. 765–781. DOI: 10.1109/TSMCC.2011.2118750.

Huang, N., Z Shen, S. Long, M. Wu, H. Shih, Q. Zheng, N. Yen, C.-C. Tung, and H. Liu (Mar. 1998). “The empirical mode decomposition and the Hilbert spectrum for nonlinear and non-stationary time series analysis”. In: *Proceedings of the Royal Society of London. Series A: Mathematical, Physical and Engineering Sciences* 454, pp. 903–995. DOI: 10.1098/rspa.1998.0193.

ITSB (2015). *ros_comm*. URL: http://wiki.ros.org/ros_comm (visited on 03/12/2019).

Jose, A., D. Merlin, N. Joseph, E. George, and A. Vadukoot (July 2014). “Performance study of edge detection operators”. In: pp. 7–11. DOI: 10.1109/EmbeddedSys.2014.6953040.

Joseph, L. (2015). *Mastering ROS for Robotics Programming*. Ed. by P. Publishing. Packt Publishing. ISBN: 9781783551798. URL: https://www.packtpub.com/hardware-and-creative/mastering-ros-robotics-programming.

Jung, B.-G., Y.-J. Cha, H.-H. Kim, S.-I. Jun, and J.-H. Cho (1995). “Dynamic Code inding for Scalable Operatin in Distributed Real-Time Systems”. In:

Khan, Y. N., P. Komma, and A. Zell (Nov. 2011). “High Resolution Visual Terrain Classification for Outdoor Robots”. In: pp. 1014–1021. DOI: 10.1109/ICCVW.2011.6130362.

Koch, R., S. May, and A. Nuchter (Mar. 2017). “DETECTION AND PURGING OF SPECULAR REFLECTIVE AND TRANSPARENT OBJECT INFLUENCES IN 3D RANGE MEASUREMENTS”. In:

Lacaze, A., K. Murphy, and M. DelGiorno (Jan. 2002). “Autonomous Mobility for the Demo III Experimental Unmanned Vehicles”. In:

LeCun, Y., Y. Bengio, and G. Hinton (May 2015). “Deep learning”. In: pp. 436–444. DOI: 10.1038/nature14539.

Lecun, Y., K. Kavukcuoglu, and C. Farabet (May 2010). “Convolutional Networks and Applications in Vision”. In: pp. 253–256. DOI: 10.1109/ISCAS.2010.5537907.

Li, B., K. Cheng, and Z. Yu (Oct. 2016). “Histogram of Oriented Gradient Based Gist Feature for Building Recognition”. In: *Computational Intelligence and Neuroscience* 2016, pp. 1–9. DOI: 10.1155/2016/6749325.

Liu, H., T. Hong, M. Herman, and R. Chellappa (Jan. 2006). “Accuracy vs. efficiency trade-offs in optical flow algorithms”. In: pp. 174–183. DOI: 10.1007/3-540-61123-1_137.

Lookingbill, A., J. Rogers, D. Lieb, J. Curry, and S. Thrun (Jan. 2007). “Reverse Optical Flow for Self-Supervised Adaptive Autonomous Robot Navigation”. In: *International Journal of Computer Vision* 74, pp. 287–302. DOI: 10.1007/s11263-006-0024-x.

Lucas, B. and T. Kanade (Apr. 1981). “An Iterative Image Registration Technique with an Application to Stereo Vision (IJCAI)”. In: vol. 81.

Ma, X., S. Hao, and Y. Cheng (2017). “Terrain classification of aerial image based on low-rank recovery and sparse representation”. In: *2017 20th International Conference on Information Fusion (Fusion)*, pp. 1–6. DOI: 10.23919/ICIF.2017.8009627.

Mallick, S. (2016). *Histogram of Oriented Gradient Opencv.* URL: https://www.learnopencv.com/histogram-of-oriented-gradients (visited on 06/12/2019).

Matos-Carvalho, J. P., F. Moutinho, A. B. Salvado, T. Carrasqueira, R. Campos-Rebelo, D. Pedro, L. M. Campos, J. M. Fonseca, and A. Mora (2019). “Static and Dynamic Algorithms for Terrain Classification in UAV Aerial Imagery”. In: *Remote Sensing* 11.21, p. 2501. ISSN: 2072-4292. DOI: 10.3390/rs11212501. URL: http://dx.doi.org/10.3390/rs11212501.

Matos-Carvalho, J. P., J. M. Fonseca, and A. D. Mora (2018). “UAV downwash dynamic texture features for terrain classification on autonomous navigation”. In: *Proceedings of the 2018 Federated Conference on Computer Science and Information Systems*. Ed. by M. Ganzha, L. Maciaszek, and M. Paprzycki. Vol. 15. Annals of Computer Science and Information Systems. IEEE, pp. 1079–1083. DOI: 10.15439/2018F185. URL: http://dx.doi.org/10.15439/2018F185.

Matthies, L., P Belluta, and M. McHenry (Sept. 2005). “Detecting water hazards for autonomous off-road navigation”. In: pp. 231–242. DOI: 10.1117/12.496942.

Mboga, N., C. Persello, J. R. Bergado, and A. Stein (2017). "Detection of informal settlements from VHR satellite images using convolutional neural networks". In: *2017 IEEE International Geoscience and Remote Sensing Symposium (IGARSS)*, pp. 5169–5172. DOI: 10.1109/IGARSS.2017.8128166.

Moore, T. and D. Stouch (Jan. 2016). "A Generalized Extended Kalman Filter Implementation for the Robot Operating System". In: 302, pp. 335–348. DOI: 10.1007/978-3-319-08338-4_25.

Mora, A., T. Santos, S. Łukasik, J. Silva, A. Falcão, J. Fonseca, and R. Ribeiro (Nov. 2017). "Land Cover Classification from Multispectral Data Using Computational Intelligence Tools: A Comparative Study". In: *Information* 8, p. 147. DOI: 10.3390/info8040147.

Nailon, W. (Mar. 2010). "Texture Analysis Methods for Medical Image Characterisation". In: ISBN: 978-953-307-071-1. DOI: 10.5772/8912.

Nanni, L., A. Lumini, and S. Brahnam (Mar. 2010). "Local binary pattern variants as texture descriptors for medical image analysis". In: *Artificial intelligence in medicine* 49, pp. 117–25. DOI: 10.1016/j.artmed.2010.02.006.

NASA (2018). *VTOL UAV With the Cruise Efficiency of a Conventional Fixed Wing UAV*. URL: https://technology.nasa.gov/patent/LAR-TOPS-241 (visited on 03/12/2019).

Ntouskos, V., I. Kalisperakis, and G. Karras (Jan. 2007). "Automatic calibration of digital cameras using planar chess-board patterns". In: *Optical 3-D Measurement Techniques VIII* 1.

Ojala, T., M. Pietikainen, and T. Maenpaa (2002). "Multiresolution gray-scale and rotation invariant texture classification with local binary patterns". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 24.7, pp. 971–987. ISSN: 0162-8828. DOI: 10.1109/TPAMI.2002.1017623.

Oppenheim, A. V., R. W. Schafer, and J. R. Buck (1999). "Discrete-time signal processing (2nd ed.)" In:

Otte, S., S. Laible, R. Hanten, and A. Zell (Jan. 2015). "Robust Visual Terrain Classification with Recurrent Neural Networks". In: DOI: 10.15496/publikation-11656.

Oyelade, J., O. Oladipupo, and I. Obagbuwa (Feb. 2010). "Application of k Means Clustering algorithm for prediction of Students Academic Performance". In: *International Journal of Computer Science and Information Security* 7.

P. Valavanis, K. (Jan. 2007). *Advances in Unmanned Aerial Vehicles: State of the Art and the Road to Autonomy*. ISBN: 978-1-4020-6113-4. DOI: 10.1007/978-1-4020-6114-1.

Pombeiro, R., R. Mendonça, P. Rodrigues, F. Marques, A. Lourenço, E. Pinto, P. Santana, and J. Barata (2015). "Water detection from downwash-induced optical flow for a multirotor UAV". In: *OCEANS 2015-MTS/IEEE Washington*. IEEE, pp. 1–6.

Pérez-Barnuevo, L. (Dec. 2017). "Automated recognition of drill core textures: A geometallurgical tool for mineral processing prediction". In: *Minerals Engineering* 118. DOI: 10.1016/j.mineng.2017.12.015.

R. Rosell, J., J. Llorens Calveras, R. Sanz, J. Arnó, M. Ribes-Dasi, J. Masip, A. Escolà, F. Camp, F. Solanelles, F. Gràcia, E. Gil, L. Val, S. Planas de Martí, and J. Palacín (Sept.

2009). “Obtaining the three-dimensional structure of tree orchards from remote 2D terrestrial LIDAR scanning”. In: *Agricultural and Forest Meteorology* 149, pp. 1505–1515. DOI: 10.1016/j.agrformet.2009.04.008.

Ramasamy, S., R. Sabatini, A. Gardi, and J. Liu (Aug. 2016). “LIDAR obstacle warning and avoidance system for unmanned aerial vehicle sense-and-avoid”. In: *Aerospace Science and Technology* 55, pp. 344–358. DOI: 10.1016/j.ast.2016.05.020.

Rankin, A. and L. Matthies (Nov. 2010). “Daytime water detection based on color variation”. In: pp. 215 –221. DOI: 10.1109/IROS.2010.5650402.

Rankin, A. L., L. H. Matthies, and P. Bellutta (2014). “Daytime water detection based on sky reflections”. In: *2011 IEEE International Conference on Robotics and Automation*, pp. 5329–5336.

Rato, R. E.C. T. (2012). “Formalização da tolerância à ausência de dados no processamento de sinais discretos”. PhD thesis. Faculdade de Ciências e Tecnologia Nova de Lisboa. URL: http://hdl.handle.net/10362/7971.

Raudies, F. (2013). *Optic flow*. URL: http://www.scholarpedia.org/article/Optic_flow (visited on 03/12/2020).

Rosebrock, A. (2015). *What are Local Binary Patterns?* URL: https://www.pyimagesearch.com/2015/12/07/local-binary-patterns-with-python-opencv (visited on 06/12/2019).

Salmon, J. (2010). “On Two Parameters for Denoising With Non-Local Means”. In: *IEEE Signal Processing Letters* 17.3, pp. 269–272. ISSN: 1558-2361. DOI: 10.1109/LSP.2009.2038954.

Salvado, A. B.d. T. (2018). “Aerial Semantic Mapping for Precision Agriculture using Multispectral Imagery”. MA thesis. Faculdade de Ciências e Tecnologia Universidade Nova de Lisboa. URL: http://hdl.handle.net/10362/59924.

Satpathy, A., X. Jiang, and H.-L. Eng (May 2014). “LBP-Based Edge-Texture Features for Object Recognition”. In: *Image Processing, IEEE Transactions on* 23, pp. 1953–1964. DOI: 10.1109/TIP.2014.2310123.

Schmidhuber, J. (2015). “Deep learning in neural networks: An overview”. In: *Neural Networks* 61, pp. 85 –117. ISSN: 0893-6080. DOI: https://doi.org/10.1016/j.neunet.2014.09.003. URL: http://www.sciencedirect.com/science/article/pii/S0893608014002135.

Shamos, M. I. and D. Hoey (1975). “Closest-point problems”. In: *16th Annual Symposium on Foundations of Computer Science (sfcs 1975)*, pp. 151–162.

Shen, K. and M. Kelly (2017). “Terrain Classification for Off-Road Driving CS-229 Final Report”. In:

Simple Introduction to Convolutional Neural Networks. https://towardsdatascience.com/simple-introduction-to-convolutional-neural-networks-cdf8d3077bac. Accessed: 12-06-2019.

Singh, M. K. (2019). *Difference between CPU and GPU*. URL: https://www.geeksforgeeks.org/difference-between-cpu-and-gpu/ (visited on 11/12/2019).

Sklar, D. L. (2015). *San Diego to Take Community Feedback on Future Drone Use.* URL: https://timesofsandiego.com/politics/2018/11/15/san-diego-to-take-community-feedback-on-future-drone-use/ (visited on 03/12/2019).

Sofman, B., J Bagnell, A. Stentz, and N. Vandapel (Jan. 2006). "Terrain Classification from Aerial Data to Support Ground Vehicle Navigation". In:

Spary, S. (2015). *Millennials' demand for instant gratification is shaping the future of retail.* URL: https://www.campaignlive.co.uk/article/millennials-demand-instant-gratification-shaping-future-retail/1331511 (visited on 03/12/2019).

Specht, D. (Feb. 1991). "A general regression neural network". In: *IEEE transactions on neural networks / a publication of the IEEE Neural Networks Council* 2, pp. 568–76. DOI: 10.1109/72.97934.

Tan, X. and B. Triggs (2010). "Enhanced Local Texture Feature Sets for Face Recognition Under Difficult Lighting Conditions". In: *IEEE Transactions on Image Processing* 19.6, pp. 1635–1650. ISSN: 1057-7149. DOI: 10.1109/TIP.2010.2042645.

Tong, X.-Y., Q. Lu, G.-S. Xia, and L. Zhang (2018). "Large-Scale Land Cover Classification in Gaofen-2 Satellite Imagery". In: *IGARSS 2018 - 2018 IEEE International Geoscience and Remote Sensing Symposium*, pp. 3599–3602.

Tüceryan, M. and A. K. Jain (Feb. 1990). "Texture Segmentation Using Voronoi Polygons". In: *IEEE Trans. Pattern Anal. Mach. Intell.* 12.2, pp. 211–216. ISSN: 0162-8828. DOI: 10.1109/34.44407. URL: http://dx.doi.org/10.1109/34.44407.

Tully Foote, M. P. (2014). *REP 103 – Standard Units of Measure and Coordinate Conventions (ROS.org). (n.d.).* URL: https://www.ros.org/reps/rep-0103.html (visited on 03/12/2020).

TullyFoote (2019). *tf2.* URL: http://wiki.ros.org/tf2 (visited on 03/12/2019).

Valdes, R. (2015). *How the Predator UAV Works.* URL: https://science.howstuffworks.com/predator (visited on 01/19/2018).

Vamvakas, A., I. Tsougos, N. Arikidis, E. Kapsalaki, K. Fountas, I. Fezoulidis, and L. Costaridou (May 2018). "Exploiting morphology and texture of 3D tumor models in DTI for differentiating glioblastoma multiforme from solitary metastasis". In: *Biomedical Signal Processing and Control* 43, pp. 159–173. DOI: 10.1016/j.bspc.2018.02.014.

Vooon (2018). *mavros.* URL: http://wiki.ros.org/mavros (visited on 03/12/2019).

Voronoi, G. (1908). "Nouvelles applications des paramètres continus à la théorie des formes quadratiques. Deuxième mémoire. Recherches sur les parallélloèdres primitifs." In: *Journal für die reine und angewandte Mathematik* 134, pp. 198–287. URL: http://eudml.org/doc/149291.

Wallace, L., A. Lucieer, Z. Malenovský, D. Turner, and P. Vopěnka (2016). "Assessment of Forest Structure Using Two UAV Techniques: A Comparison of Airborne Laser Scanning and Structure from Motion (SfM) Point Clouds". In: *Forests* 7.12, p. 62. ISSN: 1999-4907. DOI: 10.3390/f7030062. URL: http://dx.doi.org/10.3390/f7030062.

Wang, G., X.-Y. Chen, F.-L. Qiao, Z. Wi, and N. E. Huang (2010). "On Intrinsic Mode Function". In: *Advances in Adaptive Data Analysis*. ISSN: 1793-5369. DOI: 10.1142/s1793536910000549.

Wang, L., J. Peng, and W. Sun (2019). "Spatial–Spectral Squeeze-and-Excitation Residual Network for Hyperspectral Image Classification". In: *Remote Sensing* 11.7, p. 884. ISSN: 2072-4292. DOI: 10.3390/rs11070884. URL: http://dx.doi.org/10.3390/rs11070884.

Wang, L., D.-W. Sun, H. Pu, and Z. Zhu (May 2015). "Application of Hyperspectral Imaging to Discriminate the Variety of Maize Seeds". In: *Food Analytical Methods* 9. DOI: 10.1007/s12161-015-0160-4.

Woods, M., J. Guivant, and J. Katupitiya (Dec. 2013). "Terrain Classification using Depth Texture Features". In:

Yan, W. Y., A. Shaker, and N. El-Ashmawy (Mar. 2015). "Urban land cover classification using airborne LiDAR data: a review". In: *Remote Sensing of Environment* 158, pp. 295–310. DOI: 10.1016/j.rse.2014.11.001.

Yao, T., Z. Xiang, J. Liu, and D. Xu (Sept. 2007). "Multi-Feature Fusion Based Outdoor Water Hazards Detection". In: pp. 652 –656. ISBN: 978-1-4244-0828-3. DOI: 10.1109/ICMA.2007.4303620.

Zhao, Q., Y. Wang, and Y. Li (2016). "Voronoi tessellation-based regionalised segmentation for colour texture image". In: *IET Computer Vision* 10.7, pp. 613–622. ISSN: 1751-9640. DOI: 10.1049/iet-cvi.2015.0299.

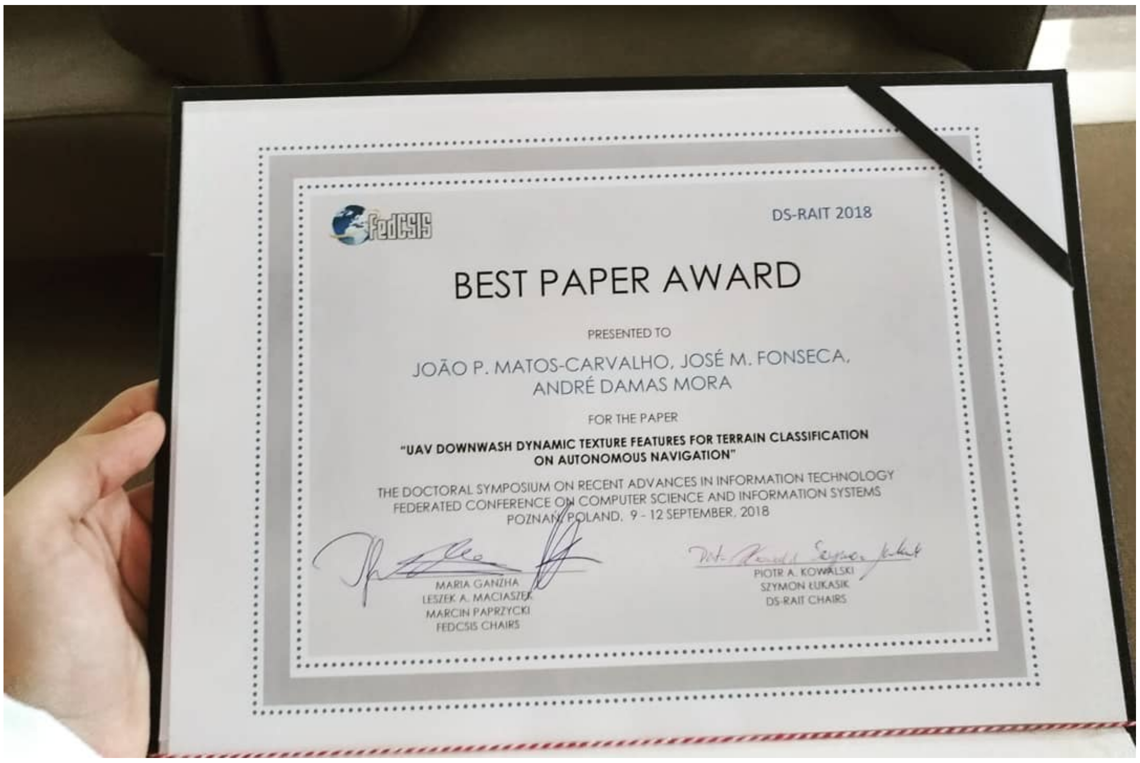
ZHENG, Y., S. Yang, X. Liu, J. WANG, T. Norton, J. CHEN, and Y. TAN (May 2018). "The computational fluid dynamic modeling of downwash flow field for a six-rotor UAV". In: *Frontiers of Agricultural Science and Engineering* 5, pp. 159–167. DOI: 10.15302/J-FASE-2018216.

Özuysal, M. (2018). "Ground texture classification with deep learning". In: *2018 26th Signal Processing and Communications Applications Conference (SIU)*, pp. 1–4.

Appendix A

Dissemination

Some of the concepts covered in this dissertation have been published in two International Journals and six International Conference Proceedings as shown in Table A.1 and Table A.2 respectively. In the case of one of the conference papers (first row in Table A.2) the author of this dissertation won the Best Paper Award, as shown in Figure A.1.



DS-RAIT 2018

BEST PAPER AWARD

PRESENTED TO

JOÃO P. MATOS-CARVALHO, JOSÉ M. FONSECA,
ANDRÉ DAMAS MORA

FOR THE PAPER

"UAV DOWNWASH DYNAMIC TEXTURE FEATURES FOR TERRAIN CLASSIFICATION ON AUTONOMOUS NAVIGATION"

THE DOCTORAL SYMPOSIUM ON RECENT ADVANCES IN INFORMATION TECHNOLOGY
FEDERATED CONFERENCE ON COMPUTER SCIENCE AND INFORMATION SYSTEMS
POZNAŃ, POLAND, 9 - 12 SEPTEMBER, 2018

[illegible]

MARIA GANZHA
LESZEK A. MACIASZEK
MARCIN PAPRZYCKI
FEDCSIS CHAIRS

[illegible]

PIOTR A. KOWALSKI
SZYMON ŁUKASIK
DS-RAIT CHAIRS

Figure A.1: Best Paper Award in (Matos-Carvalho et al., 2018).

Table A.1: Publications in International Journals.

	Citation
1	J.P. Matos-Carvalho, José Manuel Fonseca, and André Damas Mora (2019). Terrain Classification Using Static and Dynamic Texture Features by UAV Downwash Effect, Journal of Automation, Mobile Robotics and Intelligent Systems, 13(1), 84–93.
2	J.P. Matos-Carvalho, F. Moutinho, A.B. Salvado, T. Carrasqueira, R. Campos-Rebelo, D. Pedro, L.M. Campos, J.M. Fonseca, and A. Mora. Static and Dynamic Algorithms for Terrain Classification in UAV Aerial Imagery. Remote Sensing. 2019, 11, 2501.

Table A.2: Publications in International Conferences Proceedings.

	Citation
1	J. P. Matos-Carvalho, J. M. Fonseca, and A. Mora, "UAV Downwash Dynamic Texture Features for Terrain Classification on Autonomous Navigation," 2018 Federated Conference on Computer Science and Information Systems (FedCSIS), Poznan, 2018, pp. 1079-1083.
2	J. P. Matos-Carvalho, A. Mora, R. Rato, R. Mendonça, and J. M. Fonseca, 2019. UAV Downwash-Based Terrain Classification Using Wiener-Khinchin and EMD Filters. In Technological Innovation for Industry and Service Systems (pp. 83–90). Springer International Publishing.
3	J. P. Matos-Carvalho, D. Pedro, L.M. Campos, J.M. Fonseca, and A. Mora. 2020. Terrain Classification Using W-K Filter and 3D Navigation with Static Collision Avoidance. In Intelligent Systems and Applications (pp. 1122–1137). Springer International Publishing.
4	I. Kim, J. P. Matos-Carvalho, I. Viksnin, L. M. Campos, J. M. Fonseca, A. Mora, and S. Chuprov 2019. Use of Particle Swarm Optimization in Terrain Classification based on UAV Downwash. In 2019 IEEE Congress on Evolutionary Computation (CEC) (pp. 604-610).
5	A. B. Salvado, R. Mendonça, A. Lourenço, F. Marques, J. P. Matos-Carvalho, L. Miguel Campos, and J. Barata 2019. Semantic Navigation Mapping from Aerial Multispectral Imagery. In 2019 IEEE 28th International Symposium on Industrial Electronics (ISIE) (pp. 1192-1197).
6	P. Prates, R. Mendonça, A. Lourenço, F. Marques, J. P. Matos-Carvalho and, J. Barata, 2018. Vision-based UAV detection and tracking using motion signatures. In undefined (pp. 482-487).

Appendix B
Supporting Concepts

This chapter introduces some key aspects which serve as a basis for this dissertation. These key aspects are the differences between different UAV models (Section B.1), as well as the one chosen for this thesis; the concept of downwash in Section B.2, what it is and how is it produced by an aerial vehicle; computer vision is introduced in Section B.3, where its sub-domains are briefly described; the framework used in this dissertation (Section B.4); the sensors that will be used (Section 3.1.1); and a brief description of the system specifications and architecture (Section 3.1.3). This chapter introduces some important concepts which will help the reader to better understand the content of this dissertation.

B.1 UAV Models

What started as a platform for leisure is today becoming a serious industry. Vehicles, known as UAVs are not only growing in number, but also in the variety of potential applications. Some of the possible applications enumerated in (P. Valavanis, 2007), are: pollution and forest-fire monitoring; delivery of retail products; border patrol; aerial mapping and surveillance; traffic monitoring; precision agriculture; and search and rescue operations).

Several UAV models are shown in Figure B.1. In Figure B.1 a), a military drone from General Atomics was developed to be used in dangerous war zones for reconnaissance and combat, controlled far from these areas (Valdes, 2015). A solution from Amazon is intended to deliver packages, with shorter delivery times (Figure B.1 b)). Next, Figure B.1 c) shows a drone carrying a camera with the objective of filming a nature documentary (notice that, from Figure 1.1, photography is the main type of use for drones registered with the FAA). Furthermore, Figure B.1 d) presents an agriculture-related solution. Finally, Figures B.1 e) and B.1 f), also present military solutions, with the former being a single-rotor helicopter used by the US Armed Forces (for surveillance, reconnaissance, fire support, among other uses) and the latter, a fixed-wing hybrid VTOL with 10 engines (named GL-10, a prototype from NASA) in hover mode. These solutions differ, not only in terms of their applications, but also in the type of UAV in use. These types will be discussed below.

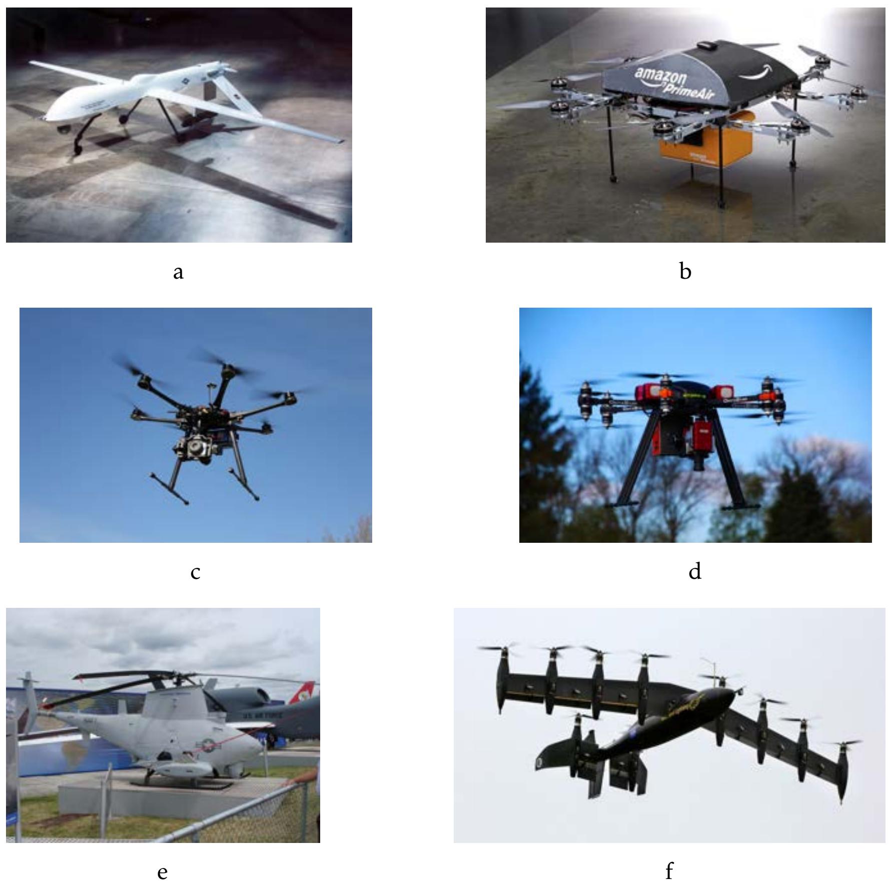


Figure B.1: Several examples of UAV models: (a) General Atomics RQ-1A Predator. (b) Amazon Prime Air, (Spary, 2015). (c) DJI S800, carrying a camera for filming a nature documentary under the Helicam Project (photo by Alexander Glinz), (Sklar, 2015). (d) Agricultural research conducted with a drone and hyperspectral camera in April 2016 in Belgium, (Cargyrak, 2016). (e) MQ-8B Fire Scout Helicopter UAV (Single-rotor) (Fleming, 2014). (f) Photograph of Hybrid-VTOL UAV prototype. Image credit: NASA (NASA, 2018).

Now, It has been noted that the use of UAVs can range rom military applications to recreation, with clear advantages that include: not exposing the aircraft operator; entering environments that might be dangerous to humans; performing strenuous or repetitive tasks, tele-operated or autonomously (AUVSI, 2015). According to (Chapman, 2016), UAVs can be divided into four main categories:

- **Fixed-Wing (Figure B.2):** Cannot lift vertically and cannot hold their position in the air because of their "wing". As with normal airplanes, this type of UAV is much more energy-efficient than multi-roto UAVs.

Figure B.2: Fixed-Wing UAV.

Being able to use a gasoline engine in these types of UVA will increase their autonomy to more than 16 hours and will allow them to cover much longer distances. The downsides are that they cannot hover, so it is not possible to process aerial mappings, and because of their size the launching phase is a lot trickier since they need a runaway or a catapult launcher to get them into the air. The landing phase is also more complicated, as they will require a runway, a parachute or a net to recover them safely. Another downside is their higher cost of manufacture and a steeper learning curve when it comes to piloting;

- **Single-Rotor Helicopter (Figure B.3):** Looks very similar in design and structure to actual helicopters since it has one main rotor plus a smaller one on the tail of the drone to control direction. The benefits are that they are much more efficient than multi-rotor helicopter UAVs and can also be powered by a gasoline engine for even longer endurance.

Figure B.3: Single-Rotor Helicopter UAV.

The higher efficiency is due to a rule of aerodynamics that states the larger the rotor blade and the slower it spins, the more efficient it is. In terms of complexity, operational and manufacturing costs, the single-rotor is superior when compared with the other alternatives. However, the big drawback is the danger posed by their large spinning blades: the long sharp blade of a single-rotor can be fatal and also cause major property losses if due caution is not exercised;

- **Multi-Rotor (Figure B.4):** These are the most common type of UAVs, and also the cheapest, which are used by not only professionals but also amateurs, mainly for aerial photography and aerial video surveillance. This is due to the fact that they enable the greatest control over position and framing.

Figure B.4: Multi-Rotor UAV: HEIFU.

The big drawback of this type of UAV is their major limitations when it comes to endurance and speed. Not only this, but they also require a lot of energy just to fight gravity and stabilize themselves, so they drain their batteries relatively quickly during a flight (20 to 30 mins). They can be further classified based on the number of rotors present on the drone: are Tricopter (3 rotors), Quadcopters (4 rotors), Hexacopters (6 rotors) and Octocopters (8 rotors).

- **Fixed-Wing Hybrid Vertical Take-Off and Landing (VTOL) (Figure B.5):** These UAVs combine the benefits of fixed-wing models (longer flying time) with that of rotor based models (hovering).

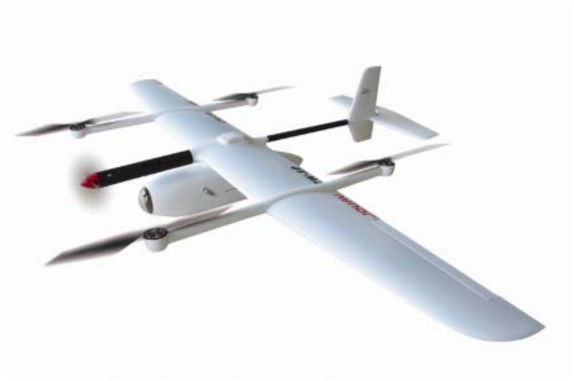


Figure B.5: Fixed-Wing Hybrid VTOLUAV.

Vertical lift is used to lift the drone up into the air from the ground. Gyros and accelerometers work in automated mode (autopilot concept) to keep the drone stabilized in the air. Remote based (or even programmed) manual control is used to guide the drone on the desired course.

Without further ado, Table B.1 presents a summarized comparison of the four UAV types. From the table, it is possible to infer that the multi-rotor solution can be a suitable choice for detecting water, given that this solution can easily approach any terrain type in order to retrieve detailed measurements, and it is considered to be low cost. The quadcopter shown in Figure B.7 is the solution used in this thesis, possessing all of the manoeuvrability enunciated previously. Moreover, it carries a gimbal with an RGB camera, which will help with the terrain classification tasks.

B.2 Downwash Effect

The concept of flying consists in using rotors to project air down beneath the aircraft, making it fly against the force of gravity. This is applicable to large aircraft, helicopters and multi-rotor unmanned aerial vehicles. The blades spin around, forcing the air over their curved upper surface and projecting it downwards towards the ground, producing and upward force called lift.

The downwash effect is a phenomenon produced, at low altitudes, by the aircraft rotors, due to the gradient in air pressure between the upper and lower rotor blade surfaces. When flying, the air flows in the direction in which it is forced by the blades, as seen in Figure B.6. When the aircraft gets closer to the surface, the air forced beneath the aircraft cannot continue its movement because of the ground, and is diverted, traveling in different directions. This effect is exploited by both small and large airplanes, and increases the lift generated by wings when the aircraft is closer to the ground, as seen in Figure B.7.

Normally, some of the high-pressure air beneath the wing wraps around the low-pressure upper surface of the wing. This destroys some of the lift the wing generates, and is an accepted inefficiency of wing design. However, when an airplane is benefiting from ground effect, the ground interferes with this process, improving the efficiency of the wing. An airplane can fly with ground effect in situations where it could not fly in normal air.

For helicopters such effect is also a very important for hovering. The downwash effect varies greatly depending on the surface beneath the aircraft. Concrete, water and vegetation (tall or short) behave very differently. In water environments, this reaction translates into the movement of the water outwardly in all directions from the position of the aircraft, creating a radial pattern on the water's surface (Figure B.7).

Table B.1: Comparison of UAV Categories.

	Advantages	**Drawbacks**	**Example(s)**
Fixed-Wing	Covers wide areas, for long periods of time; Its wings allow it to control and maintain altitude; the only function of the motors is to move the vehicle forward (more efficient).	Low manoeuvrability; considerably expensive; taking-off/landing is difficult and may require mechanisms such as catapult, runway or hand launching.	Figure B.1 a).
Single-Rotor Helicopter	Can take off and land vertically; can carry a high payload (e.g. precise LiDAR sensors); has high autonomy (time of flight).	Can be significantly expensive; and, if used in agriculture environments, its blades can easily damage crops.	Figure B.1 e).
Multi-Rotor	Cheapest solution (compared to the other three types); high manoeuvrability and positioning control.	Does not allow high speeds (compared to the previous solutions); and has poor autonomy given the great effort required by the motors to hover. This makes this solution unsuitable for large scale coverage [19].	Figures B.1 b), B.1 c), B.1 d) and B.7.
Fixed-Wing Hybrid	Combines the previous approaches: incorporates both hover and forward flight modes.	Rather recent solution; higher price due to the extra technology requirements.	Figure B.1 f); currently being researched by Amazon (to deliver retail products).

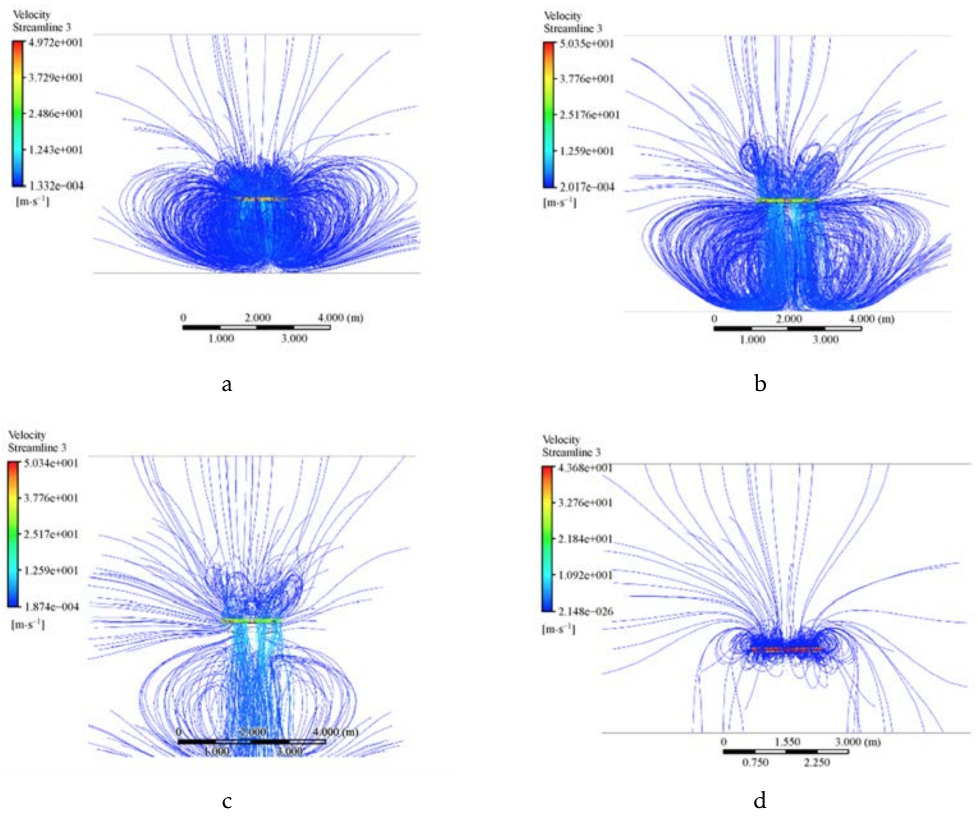


Figure B.6: UAV downwash flow fields at different hovering heights: a) 2m; b) 3m; c) 5m; and the relative boundless height (d) (ZHENG et al., 2018).

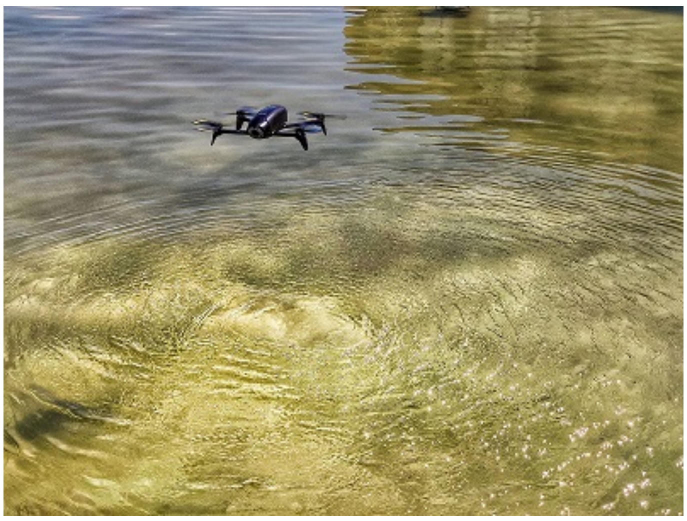


Figure B.7: Downwash Effect: Using Bebop2 to classify water terrain in Lisbon, Portugal.

B.3 Computer Vision

Acquiring, processing, analysing and understanding images is the main purpose of the computer vision field. In general it gathers 2D or 3D high-dimensional data from the real world, producing symbolic or numerical information in order to generate decisions. It embraces models constructed with the aid of physics, statistics and geometry. Computer vision exceeds human vision as it takes advantages of the high processing capacity of computer systems. Human vision has limited memory, limited to the visible spectrum, and has a propensity to generate illusion. Computer vision englobes many sub-domains such as:

- **3-D Reconstruction:** This technique can make use of stereo vision, which is the process of understanding depth from camera images by comparing two or more views of the same scene. The output generated is a 3-D point cloud where each point corresponds to a pixel in one of the images. This technique is frequently used for modelling objects and scene reconstruction;

- **Event Detection:** Consists of detecting predictable events using a set of images, e.g. for visual surveillance or people counting;

- **Object Recognition:** The main focus is to determine a specific object by means of a given image or images that have been previously learned. This technique can either use 2-D or 3-D images;

- **Motion Estimation:** This is a topic of computer vision that studies movements in image sequences. This technique can be used in such tasks as egomotion, which involves determining the camera rotation and translation from an image sequence (e.g. using the optical flow algorithm).

Some systems are intended to solve a specific issue as a stand-alone application, others have a much broader design, containing more sub-systems for controlling external devices, information databases, man-machine interfaces, etc. However the core general functions for a computer vision system tend to follow a similar path, as can be seen in Figure B.8. Where:

- **Image Acquisition:** Digital image produced by an image sensor. It can be static or a sequence of images (dynamic). The information gathered can be the pixel values in one or more spectral bands or other types of information, such as depth or electromagnetic waves;

- **Pre-processing:** This phase assures that the analysed images satisfy the requirements of the method, such as resolution and color grading. Other examples of pre-processing consists of noise reduction, contrast enhancement and re-sampling;

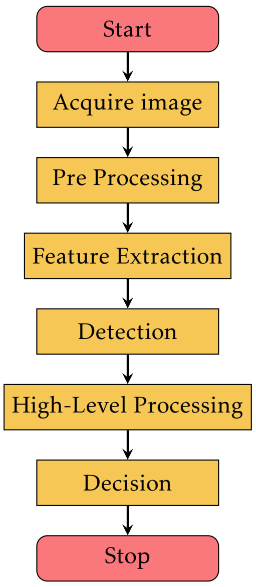


Figure B.8: Example of a generic computer vision algorithm.

- **Feature extraction:** A crucial step in computer vision is the extraction of features from the image data. Features can be lines, edges, ridges or interesting features such as corners or points;

- **Detection:** This is where the specific points of interest are relevant enough to be selected for further processing;

- **High-level processing:** Consisting of a set image regions (or a full image). This is where this data will be processed using the methods of the application in order to generate a decision;

- **Decision:** The final decision for the application.

B.4 Robot Operating System

The Robot Operating System (ROS) is a "robot application development platform that provides various features such as message passing, distributed computing, code reusing, (...)" (Joseph, 2015), incorporating software and hardware applied to robotics applications. The ROS also provides support for a wide range of programming languages; the possibility of visualizing data (e.g. the content of topics, nodes, packages, coordinate systems graphs and sensor data); and the possibility to write and execute code in a modular way, increasing robustness and also contributing to the standardization of this framework.

Some related concepts will be introduced, as they will be required to understand some of the algorithms that will be discussed in this dissertation.

B.4.1 ROS Packages

The basic concept that the reader needs to be familiarized with is the ROS package. This is where nodes' source and header files, executable scripts, launch files, among other items are stored and organized.

B.4.2 ROS Graph Layer

One of the packages installed from source is ros_comm[1] contains the ROS middleware/-communication packages, known as the ROS "Graph" Layer. This package is directly associated with implementations and tools for the concepts shown in Figure B.9. These concepts can be found in any book, and in the ROS wide online documentation, although some of these will be briefly introduced here given their relevance to the content of this dissertation.

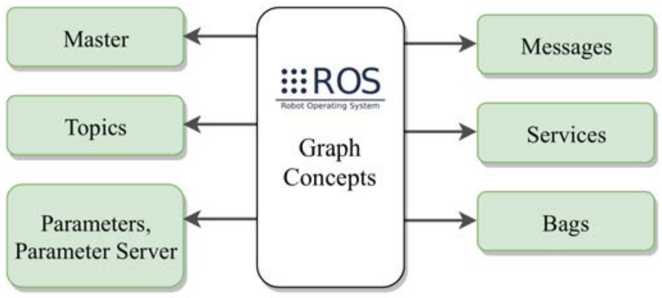


Figure B.9: ROS Graph Concepts (ITSB, 2015).

The ROS Master[2] is what makes it possible for different nodes to find and communicate with each other. A distributed approach will be adopted in this thesis. The process will be initialized on one machine, and nodes will be able to communicate with it from a remote machine (Joseph, 2015).

Given that ROS is a distributed computing framework, it allows network connections and information exchange to perform different tasks. Messages allow a publish/subscribe pattern of Topics[3] (the message named bus - its identifier), whereas services[4] provide a request/response behavior. Therefore, while ROS topics are unidirectional, with ROS services, one node is the server, from which a client may request a service and, after completion of a procedure, send a reply (Joseph, 2015). This is done with nodes[5], explained in Section B.4.3.1, which can be seen as the system's processes, and it may run on different machines.

[1] http://wiki.ros.org/ros_comm
[2] http://wiki.ros.org/roscore
[3] http://wiki.ros.org/Topics
[4] http://wiki.ros.org/Services
[5] http://wiki.ros.org/Nodes

Finally, bags[6] will be used extensively in this thesis, as they allow the information being transmitted via different topics to be saved. The parameter server allows different parameters to be accessible from a node. These can be loaded from a launch file or from inside a node and can be declared in a YAML file.

B.4.3 Nodes vs. Nodelets

Two important concepts are the aforementioned ROS Nodes (Joseph, 2015), and Nodelets, which are where all of the computation will take place. Figure B.10 summarizes the difference between these two methods.

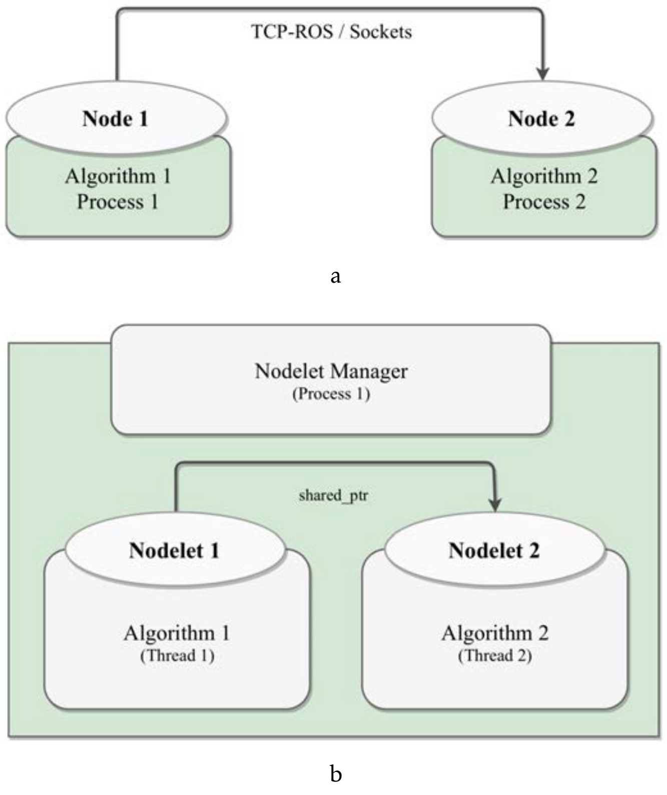


Figure B.10: a) Nodes vs. b) Nodelets.

B.4.3.1 Nodes

The nodes are presented in Figure B.10 a). The way in which they function has already been explained. Each node can directly communicate with another via standard TCP/IP sockets and the master functions in the same way as a Domain Name System (DNS) server. Each node can therefore subscribe to a topic which leads to the establishment of a connection with the publisher. This communication between nodes is one of the most important aspects in ROS.

[6]http://wiki.ros.org/rosbag

The problem with this approach is that, in the case of the TCP/IP protocol, when large amounts of data are being transferred (such as Point Clouds or a large Octomap) then, the whole process of packing, sending and unpacking the message takes a lot of time (Aaron, 2016), which becomes unbearable if tasks such as textures extraction and obstacle detection are included.

B.4.3.2 Nodelets

As an alternative, nodelets[7], shown in Figure B.10 b), provide zero copy communication between processes (which are called nodelets), since the communication process sends boost shared pointers. Nevertheless, two drawbacks exist, which are that nodelets are usually more unstable than nodes; and given that a pointer is used to avoid copying data, data can only be shared between processes on the same machine, which may be a problem if expensive processing requirements are required (Aaron, 2016).

B.4.4 Coordinate Frame Management

The question "How does the ROS framework manage the relationships between coordinate frames?" remains unanswered. This is important given that a robot has to be described by coordinate frames that may change over time and has to be localized in reference to world frame(s).

There are three packages related to this topic that are worth mentioning: `tf2`[8]; `jointStatePublisher`[9]; and `robotStatePublisher`[10]. The first maintains the relationships between frames organized in a tree-like structure and allows to access transforms between any two coordinate frames at any time (TullyFoote, 2019). The robot description can be written on a specific Unified Robot Description File (URDF) file type, where each joint can be classified, for example, as revolute (meaning that it can rotate around an axis). The `jointStatePublisher` finds all of the non-fixed joints from a robot description and publishes a message with those joints defined, and `robotStatePublisher` uses the robot description from the URDF file and calculates the forward kinematics, publishing the results via tf. A graphic interface can be used to change the joint states.

B.5 Other Packages

Other packages that are being used in this project are mentioned in Table B.2, along with a brief description. Although, it is important to mention that a large number of packages have not been mentioned, mostly because they are very commonly used in ROS, the ones presented in the table are considered to be important for this project and had to be installed following installation of the ROS.

[7] `http://wiki.ros.org/nodelet`
[8] `http://wiki.ros.org/tf2`
[9] `http://wiki.ros.org/joint_state_publisher`
[10] `http://wiki.ros.org/robot_state_publisher`

Table B.2: Installed Packages.

Package Name	**Description**
nmea_navsat_driver; nmea_msgs	Provides an ROS interface for Global Positioning System (GPS) devices that output compatible NMEA sentences and publish a topic of sensor_msgs/NavSatFix with the global coordinates of the device. It will also be used when collecting data for tests (Vooon, 2018).
mavros	Provides a communication driver for various autopilots with MAVLink communication protocol (Moore and Stouch, 2016). It will be used to connect with the ArduPilot board.
robot_localization	Will be used to estimate the global frame based on the IMU and GPS. It converts the sensor_msgs/NavSatFix to IMU and uses an EKF to compute the global frame estimation.