NOVA SCHOOL OF SCIENCE & TECHNOLOGY

DEPARTAMENTO DE INFORMÁTICA

OCTÁVIO NUNO JARDIM FREIRE

Licenciatura em Engenharia Informática

AUTONOMIZAÇÃO DE SOFTWARE OPEN SOURCE (WEBODM) PARA MODELAÇÃO E MAPEAMENTO DE BARRAGENS COM DRONES

MESTRADO EM ENGENHARIA INFORMÁTICA

Universidade NOVA de Lisboa
Julho, 2022

NOVA SCHOOL OF SCIENCE & TECHNOLOGY

DEPARTAMENTO DE INFORMÁTICA

AUTONOMIZAÇÃO DE SOFTWARE OPEN SOURCE (WEBODM) PARA MODELAÇÃO E MAPEAMENTO DE BARRAGENS COM DRONES

OCTÁVIO NUNO JARDIM FREIRE

Licenciatura em Engenharia Informática

Orientador: Nuno Miguel Cavalheiro Marques
Professor Auxiliar, Universidade NOVA de Lisboa

Coorientadores: João Manuel Marcelino Mateus da Silva
Investigador Principal, Laboratório Nacional de Engenharia Civil

João Miguel Gomes Pires Manso
Investigador Auxiliar, Laboratório Nacional de Engenharia Civil

Daniel Teixeira Leite
Estudante de Doutoramento, Universidade NOVA de Lisboa

MESTRADO EM ENGENHARIA INFORMÁTICA

Universidade NOVA de Lisboa
Julho, 2022

Para a minha Mãe e Madrinha Elisa,

Agradecimentos

Gostaria de agradecer ao meu orientador, o Prof. Nuno Marques por me ter ter dado a oportunidade de trabalhar nesta dissertação, juntamente com os engenheiros do LNEC, Eng. João Marcelino, Eng. João Manso e Eng. Daniel Leite, pois sem eles esta dissertação não existia. Agradeço também à Faculdade de Ciências e Tecnologia da Universidade Nova de Lisboa por todo o conhecimento transmitido ao longo destes 5 anos.

Gostaria de agradecer aos meus pais e à minha família, que sempre acreditou em mim. Gostaria de agradecer a todos os meus amigos que me apoiaram durante a escrita da dissertação, seja física ou psicologicamente.

Um agradecimento especial ao meu primo Valério, que me ajudou a fazer testes com o seu drone para a validação de resultados intermédios durante a escrita da dissertação. Gostaria também de agradecer ao Eng. Mário Pereira do LREC, que me ajudou com a medição de coordenadas GPS enquanto eu estava na Ilha da Madeira.

Por fim, gostaria de agradecer à minha madrinha Elisa, que acompanhou pessoalmente, o início do meu percurso académico e que agora me acompanha lá em cima.

Eu consegui Madrinha.

"As a well-spent day brings happy sleep, so life well used brings happy death." (Leonardo da Vinci)

Resumo

Os sistemas de monitorização da saúde estrutural das barragens são cruciais para identificar comportamentos anómalos e desta forma minimizar ou eliminar os seus efeitos. A caraterização das barragens, no que diz respeito aos princípios de engenharia, é viabilizada com o uso de equipamentos avançados de monitorização como por exemplo piezómetros e medidores de caudal, que descrevem o desempenho mecânico das mesmas, evitando desastres naturais e mitigando riscos socioeconómicos.

A colocação de um número limitado de pontos de monitorização é, geralmente, insuficiente para barragens de grande porte, uma vez que muitas das áreas das barragens são inacessíveis. A solução encontrada para esta dificuldade passa pelo uso de métodos em larga escala e sem contacto. Consequentemente, as tecnologias baseadas na visão têm sido consideradas como uma abordagem eficiente para a monitorização da saúde estrutural das barragens. Sensores baseados em visão computacional, combinados com o constante aperfeiçoamento na resolução digital e capacidade de computação, surgiram como uma ferramenta promissora para a aferição remota de estruturas.

Outra aplicação das imagens é a criação de modelos tridimensionais (3D) das estruturas. Esta metodologia consiste numa renderização baseada em imagens, capaz de identificar danos e realizar medições óticas sem contacto, que podem ser aplicadas na prevenção de desastres. Para obter o número necessário de imagens do objeto, em tempo útil e a baixo custo, é necessária uma plataforma flexível que possa ser controlada remotamente e suportar diversos tipos de configurações.

Devido aos recentes avanços no uso e disponibilidade de plataformas de Veículos Aéreos Não Tripulados (VANT) e o desenvolvimento de software de processamento de imagem de fácil manuseamento, a fotogrametria baseada em VANT está a ser adotada cada vez mais para produzir topografia de alta resolução com o intuito de estudar alterações nas superfícies. Os VANT equipados com câmeras são adequados para o levantamento da superfície do terreno, devido à geração de topografia de alta resolução para documentar as caraterísticas topográficas em tempo real, pelo que permitem a avaliação da integridade da infraestrutura e a deteção de danos de grandes estruturas.

Tendo em conta que um dos fatores com maior influência na precisão de um modelo

tridimensional é o uso de Ground Control Points (GCPs), o seu número e distribuição pelo terreno são importantíssimos. Torna-se necessário colocar na superfície da barragem um número específico de GCPs, de forma a obter um modelo georreferenciado de elevada precisão e à escala.

Nesta dissertação, propõe-se desenvolver um processo automático de identificação de GCPs usando fotogrametria de VANT, para deteção de danos em barragens e monitorização de emergências. Este processo acabará por aumentar a eficiência, bem como a precisão, na criação de ortofotos e no desenvolvimento de modelos tridimensionais (3D) dessas estruturas.

Palavras-chave: Open source, Modelos 3D, Mapeamento Aéreo, Veículos Aéreos Não Tripulados (Drones), Barragens, Pontos de Controlo, Levantamento Topográfico, Open Drone Map, Marcadores ArUco

Abstract

Dam health monitoring systems are crucial for identifying anomalous behaviours and eliminating or minimising their effects.

Advanced monitoring equipment like piezometers and flow meters allows the characterization of the dams, regarding the principles of engineering, and describes their mechanical performance, thus avoiding natural disasters and mitigating socio-economic risk.

The placement of a limited number of monitoring points is usually insufficient for large-scale dams, since many areas of the dams are inaccessible. Large-scale and non-contact methods applied for dam health monitoring presented themselves as a solution. Vision-based technologies have been considered as an efficient approach for structural health monitoring. Camera and computer vision-based sensors combined with the improvements in camera resolution and computation ability, have emerged as a promising tool for the non-contact remote measurement of structural responses.

Another application of images is the creation of three-dimensional (3D) models of the structures. This methodology consists of an image-based rendering capable of identifying damage and performing non-contact, optical-based measurements that can be applied for disaster prevention. In order to obtain the necessary number of photographs of the object, in a timely and inexpensive manner, a flexible platform is required that should be controlled remotely and support several types of payloads.

Due to the recent advances in the use and availability of Unmanned Aerial Vehicle (UAV) platforms and the development of easy-to-operate image processing software, UAV-based photogrammetry is being adopted increasingly to produce high-resolution topography for studying surface processes. UAVs equipped with cameras are suitable for surveying the terrain surface due to the generation of high-resolution topography to document the real-time topographic features. The infrastructure integrity assessment and damage detection of large structures are often performed by Unmanned Aerial Vehicles (UAVs).

A specific number of GCPs, which are usually measured by traditional surveying instruments, need to be deployed on the object to generate the UAV-based model with

the geolocation and scale. One of the factors with the greatest influence on the accuracy of a UAV is the number and distribution of the GCPs.

In this thesis, a semi-automated process of Ground Control Point (GCP) identification using UAV photogrammetry is proposed for dam damage detection and emergency monitoring. This process will ultimately increase efficiency, as well as accuracy, in the creation of orthophotos, and in the development of three-dimensional (3D) models of these structures.

Keywords: Open Source, 3D Models, Aerial Mapping, Unmanned Aerial Vehicles (Drones), Dams, Control Points, Topographic Information Gathering, Open Drone Map, ArUco Markers

Índice

Índice de Figuras

Índice de Tabelas

Índice de Listagens

Glossário

EXIF A maioria dos equipamentos de captura de imagem, guardam meta-informação associada às imagens como por exemplo a data e hora, o número pixeis que a foto tem ou as coordenadas GPS do local onde a foto foi tirada. Essa informação é geralmente guardada com o formato Exchangeable Image File Format (EXIF), como foi o caso das fotos retiradas pelos drones utilizados nesta dissertação [52]. 19

Família de AprilTags As famílias de Apriltags representam diferentes modelos de marcadores. Tomando como exemplo a família ***16h5***, o primeiro valor antes da letra ***h***, representa o número de bits que o marcador tem. No caso da família ***16h5***, significa que o marcador tem 16 bits, ou seja, é constituído por 4x4 blocos pretos ou branco que equivalem a 16 bits. O segundo número depois da letra ***h*** representa a distância de Hamming, que quanto maior for, melhor será correção de erro aquando da descodificação do marcador, pela biblioteca que fizer a sua descodificação. No entanto, quanto maior for esta distância menor será a quantidade de id's que aquela família de marcadores consegue gerar [31]. 13

GeoTIFF O formato GeoTIFF foi inicialmente desenvolvido no início dos anos 90, com o intuito de adicionar meta-informação geográfica ao formato de imagem TIFF [10]. 9

Siglas

CLI Command-line interface 26, 34

DI Departamento de Informática 1

FCT Faculdade de Ciência e Tecnologia da Universidade NOVA de Lisboa 1

GCP Ground Control Point 9, 13, 2, 3, 9, 10, 11, 13, 15, 16, 17, 18, 19, 20, 21, 22, 24, 26, 29, 33, 39

GCPs Ground Control Points 7, 8, 9, 3, 9, 11, 13, 18, 19, 35, 37, 39, 43, 44, 45

GeoTIFF Geographic Tagged Image File Format 17

GPS Global Positioning System 15, 16, 17, 19, 21, 22, 24, 26, 28, 30, 31, 35

GSD Ground Sample Distance 23, 24, 29, 40

GUI Graphical User Interface 14

IGN French National Geographic Institute 8

LNEC Laboratório Nacional de Engenharia Civil 1, 2, 9, 10, 14, 26, 35, 36, 44, 45

MDT Modelos Digitais de Terreno 9

ODM Open Drone Map 3, 5, 11, 13, 14, 15, 19, 22, 41, 45

SfM Structure from Motion 6

UAV Unmanned Aerial Vehicle 8, 9

UAVs Unmanned Aerial Vehicles 8

VANT Veículos Aéreos Não Tripulados 6, 7

1

Introdução

1.1 Problema e Motivação

Este trabalho decorre no âmbito de uma colaboração entre o Departamento de Informática (DI) da Faculdade de Ciência e Tecnologia da Universidade NOVA de Lisboa (FCT) e o Laboratório Nacional de Engenharia Civil (LNEC) com o objetivo de tornar o levantamento topográfico de estruturas de engenharia, nomeadamente barragens, mais prático, preciso e rápido. Atualmente, entre outros, o LNEC usa o *software* de processamento de imagens aéreas WebODM, que sendo *open source* permite um melhor controlo face às alternativas comerciais, e permite a inclusão de desenvolvimentos como aquele que agora se propõe.

Neste momento Portugal possui cerca de 260 grandes barragens, sendo estas de betão, de aterro ou mistas, as quais desenvolvem um papel muito importante no abastecimento de água potável, bem como na produção de energia verde, contribuindo para a desativação das Centrais de Carvão e redução da emissão de CO2 [3]. Por outro lado, acidentes com barragens, apesar de raros, quando ocorrem causam imensos prejuízos económicos e ambientais. Acidentes como o assentamento excessivo do corpo de uma barragem de aterro ou a existência de zonas húmidas na mesma são eventos podem ser evitados e mitigados quando detetados precocemente. Normalmente, a observação destas barragens é realizada através da leitura de instrumentos como piezómetros, marcas superficiais, medidores de caudal, entre outros. Porém, estes instrumentos apesar de serem eficientes, possuem a limitação de observar pontualmente partes da estrutura, não sendo viável aplicá-los em toda a extensão da barragem. Salienta-se ainda, que uma parcela importante destas barragens não possui instrumentos instalados, aumentando a importância de uma outra componente do controlo de segurança, que é a realização de inspeções visuais.

O aparecimento de novas tecnologias, tais como drones, vieram possibilitar uma visão global das barragens. Permitem ainda efetuar uma recolha de dados com uma ótima relação custo-benefício [27], e ajudam na deteção precoce de anomalias através da aferição de deslocamentos, da deteção de áreas húmidas, ou da presença de fissuras.

Com a utilização de drones é possível fazer uma recolha de dados a baixo custo. Dados

estes que podem ser posteriormente processados e utilizados como um levantamento topográfico. Para aumentar a precisão da georreferenciação das imagens, posicionam-se pontos de controlo em locais estratégicos no solo, também designados por Ground Control Points (GCP), cujas coordenadas são conhecidas com elevada exatidão. No pós-processamento das imagens, ao identificar os alvos nas fotografias tiradas pelo drone, é possível corrigir a posição dos objetos no modelo tridimensional, com base nos GCP identificados nas fotografias.

1.2 Objetivos

O objetivo desta dissertação de mestrado é desenvolver ferramentas que automatizem o processo de identificação dos GCP nas imagens aéreas, obtidas com drones, para posterior processamento com o *software* WebODM. Pretende-se desenvolver um programa com uma interface apelativa e *user friendly*, de forma a facilitar a introdução dos dados recolhidos pelos drones, sendo esta utilizada na construção de um ficheiro de texto. Este ficheiro irá conter a informação dos GCPs, que pode ser aplicado no WebODM, para a obtenção de um ortomosaico/modelo tridimensional mais preciso. Esta ferramenta irá contribuir para aumento da eficiência e precisão dos levantamentos. Um levantamento topográfico consiste numa serie de métricas retiradas a um terreno com a ajuda de uma estação total ou drones com o intuito de mapear de forma rigorosa o terreno em questão [21].

1.3 Abordagem

Inicialmente, serão utilizados dados provenientes de diferentes conjuntos de imagens, presentes em relatórios de campanhas de informação conexa, efetuados pelo LNEC. Para complementar a base de dados atual, serão efetuados novos levantamentos. Após a nova recolha proceder-se-á da seguinte forma:

1. Implementação de um algoritmo de identificação automático dos GCP, nas imagens recolhidas pelos drones, sendo o resultado final compatível com o sistema WebODM;

2. Validação e *tuning* do algoritmo acima implementado, através de testes com datasets[1] recolhidos em campanhas realizadas em parceria com o LNEC durante o desenvolvimento desta dissertação.

3. Disponibilização do mecanismo criado em formato *open source* na plataforma GitHub.

[1] Dado que o termo em inglês *dataset* é utilizado vulgarmente no meio, este irá ser utilizado ao longo desta dissertação.

1.4 Contribuições Esperadas

As contribuições esperadas com esta dissertação são:

- Automatização do processo de selecção dos GCPs nas imagens, diminuindo consideravelmente o tempo total de processamento (desde a importação das imagens até ao geração de um ortomosaico/modelo 3D).
- Diminuição do erro na identificação dos GCPs.
- Proposta de alteração do atual processo de marcação dos GCPs ao projeto Open Drone Map (ODM), deixando de utilizar a interface atual e passando a utilizar o algoritmo desenvolvido nesta dissertação (GCP Finder), descrito no capítulo 3. Melhorando desta forma a eficiência do *software open source* WebODM.
- Construção de vários datasets de forma a contribuir para futura investigação na identificação automática de GCPs.

1.5 Organização do Documento

Esta dissertação divide-se em 6 capítulos:

Introdução: O primeiro capítulo descreve o problema e a motivação que originaram a dissertação, menciona os objetivos propostos e quais as abordagens adotadas.

Estado da Arte: Este capítulo descreve conceitos relevantes que serão mencionados ao longo desta dissertação. São descritas as atuais aplicações e os *softwares* existentes para o processamento de imagens aéreas, são enumeradas as vantagens e desvantagens de cada solução e é analisado o uso de GCP no mapeamento de território com drones. São também mencionados projetos que se consideram relevantes e que têm objetivos em comum com este trabalho.

Arquitetura: Neste capítulo é apresentada a arquitetura do algoritmo GCP Finder, desenvolvida nesta dissertação, que consiste em separar imagens que contêm um GCP, das imagens que não contêm. Desta forma, pretende-se otimizar o processo, analisando apenas as imagens que contêm um GCP.

Implementação: Neste capítulo é explicado como é que se utiliza o algoritmo GCP Finder do ponto de vista de um utilizador, que interfaces é que são disponibilizadas e como é que é feita a sua instalação.

Avaliações e Testes: Descreve os testes e validações efetuados ao algoritmo que foi desenvolvido no âmbito desta dissertação. Apresenta os datasets recolhidos nos levantamentos topográficos efetuados durante o desenvolvimento da dissertação e avalia a precisão da identificação feita pelo algoritmo, assim como o erro associado aos drones utilizados.

Conclusões e Trabalho Futuro: Neste último capítulo é feita uma síntese do trabalho efetuado até à data e são sugeridas melhorias ao algoritmo GCP Finder, para que numa próxima revisão este seja mais preciso e eficiente.

2

Estado da Arte

Neste capítulo aborda-se o projeto ODM (Open Drone Map), um programa *open source* utilizado para processamento de imagens aéreas e que será usado nesta dissertação. Descrevem-se as melhorias propostas, tendo em conta o que é oferecido na versão atual do ODM. São também mencionados outros programas, noutros regimes de licenciamento, nomeadamente na metodologia usada para melhorar a precisão dos modelos. Por fim, são abordados vários projetos, já existentes, com objetivos comuns a esta dissertação.

2.1 Conceitos Relevantes

Ao longo desta dissertação serão mencionados conceitos importantes na qual é importante o leitor estar familiarizado com seu significado. Uma nuvem de pontos, por exemplo, representa uma coleção de esferas num espaço 3D. Cada ponto constitui o contacto com uma superfície ou edifício na realidade [24]. A nuvem de pontos da Barragem da Lapa ou da Barragem de Sambade, podem ser vistas nas figuras 2.1 e 5.2.

Um modelo digital de elevação é uma representação em 2D, constituída por uma malha de pontos que descreve a altura de uma certa área geográfica que inclui apenas o terreno em si [52].

Um modelo digital de superfície é também uma representação em 2D de uma malha de pontos que descreve a altura de uma área geográfica mas que também inclui árvores, edifícios e outras estruturas que estejam acima da superfície [52].

Um ortomosaico consiste num conjunto de ortofotos colocadas umas ao lado das outras, normalmente tiradas por veículos aéreos. As ortofotos são imagens georreferenciadas e que foram previamente processadas de forma a não apresentar a distorção das lentes da câmara. Um conjunto de ortofotos forma um ortomosaico [12]. O exemplo de um ortomosaico pode ser encontrado na figura 2.7.

Uma estação total é um instrumento utilizado na construção civil para medir a distância a um objeto, saber a sua latitude, longitude e altitude e calcular o ângulo que este faz com o solo e com outros objetos. As estações totais são usadas principalmente por topógrafos e engenheiros civis que precisam de medições precisas de um terreno ou edifício

[48].

A técnica Structure from Motion (SfM) é capaz de através de várias imagens do mesmo objeto, tiradas de ângulos diferentes estimar a posição de objetos num espaço a 3 dimensões, criando desta forma uma estrutura designada de nuvem de pontos [45]. Desta forma, é criado um objeto 3D a partir das imagens 2D, como se pode ver na figura 2.1.

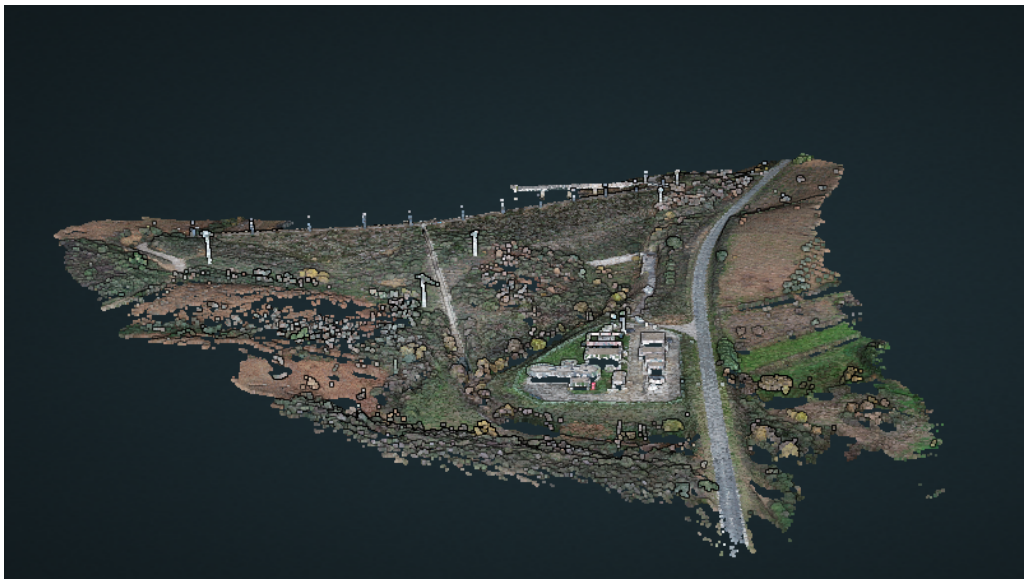

Figura 2.1: Nuvem com 1 630 000 pontos, gerada pelo WebODM a partir das imagens do levamento topográfico da barragem de Sambade, efetuado no dia 13 de Dezembro de 2019 pelo LNEC.

2.2 Software de Processamento de Imagens Aéreas

Uma imagem aérea é caracterizada principalmente por ser tirada a partir de um veículo aéreo, seja ele um avião, um drone, um satélite, entre outros [9]. No entanto não é só isso que caracteriza uma imagem aérea. Nem todas as imagens tiradas por um veículo aéreo servem para construir um Ortomosaico ou um Modelo Digital de Terreno.

As imagens têm que ter certas propriedades de forma a ser possível obter um bom resultado. O ângulo em que a imagem é capturada, a altitude e a percentagem de sobreposição das imagens faz toda a diferença entre um bom e um mau resultado final. No caso do Pix4DMapper, mencionado na secção 2.2.4 deste capítulo, é geralmente recomendado utilizar um padrão de voo em rede e ter pelo menos 75% de sobreposição frontal das imagens [36]. Na prática utiliza-se a direção em que o drone está a voar e garante-se pelo menos 60% de sobreposição lateral, ou seja, entre os "corredores" do voo como se pode ver na figura 2.2 [36].

No entanto a altitude e a percentagem de sobreposição das imagens irá variar conforme o tipo de terreno que estamos a mapear. Um descampado e uma floresta não deverão seguir a mesma abordagem [36].

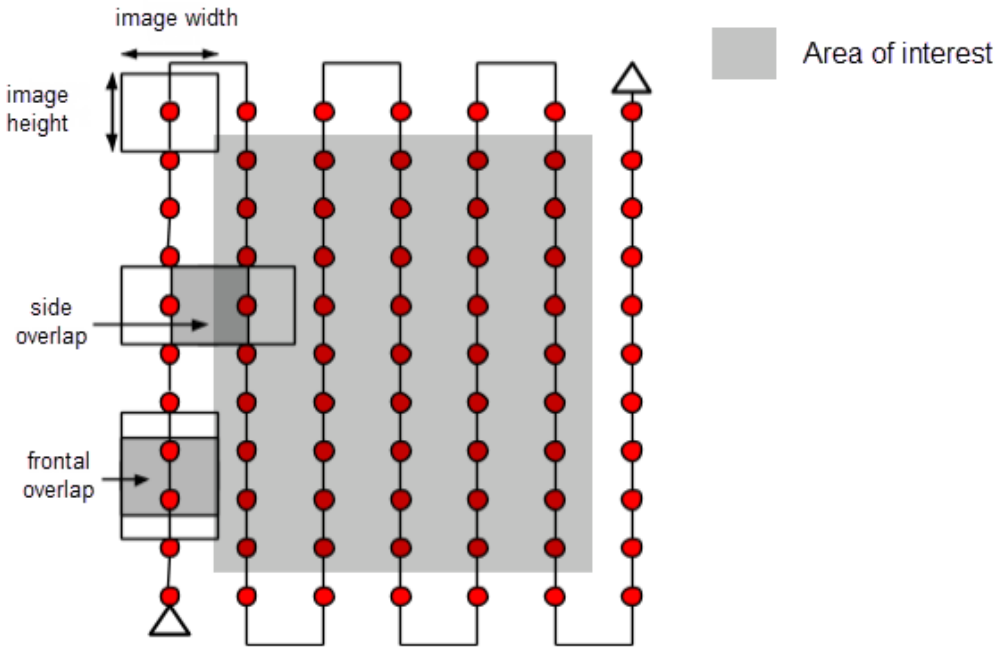

Figura 2.2: Padrão de voo em rede recomendado pelo Pix4DMapper para a captura de imagens aéreas [43].

Sendo o WebODM um programa gratuito e em regime de código aberto, era de esperar que existissem programas comerciais que tivessem os mesmos objetivos. Na tabela 2.1 é possível comparar os programas mais usados atualmente e perceber onde é estes se diferenciam. Nas próximas sub-secções analisamos em detalhe os diversos projetos.

Tabela 2.1: Software para processamento de imagens aéreas.

Software/Features	*Open source*	**Preço**	*Cross-Platform*	**Limite de imagens para input**	**Detecção Automática de GCP**	**Ortomosaico**
WebODM/ODM [32, 52]	Sim	Grátis	Sim	Sem limite	Não	Sim
Agisoft Metashape Standard [4, 5]	Não	179$	Sim	Sem limite[1]	Não [2]	Sim
Drone Mapper (Rapid/ Remote Expert) [17, 18, 50]	Não [3]	159$/Ano ou 999$	Windows	250/10000	Sim	Sim
DroneDeploy (Pro/Business) [11, 13, 15, 16]	Não	99$/Mês ou 299$/Mês	Sim	1000/3000	Sim [4]	Sim
Pix4DMapper & Pix4dCloud Advanced [37, 39, 40, 41]	Não	332$/Mês	Sim	Sem limite [5]	Sim	Sim

[1] O número máximo de fotos que o Agisoft Matashape consegue de processar depende da memória RAM disponível e dos parâmetros/opções usados no projeto.

[2] O agisoft não utiliza um processo totalmente automático para encontrar os GCP no entanto, chega lá perto. Basta seleccionarmos um GCP para uma foto para depois o *software* calcular com base naquela marcação, as projecções dos marcadores nas outras fotos e encontrar os restantes ground control points.

[3] O projeto em si não é *open source* mas utiliza 2 projetos que são (geoBits e o *engine* micmac) [50, 20]. O projeto geoBits consiste num sistema de marcadores com informação embutida em cada um deles, sendo estes marcadores capazes de serem reconhecidos até um máximo de 180m de altitude com elevada precisão [50]. O projeto do *engine* MicMac consiste num fork do projeto original do French National Geographic Institute (IGN), mas é patrocinado pela equipa DroneMapper.

[4] Apenas disponível para clientes do plano Enterprise e para clientes do plano Business com o custo de 49$ por mapa.

[5] O Pix4dCloud Advanced está limitado a 4000 imagens por projeto no entanto o Pix4dMapper não tem limite de imagens por projeto pois este depende das especificações do mesmo. Entre as especificações mais importantes salienta-se o tamanho das imagens, a resolução dos píxeis, o nível de detalhe e conteúdo, as opções de processamento selecionadas, o número de *features* e a capacidade de processamento do hardware.

2.2.1 Agisoft Metashape

O Agisoft Metashape, assim como outros programas de processamento de imagens aéreas [32, 40, 18, 14], é capaz gerar dados em 3 dimensões a partir de um conjunto de imagens 2D, que depois podem ser usados em sistemas que suportem informação geográfica. A geração de ortomosaicos e de Modelos Digitais de Terreno (MDT) também é suportada [5]. O Agisoft permite o uso de ground control points através de marcadores/*targets* codificados em forma circular ou marcadores com um padrão de xadrez (mas com excepções) [6]. O processo de selecção dos GCP é feita de duas formas, manualmente ou de forma semiautomática:

Na primeira é necessário identificar individualmente cada GCP, na outra basta identificar o GCP para uma foto e depois o *software* aplica aquela projecção automaticamente às outras fotos, tornando assim o processo de identificação dos GCP muito mais rápido. É também possível utilizar marcadores não codificados como é o caso do marcador em padrão de xadrez, no entanto a sua deteção é apenas feita depois da etapa de alinhamento das câmeras, o que quer dizer que estes não irão servir para ajudar a estimar a posição das câmeras, como é o caso dos marcadores codificados [5].

2.2.2 DroneMapper

O DroneMapper é um *software* comercial de processamento de imagens aéreas que possui neste momento 2 versões, a RAPID e a REMOTE EXPERT.

A versão RAPID pode ser descarregada gratuitamente a partir do website mas apenas irá produzir um *preview* do ortomosaico final caso esta não seja uma versão registada. Quando obtida a licença, esta versão é capaz de gerar modelos digitais de terreno e ortomosaicos com formato GeoTIFF a partir de um máximo de 250 imagens [19]. A versão REMOTE EXPERT não possui as limitações que a RAPID tem e suporta até 10000 imagens como input. É a versão mais popular dentro destas duas no entanto é também a mais cara [17].

2.2.3 DroneDeploy

O DroneDeploy possui uma aplicação onde é possível fazer planeamento e voo's autónomos, análise dos dados recolhidos pelo drone, geração de mapas 2D e 3D, MDT, ortomosaicos e utilizar GCPs para melhorar a precisão dos mapas a serem gerados. Além disto foi com o DroneDeploy que o LNEC efetuou todos os levantamentos a barragens utilizando um drone. Com o DroneDeploy é possível dectetar automaticamente os ground control points. Funcionalidade esta que já existe noutros *softwares* comerciais do mesmo género [40, 18], tornando assim este processo mais rápido e menos propício a erros [13].

2.2.4 Pix4Dmapper

O Pix4Dmapper é o principal *software* de processamento de imagens aéreas usado para gerar nuvens de pontos, modelos digitais de superfície, modelos 3D e ortomosaicos. Tal como outros *softwares* comerciais de processamento de imagens aéreas, o Pix4D oferece a possibilidade de utilizar ground control points de forma a obter um mapeamento mais preciso [40]. O Pix4Dmapper também oferece uma aplicação gratuita disponível para dispositivos móveis, o PIX4Dcapture, com o propósito de planear voos com o drone, dando a possibilidade ao utilizador de selecionar o tipo de voo desejado, seja ele circular, em rede ou ajustado ao formato da área a mapear, ajustar o *overlap* das fotografias e ajustar a velocidade com que o drone faz o voo [38]. Tanto esta solução de planeamento de voo como a oferecida pelo *software* DroneDeploy são excelentes. No entanto para os voos efetuados nesta dissertação foi utilizada a solução oferecida pelo DroneDeploy devido à vasta experiência que LNEC tem com o uso deste *software*.

2.3 Trabalho relacionado

2.3.1 Find-GCP project

Foi recentemente divulgado um projeto da Universidade de Tecnologia e Economia de Budapeste que pretende automatizar a identificação dos GCP no projeto Open Drone Map utilizando para o efeito ArUco markers [23]. O projeto é também open source, tem o código fonte disponível no GitHub e o possui um *paper* publicado no Baltic Journal of Modern Computing a explicar o seu funcionamento [46, 47]. Este projeto tem algumas semelhanças com o algoritmo desenvolvido GCP Finder no âmbito desta dissertação que serão explicadas mais à frente na secção 6.1 do capítulo 6.

Os ArUco Markers foram desenvolvidos por um conjunto de docentes da Universidade de Cordoba em Espanha, para resolver o problema de má estimação da pose da câmera em aplicações de realidade aumentada. Foi apresentado um novo algoritmo [23] para a geração de dicionários com marcadores configuráveis, um método para detetar automaticamente os marcadores e corrigir possível erro que seja gerado. Foi ainda apresentada uma solução para o problema de obstrução que existe em aplicações de realidade aumentada (quando algum objeto é colocado entre a câmera e a mesa com os marcadores, por exemplo). Os ArUco markers estão disponíveis no modulo "aruco" da biblioteca OpenCV [23]. Na figura 2.3 encontra-se o exemplo de um ArUco marker.

2.3.2 Checkerboard library

Para fazer o reconhecimento dos GCP com o padrão de xadrez (2 quadrados branco e 2 pretos), foram feitas várias tentativas de reconhecimento com a biblioteca Checkerboard[6], versão 0.2.4 e de modo geral os resultados obtidos foram inconsistentes, com uma elevada

[6] https://pypi.org/project/checkerboard

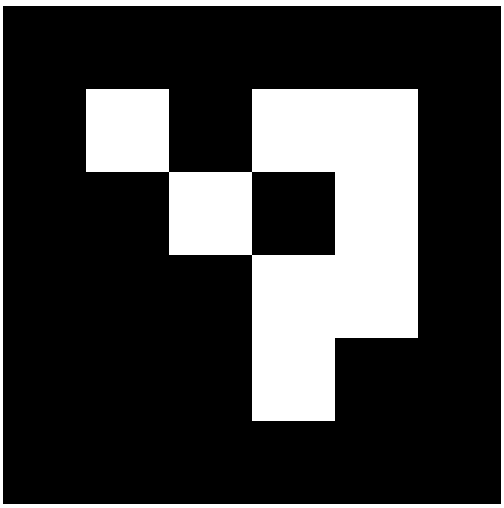

Figura 2.3: Exemplo de um ArUco marker 4x4 [28].

taxa de falsos positivos. Em todas as fotos que foram usadas existam muitas pedras perto do GCP (o que é uma situação corrente nas barragens de aterro) e nas fotos em que o GCP aparecia primeiro do que as pedras (de cima para baixo) o *software* identificava com muita precisão o seu centro, no entanto caso fosse o contrario, em que aparecessem as pedras primeiro, os resultados não eram animadores, como se pode verificar nas figuras 2.4 e 2.5. Nestas figuras é possível observar a identificação feita pela biblioteca Checkerboard, representada pelo ponto a vermelho visível em ambas as figuras.

Na figura 2.4 a identificação foi correta, no entanto na figura 2.5 é possível concluir que a identificação feita errou por completo o centro do alvo. Mas é possível perceber o porquê. A zona que foi selecionada contêm uma parte escura e outra clara à semelhança do marcador, o que pode ter levado o *software* a pensar que este seria o centro de um marcador em xadrez.

Uma má identificação do centro do GCP iria levar a falsos positivos reduzindo imenso a precisão da georreferenciação do modelo.

No entanto esta biblioteca não foi criada para este propósito, daí ser compreensível obter-se estes resultados. Uma forma de melhorar os resultados poderia passar por realizar um pré-processamento à imagem. Porém tal abordagem não foi seguida neste trabalho.

Também não foi possível utilizar a função ***findChessboardCorners*** da biblioteca OpenCV pois esta recebe como parâmetro um padrão de xadrez com um mínimo de 4x4 quadrados, enquanto que o *GCP* usado nos testes tem apenas 2x2 [51].

2.3.3 Langeroo's odm_gcp_apriltag

O projeto odm_gcp_apriltag do *developer* Langeroo consiste em identificar GCPs com AprilTags e gerar o ficheiro necessário de input ao ODM. AprilTags por sua vez são um tipo de marcadores visuais usados para realidade aumentada, calibração de câmeras ou

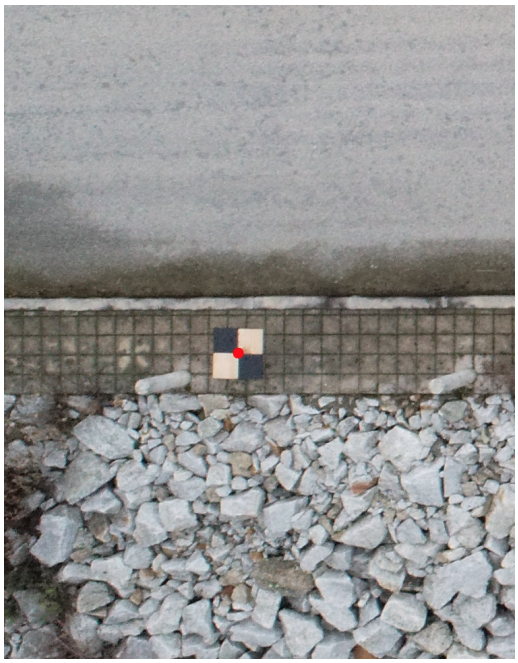

Figura 2.4: Identificação correta do centro do GCP.

Figura 2.5: Identificação incorreta do centro do GCP no canto superior esquerdo.

robótica mas que no entanto também podem ser utilizados como pontos de controlo. A vantagem destes marcadores é que podem ser facilmente gerados por *software* e impressos. É possível obter dos marcadores a sua posição na imagem, a orientação e um id. Isto pois podem haver vários marcadores na mesma imagem [33]. O projeto odm_gcp_apriltag usa os AprilTags como ground control points, e através de bibliotecas próprias é possível obter a localização dos marcadores em cada imagem. Para gerar o ficheiro de identificação de coordenadas GCP é obrigatório incluir um ficheiro com as coordenadas dos GCP existentes e com o seguinte formato:

```
tagfamily_id  geo_x  geo_y  geo_z
```

Em que o primeiro campo é o identificador do GCP em questão, pois normalmente existem vários, o segundo é a latitude do GCP, depois a longitude e finalmente a altitude. O ficheiro de *output* gerado por este projeto, é o que servirá de input ao ODM. O mesmo será gerado automaticamente e terá os seguintes campos: latitude, longitude, altitude, localização horizontal do GCP na imagem (x píxeis), localização vertical (y píxeis) e o nome da imagem em questão.

O projeto odm_gcp_apriltag utiliza os marcadores AprilTags, que poderiam ter sido utilizados no projeto desenvolvido nesta dissertação. No entanto para um dado tamanho físico do marcador, quanto mais bits tiver esse marcador, menos visível será esse marcador a uma grande altitude, daí ser recomendado utilizar o menor número de bits possíveis, que neste caso é 16. Um bit representa um bloco preto ou branco no marcador. O marcador ilustrado na figura 2.6 é constituído por 36 bits, ou seja, 6x6 bits.

Em contrapartida a única ***Família de AprilTags*** de 16 bits é a família ***16h5***, que apenas consegue gerar 30 ids únicos. Ao contrario dos ArUco Markers, que conseguem gerar até 1000 ids únicos com uma família de 16 bits. Por este motivo foram utilizados os ArUco Markers como GCPs para o desenvolvimento dessa dissertação e foi feita a integração da biblioteca que faz o reconhecimento deste tipo de alvo no algoritmo GCP Finder.

O exemplo de um Apriltag encontra-se na figura 2.6 e o exemplo de um marcador ArUco na figura 2.3.

2.4 Open Drone Map

O ODM já foi uma simples aplicação de linha de comandos usada para analisar e processar imagens aéreas, mas actualmente é um ecossistema de aplicações. Entre elas, realça-se o próprio *engine*, que faz o processamento. O *engine* ODM, em si, é um conjunto de ferramentas *open source* usado para processar e transformar imagens aéreas tiradas por drone, em mapas georreferenciados, nuvens de pontos, modelos digitais de elevação e modelos 3D com textura [32].

O projeto Open Drone Map disponibiliza ainda o *engine* MICMAC como alternativa [52]. O MICMAC é também um *engine open source* de processamento e transformação

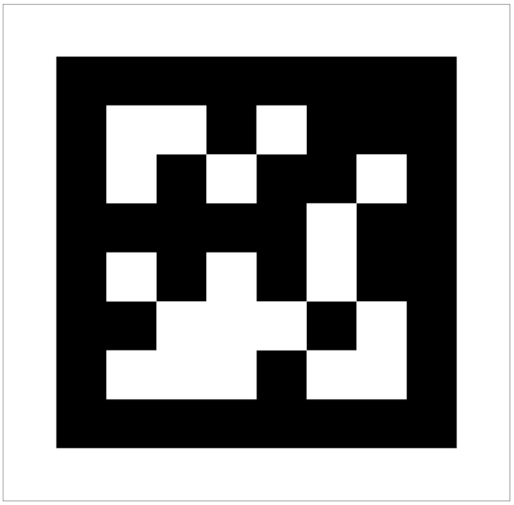

Figura 2.6: Exemplo de um AprilTag marker [22].

de imagens aéreas, mas face ao ODM possui a vantagem de oferecer uma Graphical User Interface (GUI) e um manual de utilização detalhado e gratuito [35]. Este *engine* não foi explorado com mais detalhe, pois embora seja similar ao ODM, não é utilizado correntemente pelo LNEC e consequentemente não existe experiência na sua utilização.

2.4.1 Arquitetura

O projeto OpenDroneMap é constituído por várias componentes, sendo três delas o *core* de todo o projeto (ODM, NodeODM e WebODM). Começando pelo *engine* ODM em si, criado em 2014, é uma ferramenta usada para processar imagens aéreas, nomeadamente provenientes de drones [32].

"In a nutshell, it's a program that takes images as input and produces a variety of georeferenced assets as output, such as maps and 3D models." [32]

Um outro pilar importante do projeto é a API REST(*NodeODM*), que foi criada com o intuito de facilitar a interação com o *engine* ODM, sendo também utilizada por programadores para criar outras aplicações com recurso ao *engine* ODM. Por fim, existe ainda o WebODM, que como o nome indica, é uma versão Web do ODM. Nesta aplicação já existe uma maior abstração e facilidade na utilização do *software*, face ao ODM, sendo possível criar uma conta e autenticar-se perante o sistema, visualizar mapas em 2 e 3 dimensões e paralelizar trabalho usando nós de processamento. É possível utilizar em paralelo vários servidores, estando estes a executar uma cópia do NodeODM ou do NodeMICMAC em cada um deles [32].

2.4.2 Pipeline

O *engine* ODM é constituído, maioritariamente, por 6 fases, criando assim um pipeline de execução:

1. **Structure from Motion** é uma técnica utilizada para estimar objetos num espaço tridimensional a partir de imagens em duas dimensões. Esta técnica está descrita com maior detalhe na secção 2.1, e é utilizada pelo projeto ODM tirando partido do *software open source* OpenSfM da empresa Mapillary. Nesta fase é também gerado um ficheiro de transformação, que faz uso da informação Global Positioning System (GPS) contida nas imagens ou, caso exista, do ficheiro gcp_list.txt, que será utilizado na etapa 5 para a conversão entre o sistema de coordenadas local do ODM e um sistema de coordenadas real [34, 52, 54]. O ficheiro gcp_list.txt, que será referido no resto do documento como ficheiro de identificação de coordenadas GCP, e está descrito com mais detalhe na subsecção 2.5.2, contém um conjunto de dados que estabelece a relação entre a localização dos GCP no mundo e a localização desses pontos nas imagens tiradas pelo drone.

2. **Multi View Stereo (Point Cloud Densification)** recebe a nuvem de pontos, criada pela primeira etapa, e densifica-a com algoritmos apropriados, criando assim uma nuvem de pontos visível ao olho humano [34, 54].

3. **Meshing** transforma a nuvem de pontos criada, numa superfície monocromática. Para tal, é usado o método de triangulação de Poisson disponível em Point Cloud Library [34, 54].

4. **Texturing** acrescenta à superfície criada, na etapa anterior, textura, sendo esta baseada nas imagens 2D iniciais colocadas lado a lado, seleccionando automaticamente as melhores imagens para cada parte da superfície. O resultado desta etapa é uma malha de pontos tridimensional, já com cor, na qual se conseguem identificar claramente carros, árvores, edifícios, entre outros [34].

5. **Georeferencing** é o processo de transformar um sistema de coordenadas interno (local a um *software*) num sistema de coordenadas geográficas conhecidas [26, 52]. Todas as etapas anteriores são efetuadas com recurso a um sistema de coordenadas local e não à informação GPS disponível nas imagens ou no ficheiro de identificação de coordenadas GCP. No entanto, caso estes dados existam, são usados durante a 1º etapa do pipeline para alinhamento e minimização do erro, entre todas as localizações GPS das imagens e todas as posições estimadas das câmaras [52].

6. **Orthophoto Processing and Generation** é a última etapa do pipeline, que consiste em carregar a malha de pontos tridimensional gerada na etapa 4, já texturizada, juntamente com um modelo ortográfico criado. Este resultado é guardado numa imagem com a resolução adequada sendo, por fim, georreferenciada. O ortomosaico

Figura 2.7: Ortomosaico obtido através do processamento das imagens no WebODM, aquando do levantamento topográfico efetuado no Parque das Nações no dia 17/12/2021.

é visualizado com recurso a uma vista de topo, da malha de pontos texturizada [52]. O exemplo de um ortomosaico pode ser visto na figura 2.7.

2.5 Ground Control Points

Qualquer objeto com uma forma visível e identificável a uma certa altitude pode ser considerado um *ground control point*, no entanto, existem objetos com formas e padrões que são facilmente reconhecidos independentemente do meio onde estão colocados.

2.5.1 Formato

Um bom GCP deverá ser identificável a uma grande altitude, ter uma forma ou cores que o diferenciem do meio onde está colocado e ter um ponto bem definido que seja propício colocar o GPS [2]. Independentemente do formato dos GCP utilizados estes devem ter um acabamento mate/fosco, pois se forem brilhantes, dependendo do ângulo em que as

fotos são tiradas, podem acabar por refletir luz e atrapalhar a identificação dos mesmos nas imagens [29].

Na fig 2.8 encontram-se exemplos de GCP normalmente utilizados [25, 42].

Figura 2.8: GCP normalmente utilizados no mapeamento de território.

2.5.2 Ficheiro de identificação de coordenadas GCP

No WebODM o ficheiro de identificação de coordenadas GCP pode ser gerado de 3 formas, sendo 2 delas semiautomáticas e uma delas manual. Este ficheiro irá conter as coordenadas GPS dos ground control points assim como a sua localização, em píxeis, em cada imagem.

Se o ficheiro for criado manualmente é necessário inserir, logo na primeira linha, a projeção utilizada nas coordenadas GPS colocadas no ficheiro. A partir da segunda linha podemos colocar, separadas por espaços em branco, as coordenadas reais retiradas do GPS (latitude, longitude e altitude), as *associated pixel coordinates*, o nome da imagem e opcionalmente na última coluna o id do ponto de controlo em questão. Ficando o ficheiro com a seguinte sintaxe:

```
geo_x  geo_y  geo_z  im_x  im_y  image_name  id
```

Por exemplo:

```
-9.0925361863 38.78221709 17.21 291.1999 879.66254 DJI_0911.JPG 52
```

Na figura 2.9 é possível ver um exemplo de um ficheiro de identificação de coordenadas GCP.

Na primeira coluna encontra-se a latitude do ground control point, na segunda coluna a longitude, na terceira a altitude, na quarta a localização horizontal do ground control point na imagem (x píxeis), na quinta a localização vertical (y píxeis), na sexta o nome da imagem em questão e na sétima coluna o número/id do ground control point.

A maioria destes dados são fáceis de preencher manualmente à excepção da localização dos píxeis que contêm o GCP em cada imagem (*associated píxel coordinates*), daí serem utilizadas interfaces que ajudam o utilizador a marcar com elevada precisão a localização dos mesmos nas imagens.

+proj=utm +zone=29 +ellps=WGS84 +datum=WGS84 +units=m +no_defs						
668874.17	4585131.24	732.959	2411.75	1602.00	DJI_0070.JPG	2
668874.17	4585131.24	732.959	2374.00	2438.50	DJI_0071.JPG	2
668874.17	4585131.24	732.959	2369.50	3224.00	DJI_0072.JPG	2
668874.17	4585131.24	732.959	1321.25	448.00	DJI_0122.JPG	2
668874.17	4585131.24	732.959	1197.00	1212.75	DJI_0123.JPG	2
668874.17	4585131.24	732.959	1054.00	1966.00	DJI_0124.JPG	2
668874.17	4585131.24	732.959	922.50	2591.75	DJI_0125.JPG	2
668874.17	4585131.24	732.959	3333.00	3630.91	DJI_0126.JPG	2
668973.51	4585186.51	732.996	1007.25	3269.25	DJI_0197.JPG	3
668973.51	4585186.51	732.996	1294.25	852.00	DJI_0194.JPG	3
668973.51	4585186.51	732.996	4120.25	877.00	DJI_0254.JPG	3
668973.51	4585186.51	732.996	4483.25	2887.50	DJI_0255.JPG	3
668973.51	4585186.51	732.996	3948.50	262.50	DJI_0256.JPG	3
668973.51	4585186.51	732.996	3824.50	790.50	DJI_0257.JPG	3
668973.51	4585186.51	732.996	3970.50	1575.00	DJI_0258.JPG	3
668973.51	4585186.51	732.996	4115.50	2327.25	DJI_0259.JPG	3
668973.51	4585186.51	732.996	4218.90	3096.31	DJI_0260.JPG	3
669045.34	4585212.60	719.028	165.00	1512.00	DJI_0260.JPG	8
669045.34	4585212.60	719.028	265.46	2127.97	DJI_0261.JPG	8
669045.34	4585212.60	719.028	368.75	2721.75	DJI_0262.JPG	8
669045.34	4585212.60	719.028	477.00	3323.25	DJI_0263.JPG	8
669045.34	4585212.60	719.028	3572.26	29.64	DJI_0296.JPG	8
669045.34	4585212.60	719.028	3594.00	676.75	DJI_0297.JPG	8
669045.34	4585212.60	719.028	3590.25	1307.00	DJI_0298.JPG	8
669045.34	4585212.60	719.028	3575.50	1932.75	DJI_0299.JPG	8
669075.81	4585204.24	708.052	128.50	3054.50	DJI_0265.JPG	10
669075.81	4585204.24	708.052	22.81	2514.66	DJI_0264.JPG	10

Figura 2.9: Exemplo de um ficheiro de identificação de coordenadas GCP com a formatação necessária para o ODM.

No caso das outras duas formas semiautomáticas, uma delas, consiste na utilização do *software* POSM GCPi que é incluído por *default* com o WebODM. Este apenas automatiza o selecionar no ecrã a localização dos GCP em cada imagem, de forma a identificar em que píxeis X e Y é que o ground control point se encontra. No entanto esta procura pelos GCPs é manual. É necessário fazer upload das imagens obtidas por drone e identificar os pontos de controlo em pelo menos três imagens por cada GCP [52]. O outro *software* semiautomático faz a mesma coisa que este mas com uma interface diferente, no entanto não vem por *default* instalado com o WebODM sendo, portanto, necessário a sua instalação à parte.

Mas embora já instalado e disponível no WedODM, foi difícil compreender como é que o *software* de marcação de GCPs funciona. A informação disponível no site do WebODM e em alguns artigos encontrados na internet é insuficiente para a sua utilização. Apenas foi encontrada informação útil e detalhada no livro *OpenDroneMap: The Missing Guide* [52].

No entanto, coloca-se a questão:

Porque é que ainda não existe uma solução mais rápida e prática para este problema da identificação dos ground control points no WebODM? Embora o projeto seja *open source*, existe um certo sentido crítico aquando da adição de novas *features* ao projeto por parte dos seus fundadores.

Um dos fundadores do projeto WebODM, *Piero Toffanin*, explica o porquê do projeto ODM ainda não ter um sistema de identificação automático de marcadores:

"*The biggest issue with auto-detecting markers is that everyone uses different markers, or sometimes not even markers, they just measure street corners or other features on the ground. So a solution needs to be generic enough to be able to detect a large sets of different/common markers.*"[49]

Atualmente existem vários projetos que tentam resolver o problema de reconhecimento de pontos de controle do ODM, como por exemplo o projeto *Automatic Recognition of ArUco Codes in Land Surveying Tasks* [47] mencionado na secção 2.3.1 e o projeto *odm_gcp_apriltag* [30] mencionado na secção 2.3.3.

2.5.3 Porquê usar Ground Control Points?

Os GCP irão ajudar a gerar uma ortofotografia/modelo tridimensional mais rigoroso e preciso. Como indicado no manual ODM, os GCPs são usados de forma a reduzir o erro de georreferenciação das imagens.

"GCP observations are used to to minimize the georeferencing alignment error, but they can't compensate for other factors such as an incorrect camera model estimate due to a sub-optimal flight path or excessive camera lens distortion... "[52]

O WebODM utiliza na maioria das etapas de processamento desde o Structure from Motion (1º etapa) até à adição de Texturas (4º etapa) um sistema de coordenadas local e é com esse sistema de coordenadas que faz todo o processamento. No entanto caso esteja disponível meta-informação GPS nas fotografias ou então, preferencialmente, no ficheiro de identificação de coordenadas GCP, essa informação é utilizada na primeira fase do pipeline e é gerado um ficheiro de transformação de coordenadas locais em coordenadas reais que será utilizado na 5º fase (Georreferenciação) do pipeline [32, 34, 52]. É relevante referir que a meta-informação disponível nas imagens é geralmente guardada no formato EXIF. A parte dita manual do *software* de geração do ficheiro de coordenadas GCP tem a vantagem de qualquer ponto poder ser um ground control point. Desta forma a esquina de uma casa ou um poste de luz podem ser utilizados como um GCP. Para automatizar esta parte irá ser necessário utilizar ground control points específicos como é o caso dos AprilTags, referidos na secção 2.3.3 ou de ArUco markers, como o projeto da secção 2.3.1

utiliza. Esta abordagem irá trazer vantagens mas também desvantagens pois embora seja possível a identificação automática do GCP, perde-se flexibilidade na escolha mesmo.

3

Arquitetura

Neste capítulo aborda-se a arquitetura da solução desenvolvida no âmbito desta dissertação, o algoritmo GCP Finder. Abaixo encontram-se dois diagramas, um deles ilustra o algoritmo implementado em python que irá gerar um ficheiro de coordenadas GCP e o outro a arquitetura do sistema em geral na qual se encontra a relação entre o ficheiro de coordenadas GCP e o WebODM.

3.1 Diagramas

3.1.1 Diagrama do Algoritmo GCP Finder

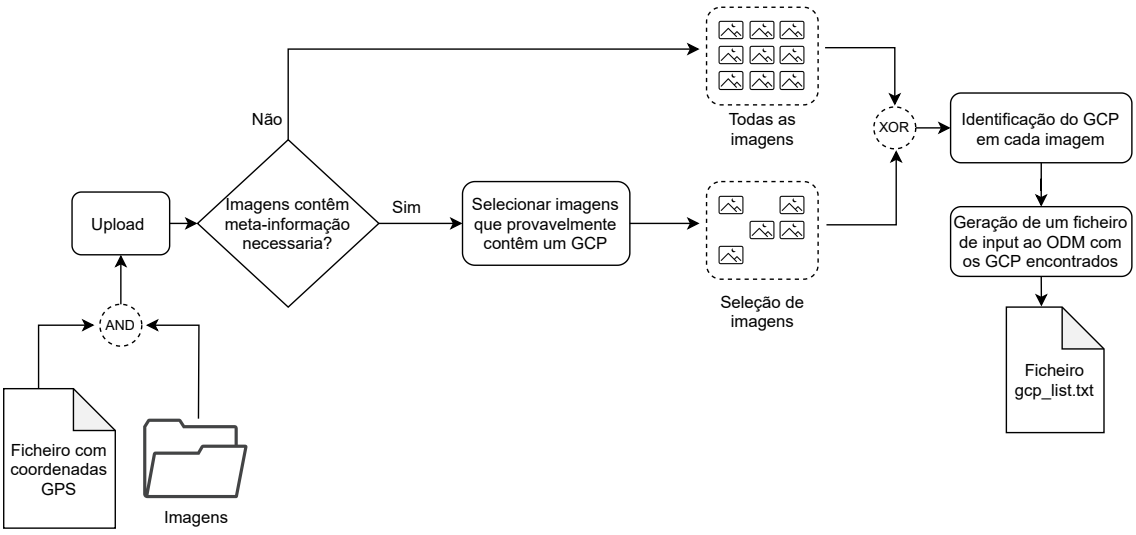

Figura 3.1: Diagrama do algoritmo GCP Finder.

O algoritmo consiste nas seguintes etapas:

1. Carregar todas as imagens do levantamento tiradas pelo drone, para o sistema, juntamente com o ficheiro com as coordenadas GPS dos pontos de controlo.

2. Verificar se existe meta-informação disponível nas imagens e em caso positivo utilizar esses dados para selecionar as imagens onde provavelmente existe um ponto de controlo. Caso falte informação em alguma das imagens, o sistema procura por pontos de controlo em todas as imagens carregadas.

3. Identificar os ground control points nas imagens que foram selecionadas na etapa anterior. Caso não esteja disponível meta-informação, o *software* irá tentar identificar ground control points em toda as imagens carregadas para o sistema. É com base nos marcadores utilizados no projeto descrito na secção 2.3.1 (ArUco Markers) que será feita a identificação.

4. Gerar o ficheiro de coordenadas GCP que contem a relação entre as coordenadas reais dos pontos de controlo e as suas respectivas coordenadas x e y em cada imagem. Este ficheiro tem obrigatoriamente que ter o nome *gcp_list.txt* pois será utilizado como ficheiro de configuração no ODM. O utilizador apenas irá interagir com o sistema na 1ª etapa e nesta última.

3.1.2 Diagrama da Arquitetura

A arquitetura presente na figura 3.2 ilustra o upload das fotos do levantamento para o sistema presente na 3.1 juntamente com o ficheiro com as coordenadas GPS dos pontos de controlo. Após esse passo, é necessário voltar a fazer upload das mesmas fotos para o WebODM, juntamente o ficheiro de identificação de coordenadas GCP gerado pelo algoritmo GCP Finder de forma a ser feita a georreferenciação das imagens com o mínimo de erro possível e assim levar à produção de um ortomosaico ou de um modelo digital de elevação preciso.

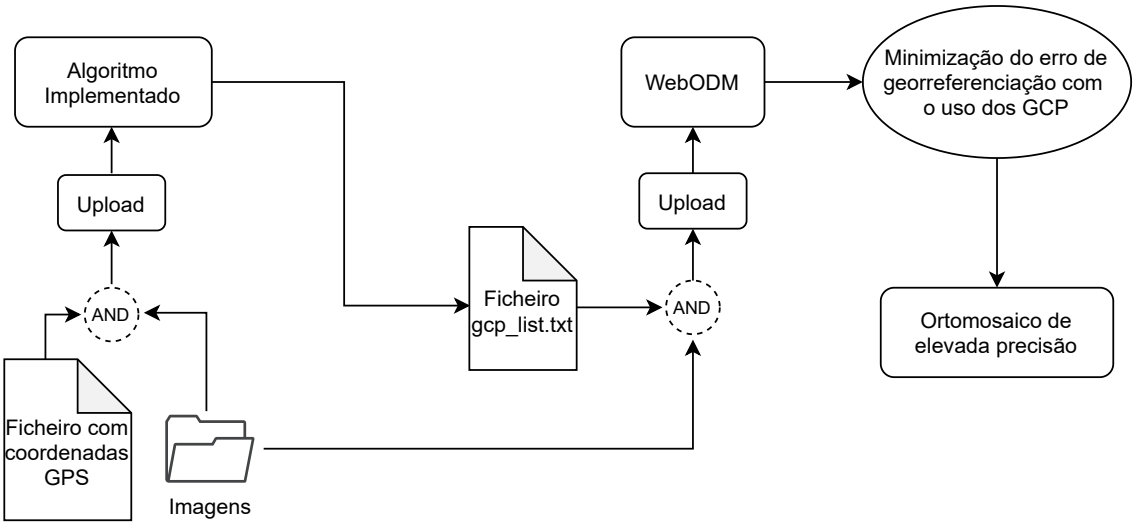

Figura 3.2: Diagrama da arquitetura.

3.1.3 Diagrama de Classes

O diagrama de classes da figura 3.3, relaciona as várias classes que compõem o algoritmo GCP Finder, sendo a principal a classe GCP Finder. É nesta classe que é feito todo o processamento das imagens, contando com o auxilio das classes Statistics, Image e GroudControlPoint. Cada instância da classe GCP Finder, inicializa uma classe Statistics, uma ou mais classes Image e uma ou mais classes GroudControlPoint.

Na secção 4.1 do próximo capítulo é explicado com mais detalhe a importância que cada uma destas classes tem no algoritmo.

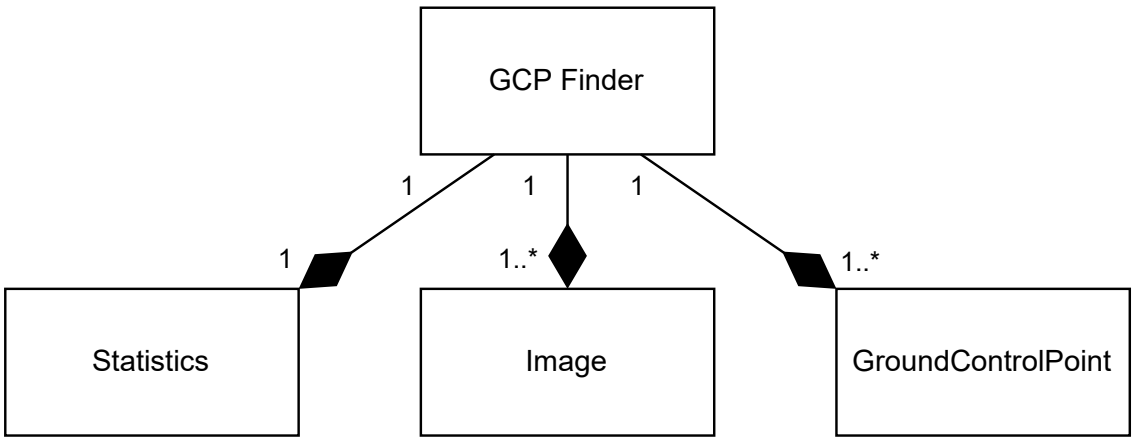

Figura 3.3: Diagrama de classes simplificado do algoritmo GCP Finder.

3.2 Validação

Na presente secção são detalhados os dados e as métricas utilizadas na construção do algoritmo GCP Finder.

3.2.1 Ground Sample Distance (GSD)

O Ground Sample Distance (GSD) representa a distância que 1 píxel no ecrã representa na realidade. Se considerarmos, por exemplo, que o GSD é de 10 cm/píxel, isto quer dizer que cada píxel na imagem representa 10 cm na realidade.

Com este valor conseguimos obter o tamanho aproximado de objetos numa imagem. O cálculo do GSD é feito segundo a equação 3.1:

$$GSD = \frac{Altitude\ do\ voo \times Largura\ do\ sensor}{Distância\ focal \times Largura\ da\ imagem} \tag{3.1}$$

O cálculo do GSD é dado pela divisão entre o produto da altitude do voo pela largura do sensor e entre o produto da distância focal pela largura da imagem [1]. Destas 4 variáveis que fazem parte da equação, 3 delas são constantes durante todo o voo. Tanto a largura do sensor que equipa o drone como o tamanho da imagem têm os mesmos valores pois que nem a qualidade nem o formato das imagens irá sofrer alterações durante o voo. A distância focal é também constante e é ditada pela lente que equipa o drone. O único valor que varia em cada foto é a altitude, que por mais que o drone tente, nunca consegue ficar à mesma altitude. Esta instabilidade deve-se normalmente às condições atmosféricas presentes no momento do voo.

3.2.2 Altura do voo

A altitude a que o drone voa durante um levantamento topográfico varia sempre um pouco durante o voo mas essa variação fica registada na meta-informação de cada foto, o que permite processar a imagem com o mesmo rigor. No entanto, o valor da altitude a que a foto foi tirada é obtido através de um barómetro presente no drone e esta informação tem algum erro associado [53]. Consequentemente, como o cálculo do GSD utiliza a altitude de cada foto, este não é tão extremamente preciso. A presença deste erro propaga-se e faz com que a estimativa das coordenadas GPS nas fotos tenha também algum erro associado.

3.2.3 Estimativa das coordenadas GPS nas imagens

De forma a selecionar as imagens que provavelmente contêm um ponto de controlo, determina-se uma área a "procurar" pela presença de um ponto de controlo. Com base no ficheiro que contêm a informação GPS dos pontos de controlo que foram colocados no terreno e na meta-informação disponível nas imagens, é possível determinar a área de inclusão, que está representada na fig 4.1, em que se procura pela presença de algum GCP. Caso esteja algum GCP dentro da área de inclusão, essa imagem é selecionada, e é adicionada ao conjunto de imagens que provavelmente contêm um ponto de controlo. Esta área de inclusão é determinada com base no GSD, no pitch e no yaw.

De forma a calcular esta área, começa-se por obter o *pitch* e o *yaw* a partir da meta-informação contida nas imagens. O *pitch* irá ser a inclinação da câmera (para cima e para baixo face ao drone) e o *yaw*, o ângulo da orientação horizontal da câmera face ao drone (para a esquerda e para a direita). A distância a que o centro da imagem está da posição do drone (distância **d** na figura 3.4) é calculada através divisão entre a altitude e a tangente do ângulo de inclinação da câmera. Esta distância é depois usada para obter aproximadamente as coordenadas reais do centro da imagem. Utilizando o GSD ou seja, o valor que cada píxel corresponde na realidade e sabendo as coordenadas reais do centro da imagem, conseguimos saber aproximadamente quais são as coordenadas reais dos cantos da imagem.

Na fig 3.5 estão presentes as rotações *pitch* e *yaw*.

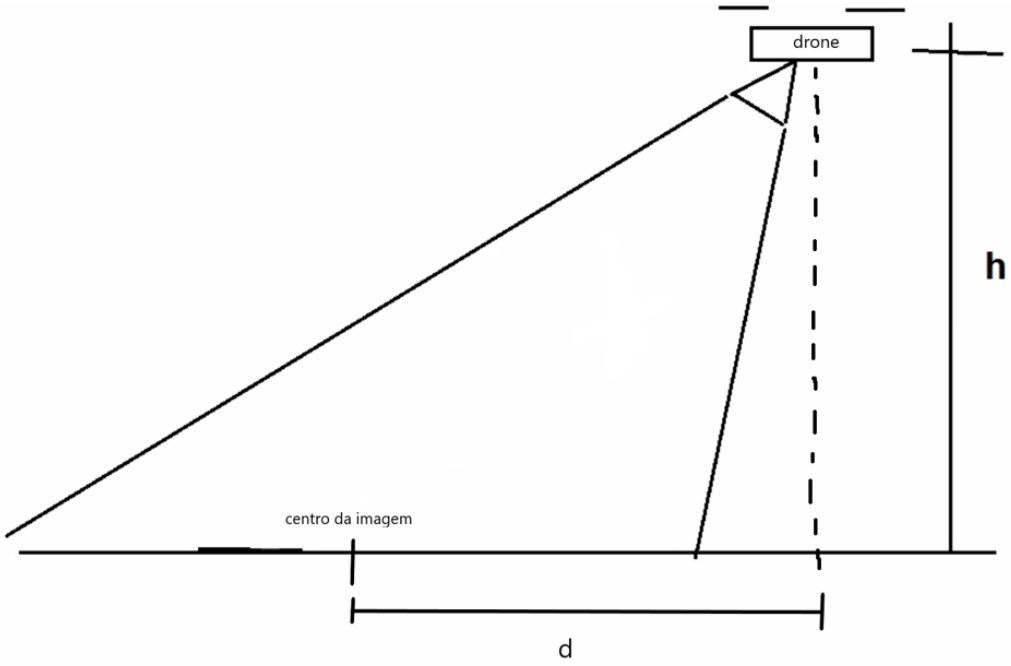

Figura 3.4: Esquemático que ilustra a área capturada pelo drone numa imagem.

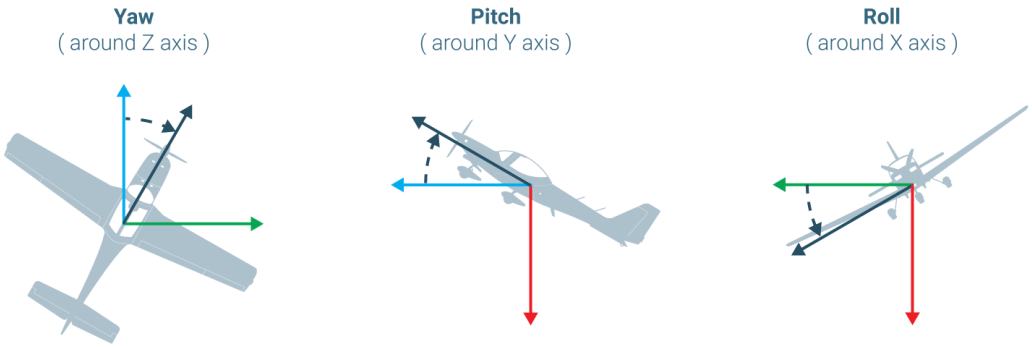

Figura 3.5: Ilustração do Pitch, Yaw e Roll [44].

Se o *pitch* = 90º, quer dizer que a câmara do drone está apontada para o chão. Se o *pitch* = 0º, quer dizer que a câmara está a apontar para o horizonte. No caso do *yaw*, se este for de 0º significa que a câmara do drone está a apontar para Norte. Se for de 90º, a câmara está a apontar para Este, com 180º está a apontar para Sul e assim sucessivamente.

4

Implementação

Ao longo do desenvolvimento da dissertação, surgiu a necessidade de desenvolver duas versões do algoritmo GCP Finder. Uma versão Web, que ficaria alojada no servidor do LNEC, estando assim disponível em qualquer parte do mundo, sendo apenas necessário autenticar-se perante o sistema para utiliza-la. Esta versão facilitaria a utilização do algoritmo GCP Finder, sendo que não seria necessário instalação. Infelizmente não foi possível disponibilizar uma nova porta no servidor do LNEC para o algoritmo GCP Finder, impossibilitando desta forma implementar o que tinha ficado previsto inicialmente. Ficando a versão Web do algoritmo disponível no GitHub, onde é possível fazer clone do repositório para o nosso computador seguindo os passos descritos em 4.4.2.

No entanto, não é necessário acesso à internet para utilizar o algoritmo GCP Finder. Foi criada uma versão que "corre" localmente ao ser executada na linha de comandos, em inglês Command-line interface (CLI). Esta versão não possui uma interface "apelativa" como a versão Web, no entanto torna a execução mais flexível, disponibiliza uma API base para interação com o sistema e obtém o mesmo resultado de forma mais rápida, devido ao facto de não ser necessário fazer o upload das imagens para o servidor Web.

A abordagem que foi seguida de forma atingir os objetivos propostos nesta dissertação consistiu no seguinte:

Começar por reduzir substancialmente a quantidade de imagens a processar na identificação dos GCP através da extração de meta-informação GPS das imagens e selecionar apenas para processamento mais pesado, ou seja, a identificação precisa do centro dos GCP com a biblioteca ArUco, as imagens que contenham dentro de uma área delimitada pelo *software*, qualquer ponto de coordenadas GPS conhecidas e que foram recolhidas manualmente. Assim reduzir-se-ão substancialmente o número de imagens a processar, ficando apenas com aquelas que muito provavelmente contêm um ponto de controle. Na figura 4.1 pode ser vista a área que o algoritmo GCP Finder irá delimitar, caso seja selecionada uma margem de 20%.

Após esta separação inicial as imagens irão passar por outra fase que consiste na identificação dos pontos de controlo com a ajuda do modulo ArUco da biblioteca OpenCV apenas nas imagens que foram selecionadas na etapa anterior, diminuindo assim o tempo de

Figura 4.1: Área selecionada pelo *software* caso seja utilizada uma margem de 20%.

execução total do algoritmo, reduzindo o número de imagens a processar e contribuindo para automatizar uma tarefa que até hoje é feita manualmente para os utilizadores do Open Drone Map.

4.1 Implementação do Diagrama de Classes

O diagrama de classes presente na figura 4.2 ilustra quais os tipos de variáveis utilizadas no algoritmo GCP Finder assim como todas as funções e relações entre as várias classes.

A função ***is_gcp_nearby*** é uma das funções mais importantes do algoritmo GCP Finder, pois valida se é provável existir um ponto de controlo numa imagem durante a primeira etapa de processamento. Em caso negativo, a imagem é descartada e não segue para a segunda etapa de processamento.

A função ***aruco_detect*** é responsável pela segunda etapa de processamento, onde é feita a deteção dos ArUco Markers através da aplicação digital de filtros e do modulo aruco da biblioteca OpenCV.

A função ***get_corner_coordinates*** retorna o ângulo entre o centro da imagem e o seu canto. Este ângulo é de 45º quando a imagem é quadrada. A função é ***get_corner_-coordinates*** é chamada na função ***is_gcp_nearby***, pois é com base no output desta função

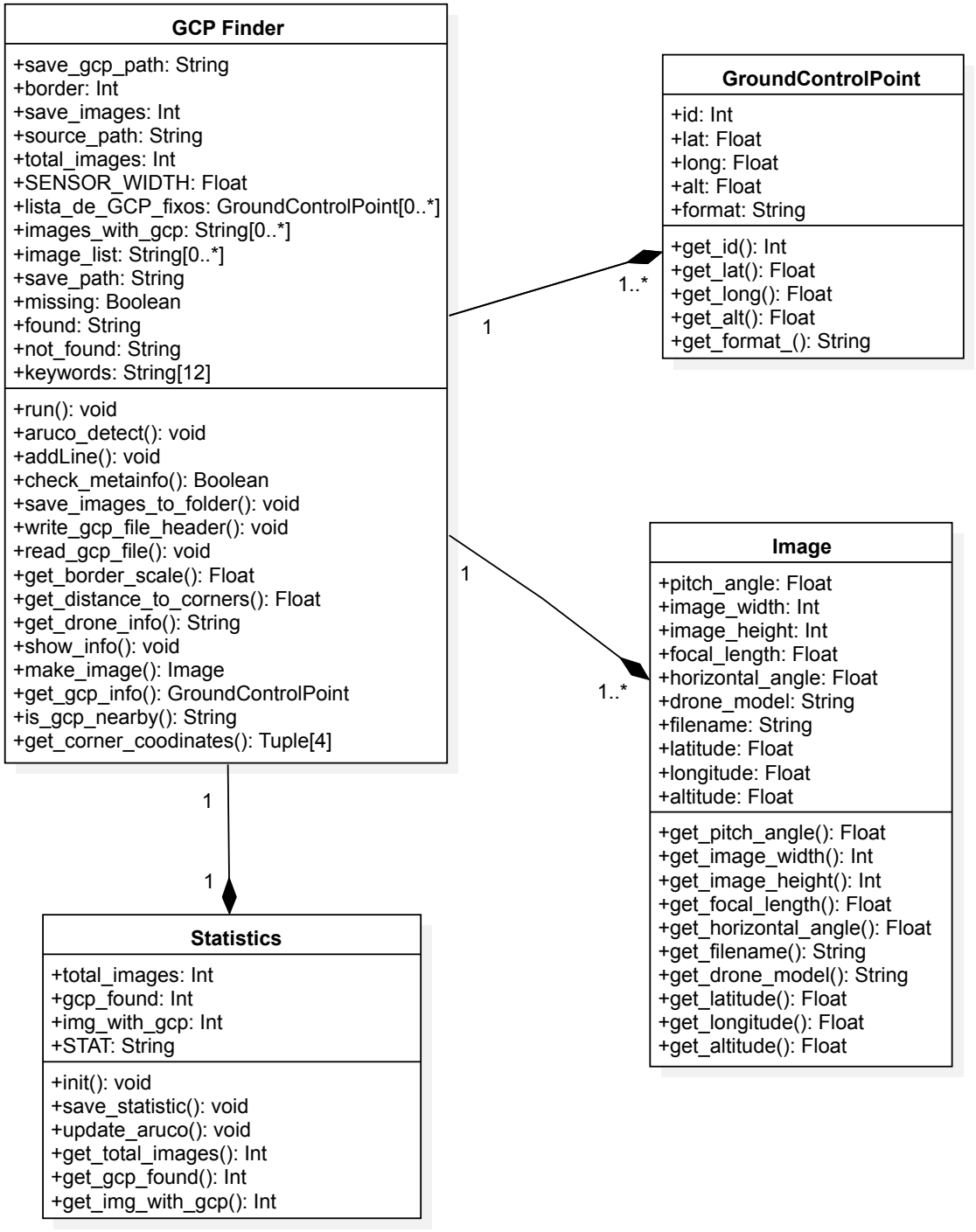

Figura 4.2: Diagrama de classes do algoritmo GCP Finder.

em comparação com a localização GPS dos pontos de controlo que é determinado se é provável uma imagem ter visivelmente um ponto de controlo.

A função ***run*** é a que inicia o algoritmo, tratando do *upload* das imagens para memória assim como do *upload* das coordenadas GPS dos pontos de controlo que foram

inicialmente espalhados pela barragem. É responsável também pela extração da meta-informação disponível nas imagens, criando e inicializando os objetos da classe Image, Statistics e GroundControlPoint.

A função ***addLine*** como o própio nome indica, cria e adiciona uma nova linha ao ficheiro de coordenadas GCP, com a identificação de um Aruco Marker feita pelo algoritmo GCP Finder.

A função ***save_images_to_folder*** apenas é usada quando o utilizador indica que pretende que as imagens identificadas com pontos de controlo sejam guardadas numa pasta. Sendo assim cada imagem que seja identificada com um ponto de controlo é escrita para disco numa pasta indicada pelo utilizador no início da execução do algoritmo GCP Finder.

A função ***get_distance_to_corners*** é uma função auxiliar da função ***get_corner_coordinates*** que através da métrica GSD, mencionada na secção 3.2.1 calcula a distância a que os cantos da imagem estão do seu centro. Esta informação é relevante para a primeira etapa de processamento, onde são excluídas as imagens que não contêm pontos de controlo.

4.2 Interface Web e CLI

A versão com interface Web vem facilitar a interação do utilizador com o algoritmo GCP Finder, no entanto não é necessário utilizar a interface para tirar partido do algoritmo. Esta também pode ser acedida através da linha de comandos, de forma rápida e prática.

4.2.1 GCP Finder - Web Interface

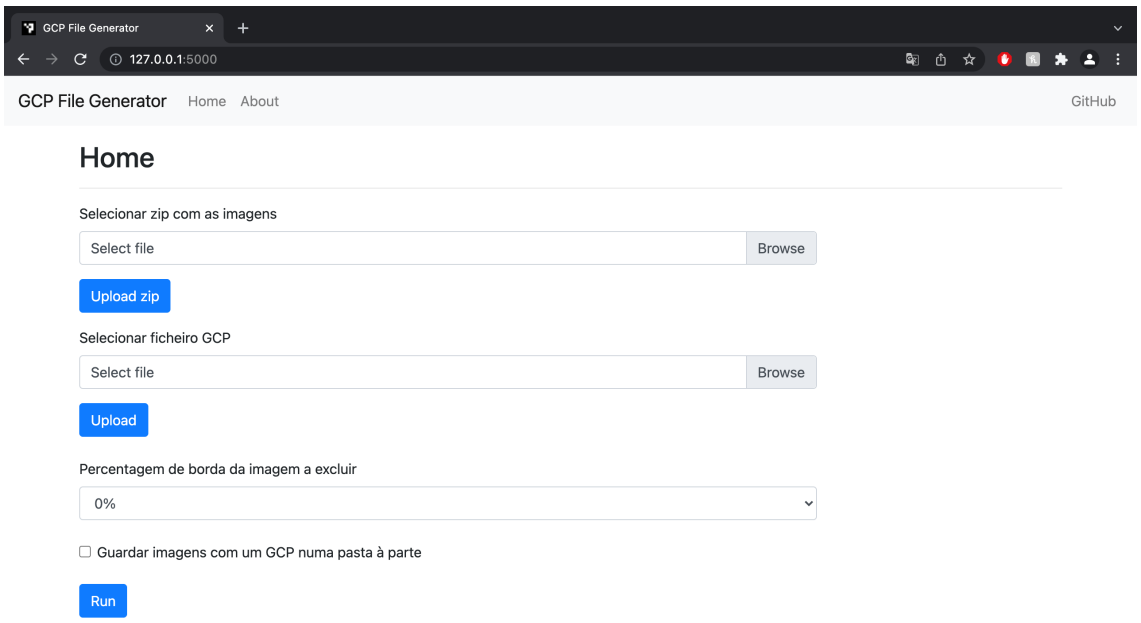

Figura 4.3: *Homepage* da Interface Web GCP Finder.

A Interface Web do algoritmo GCP Finder foi criada com recurso ao Flask, uma *micro-framework* Web escrita em python que permite tornar o desenvolvimento de aplicações Web em python mais rápido, eficiente e escalável [7]. Foi utilizado também a framework de frontend Bootstrap, uma das frameworks de frontend mais populares [8], o que nos permitiu obter uma interface com um aspecto sólido e apelativo. Comecemos por explicar como é que esta interface pode ser utilizada.

Como se pode ver na figura 4.3, o utilizador na página principal tem que fazer o upload das fotos, clicando na barra de pesquisa para selecionar o zip com as fotos do levantamento. Após selecionar o zip o utilizador tem que premir o botão "Upload zip" para confirmar o upload das fotos. Logo abaixo o utilizador volta a fazer o mesmo mas para o ficheiro de coordenadas GPS, selecionando o ficheiro desejado e clicando no botão "Upload". Estando estas duas etapas concluídas o utilizador tem que selecionar a percentagem de margem a excluir de cada imagem. Esta margem serve para excluir pontos de controlo que possam ter sido capturados pelo drone nas bordas das imagens, estando estes desfocados ou cortados. Um exemplo em que o ponto de controlo está parcialmente cortado pode ser visto na figura 4.4. No entanto caso o utilizador selecione a opção **0%** o algoritmo irá procurar por pontos de controlo em toda a imagem. Por fim o utilizador pode assinalar se quer ficar com uma cópia das imagens em que foram identificados pontos de controlo ou não.

Figura 4.4: Imagem com um ponto de controlo parcialmente cortado (à esquerda).

Clicando no botão "Run", o programa começa a processar as imagens e o utilizador é levado para uma nova página, que pode ser vista na figura 4.5 em que irá obter informação sobre a execução do programa, juntamente com uma barra de progresso.

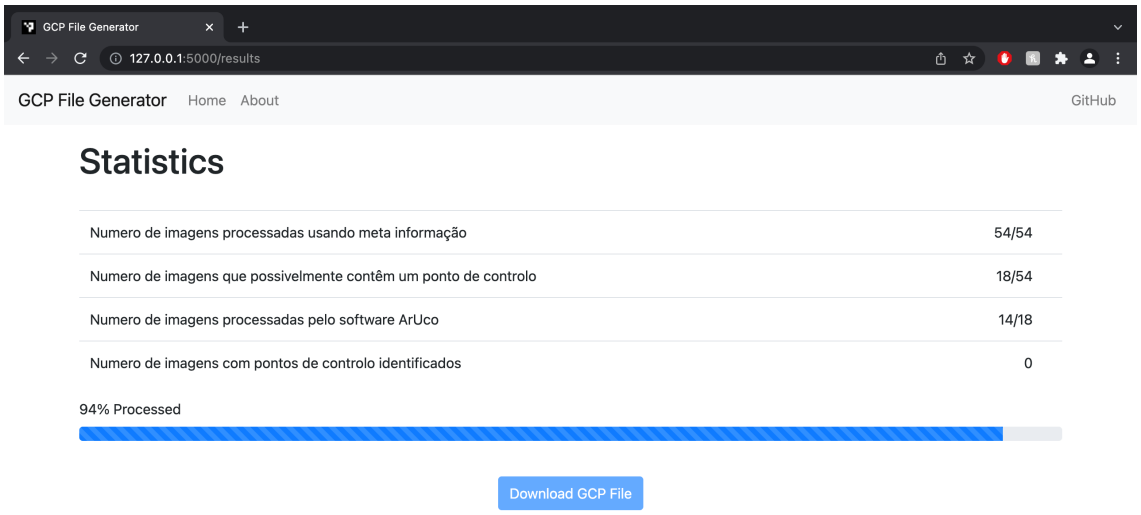

Figura 4.5: Página que mostra o programa em execução.

Quando todas as imagens forem processadas o botão "Download GCP File" ficará disponível e o utilizador ao clica-lo, descarrega para o seu computador o ficheiro "gcp_list.txt". Ficheiro este que contem a associação entre as coordenadas GPS dos pontos de controlo e as coordenadas X e Y em píxeis das imagens onde estes pontos se encontram. Este ficheiro servirá, posteriormente, de input ao WebODM.

Na imagem 4.6 é possível ver o fim da execução do algoritmo, estando disponível para download o ficheiro "gcp_list.txt".

Embora seja mencionado um botão para guardar as imagens que contenham pontos de controlo numa pasta à parte, esta *feature* não está disponível neste momento na versão Web do algoritmo GCP Finder. Ficará para trabalho futuro pois requer uma implementação diferente da versão CLI. As imagens na versão Web precisam de ser comprimidas antes de serem descarregadas.

4.2.2 GCP Finder - Command Line Interface

O algoritmo GCP Finder pode também ser utilizado através da linha de comandos, utilizando a seguinte sintaxe:

```
$ python GCP_Finder.py arg1 arg2 arg3 [options] opt1
```

- **arg1:** Caminho para a pasta com as imagens a processar.
- **arg2:** Caminho para o ficheiro de texto com a localização GPS dos pontos de controlo colocados no terreno.

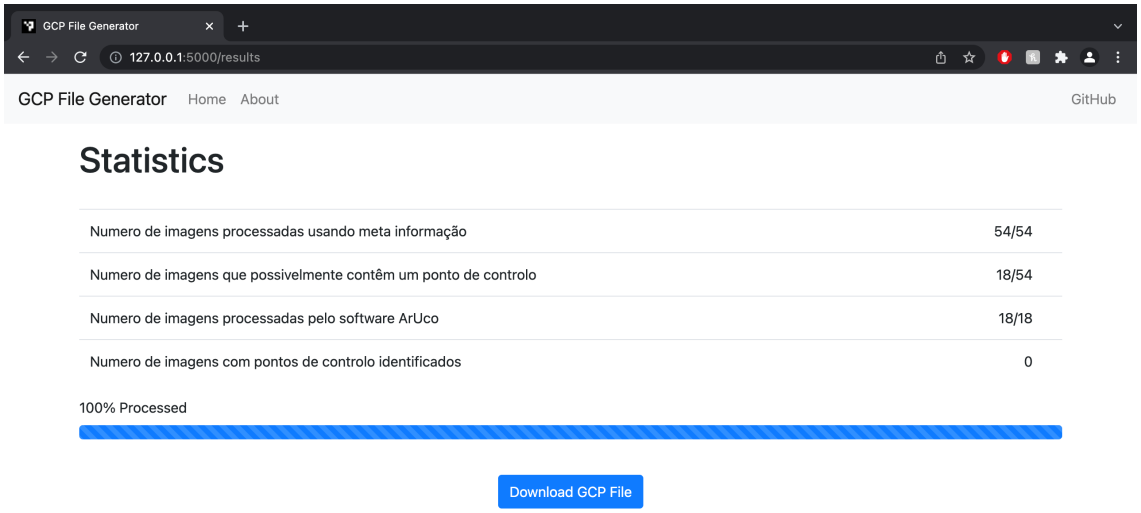

Figura 4.6: Página que mostra o resultado da execução do algoritmo.

- **arg3:** Percentagem de borda a retirar de cada imagem.
- **opt1:** Caso o utilizador queira guardar as imagens em que foram encontrados pontos de controlo, este deve indicar o caminho da pasta para onde estas imagens devem ser guardadas.

O resultado final ("gcp_list.txt") é guardado na diretoria corrente. Assim, um exemplo de comando a executar seria:

```
$ python GCP_Finder.py ./sample/ gps_coordinates.txt 20 ./fotos_selecionadas/
```

4.3 Tipo de Ground Control Point

Inicialmente para efectuar os levantamentos topográficos, existiam à escolha 3 tipos de marcadores a colocar no solo. Um marcador com padrão de xadrez, ou seja, 2 quadrados brancos e dois pretos, utilizar os ArUco markers ou então AprilTags. Não foi possível com a biblioteca Checkerboard, que iria identificar o centro dos marcadores com padrão de xadrez, obter resultados satisfatórios. Muitas vezes a identificação feita pela biblioteca era incorrecta. Muito provavelmente por esta não estar feita com este proposto mas sim para identificar padrões de xadrez maiores, como se de um tabuleiro de xadrez se tratasse. Sendo assim restaram os AprilTags e os ArUcos Markers que são ambos semelhantes. Tanto os AprilTags como os ArUco markers têm diversos dicionários com que podem ser utilizados, com diferentes graus de complexidade, traduzido pelo número de bits, isto é, o número de blocos brancos e pretos que compõem o marcador. Na imagem 2.3 é possível ver um marcador com 16 bits, que se traduz por, 4 quadrados na vertical por 4 quadrados na horizontal, sendo estes brancos ou pretos. A utilização de um baixo número de bits, neste caso 4x4 (16 bits) é recomendada pois estes marcadores tem que ser visíveis em

fotografias tiradas a grandes altitudes (50m a 100m). Como foi explicado na secção 2.3.3 do capítulo 2, a única familia de AprilTags de 16 bits apenas consegue gerar 30 ids únicos. Ao contrario dos ArUco Markers, que conseguem gerar até 1000 ids únicos com uma família de 16 bits. Por este motivo foram utilizados os ArUco Markers como GCPs para o desenvolvimento dessa dissertação.

Na figura 4.7 encontra-se um dos GCPs ArUco utilizados no levantamento topográfico de dia 17/12/21.

Figura 4.7: GCP com id=0 utilizado no levantamento topográfico de dia 17/12/21.

4.4 Proof of Concept - Setup

Para dar uso ao algoritmo GCP Finder pela primeira vez, seja utilizando uma interface Web ou simplesmente na linha de comandos, é necessário seguir os seguintes passos:

4.4.1 Linha de Comandos

Utilizando a CLI, começa-se por fazer clone do projeto para o nosso computador.

```
$ git clone https://github.com/octaviojardim/GCP_Finder
```

depois entra-se na diretoria do projeto:

```
$ cd GCP_Finder/
```

Com o comando *make* é construído um ambiente virtual e são instaladas as dependências necessárias ao projeto:

```
$ make install
```

Depois é necessário introduzir a palavra passe do utilizador do computador para instalar a biblioteca ExifTool. Após este passo é necessário ativar o ambiente virtual:

```
$ source ./venv/bin/activate
```

Por fim basta utilizar o algoritmo com o comando:

```
$ python GCP_Finder.py arg1 arg2 arg3 [options] opt1
```

E substituir os argumentos pelos parâmetros corretos, como foi indicado em 4.2.2.

4.4.2 Interface Web

A versão Web da algoritmo GCP Finder pode ser instalada no nosso computador através dos seguintes passos:

```
$ git clone https://github.com/octaviojardim/Web-Interface-GCP-Finder
```

E depois entrar na diretoria do projeto:

```
$ cd Web-Interface-GCP-Finder/
```

Com o comando *make* é construído um ambiente virtual e são instaladas as dependências necessárias ao projeto:

```
$ make install
```

Depois é necessário introduzir a palavra passe do utilizador do computador para instalar a biblioteca ExifTool. Após este passo é necessário ativar o ambiente virtual e fazer *deploy* da aplicação:

```
$ make run
```

Por fim aceder ao endereço no *browser* e utilizar o algoritmo como indicado em 4.2.1.

```
$ http://127.0.0.1:5000
```

5

Avaliações e Testes

De forma a assegurar a qualidade dos dados obtidos pelo algoritmo GCP Finder, procede-se à sua validação nos testes que se seguem. Para estes testes foram utilizados três conjuntos de dados. Dois conjuntos de dados foram obtidos num levantamento topográfico efetuado no Parque das Nações no dia 17/12/21 com dois drones. Um DJI Phantom 3 Pro e um DJI Phantom 4 Pro. Com o primeiro drone foi construído um dataset com 121 imagens e com o segundo, outro dataset com 134 imagens. A precisão da identificação feita pelo algoritmo GCP Finder, assim como o seu tempo de execução serão avaliados. O GPS utilizando durante os levantamentos topográficos efetuados foi um GPS Geodésico Topcon HiPer Pro com recetor e base, disponibilizado pelo LNEC para este propósito.

Foi também efetuado um outro levantamento, desta vez na Barragem da Lapa em Sardoal (Santarém), onde foi possível obter dados mais próximos da realidade com que o LNEC está habituado a trabalhar.

5.1 Datasets

A tabela 5.1 resume os vários levantamentos com recurso a drones efetuados ao longo desta dissertação, contribuindo assim para a construção de datasets utilizando ArUco Markers e marcadores em xadrez.

O dataset A foi recolhido pelo Eng. Daniel Leite numa das campanhas efetuadas pelo LNEC aquando da inspeção da Barragem de Sambade, situada no município de Alfândega da Fé, distrito de Bragança, em Dezembro de 2019. O drone utilizado para este voo foi o mesmo utilizado no dataset A, um DJI Phantom 4 Pro que capturou 1429 imagens a cerca de 60 m altitude. Como este levantamento foi efetuado antes do desenvolvimento desta dissertação, os marcadores utilizados como GCPs foram em xadrez.

O dataset B foi obtido no Parque das Nações em Lisboa utilizando um drone DJI Phantom 3 Pro, que capturou 121 imagens com 4000 x 3000 píxeis de resolução. O voo foi feito a cerca de 60 m de altitude, pois embora tenha sido configurado no plano de voo uma altitude fixa, com o vento que se fazia sentir aquando do voo, a altitude do voo oscilou alguns centímetros. Este dataset foi recolhido no dia 17/12/22 e foram utilizados

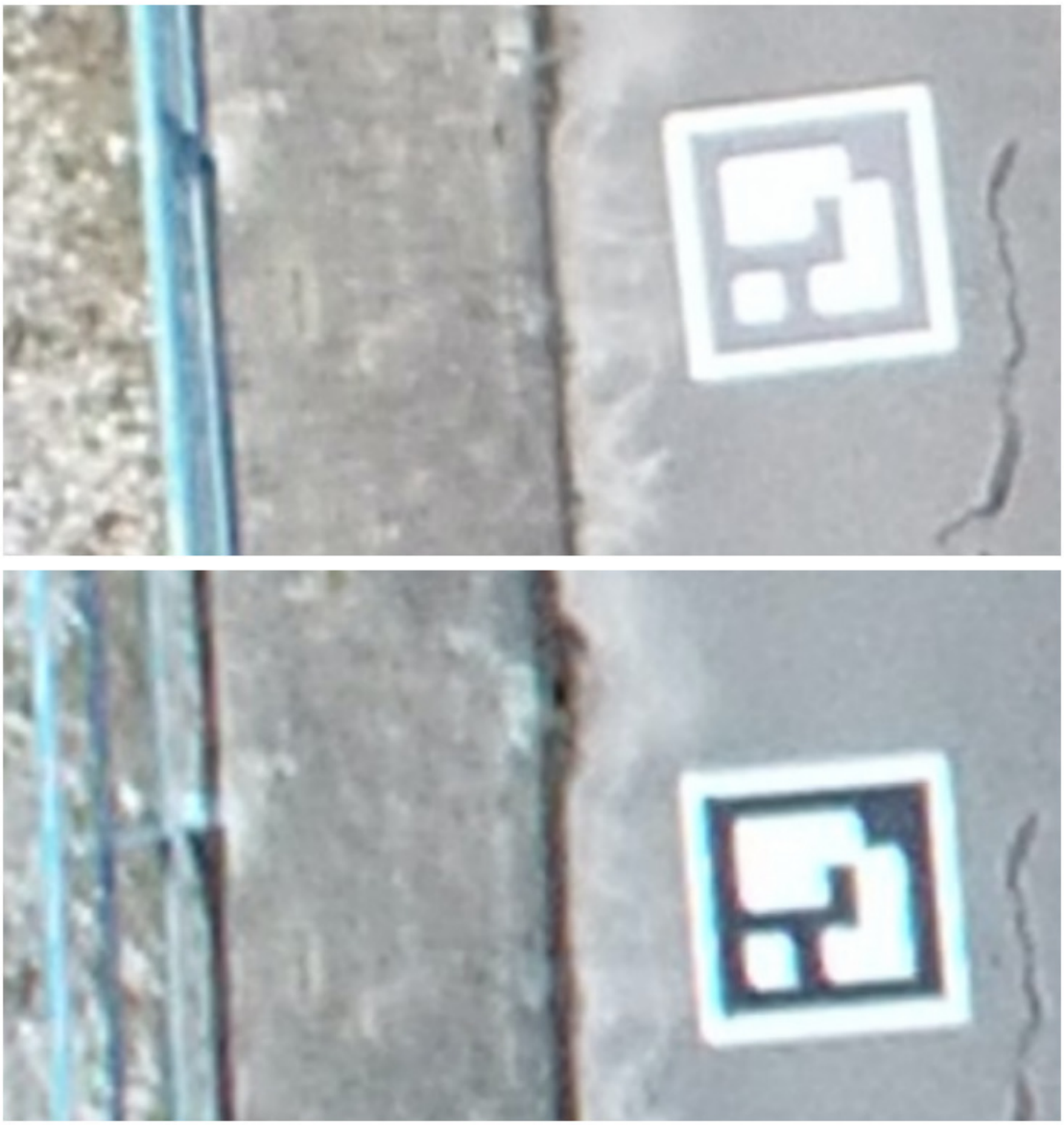

Figura 5.1: Imagens capturadas no levantamento da Barragem da Lapa a cerca de 60m de altitude. Na imagem de cima é possível o reflexo do sol no marcador ArUco, dificultando a identificação feita pelo software. Na segunda imagem é possível observar as cores reais do marcador (preto e branco).

4 marcadores ArUco com cerca de 1x1 m cada um.

O dataset C também foi recolhido no Parque das Nações utilizando os mesmos marcadores ArUco com a mesma configuração no solo. No entanto foi utilizado um drone DJI Phantom 4 Pro, que capturou 134 imagens com 4864 x 3648 píxeis de resolução. A altitude configurada no plano de voo foi a mesma do dataset anterior (60 m).

O dataset D foi recolhido numa das campanhas mais recentes efetuadas pelo LNEC aquando da inspeção da Barragem da Lapa, na freguesia de Sardoal, que se encontra neste

Tabela 5.1: Datasets recolhidos.

Dataset	A	B	C	D
Drone	DJI 4	DJI 3	DJI 4	DJI 4
Tipo de GCP	Xadrez	ArUco	ArUco	ArUco
Número de imagens	1429	121	134	134
Altura do voo (m)	60	60	60	60+
Área total (m2)	~104000	~22000	~22000	~21500
Número de GCPs	11	4	4	10
Resolução das imagens (píxeis)	4864 x 3648	4000 x 3000	4864 x 3648	4864 x 3648
Estado do Tempo	Nublado	Sol e vento fraco	Sol e vento fraco	Sol e vento moderado
Local	Barragem de Sambade	Parque das Nações	Parque das Nações	Barragem da Lapa
Data	13/12/2019	17/12/21	17/12/21	19/05/2022

momento com um grande assentamento em diversas partes. Este levantamento foi efetuado no dia 19/05/2022. O drone utilizado para este voo foi o mesmo em levantamentos passados, um DJI Phantom 4 Pro que capturou 134 imagens a cerca de 60 m altitude a partir da crista da barragem. No entanto devido às condições climatéricas que se fazia sentir na altura (vento moderado) e à hora do voo (próximo do meio dia) não nos foi possível obter dados melhores do que aqueles recolhidos no Parque das Nações. Devido ao sol estar no seu ponto mais alto, fez com os marcadores refletissem a luz solar, não sendo possível capturar o marcador corretamente como é visível na figura 5.1. O vento moderado também não ajudou, tendo movido alguns marcadores de sítio.

Num próximo levantamento estes percalços irão ser tomados em consideração de forma a ser possível obter dados com maior qualidade do que aqueles obtidos no levantamento da Barragem da Lapa no dia 19/05/2022.

Na figura 5.2 é possível observar a nuvem de pontos da Barragem da Lapa construida utilizando o algoritmo GCP Finder juntamente com o WebODM.

Pelas razões mencionadas no início do capítulo 5, os testes que se seguem irão utilizar os datasets B e C provenientes do levantamento efetuado Parque das Nações no dia 17/12/21. Esperemos no futuro obter mais datasets, obtidos por diferentes *drones* e a diferentes altitudes, contribuindo assim para futura investigação na identificação automática de GCPs.

Todos os datasets recolhidos durante a escrita dessa dissertação estão disponíveis no Google Drive[1], assim como os ficheiros de coordenadas GCP e os ficheiros com coordenadas GPS.

[1] https://bit.ly/3RzjkDp

Figura 5.2: Nuvem de pontos da Barragem da Lapa gerada pelo WebODM utilizando as imagens recolhidas no levantamento efetuado dia 19/05/2022.

5.2 Testes

Como já foi descrito no inicio do capítulo 4, o algoritmo GCP Finder assenta em duas etapas de processamento, sendo uma precedida de outra. A primeira, utiliza a meta informação GPS disponível nas imagens e verifica se existe algum ponto de controlo na proximidade, em caso positivo seleciona essa imagem para a segunda etapa, em caso negativo, essa imagem é rejeitada. Na segunda etapa o programa tenta identificar os pontos de controlo nas imagens que foram selecionadas na etapa anterior. De forma a avaliar a qualidade dessa identificação, foram efetuados os testes que se seguem.

5.2.1 Quantidade de GCP encontrados

Este primeiro teste avalia a quantidade de GCPs encontrados pelo *software* nas imagens, face ao número real de GCPs visíveis.

O algoritmo GCP Finder utiliza o modulo ArUco disponível na biblioteca OpenCV, sendo assim possível parametrizar este modulo aquando da identificação dos pontos de controlo presentes nas imagens. No levantamento efetuado no Parque das Nações com o *drone* DJI Phantom 4 Pro, obtivemos 134 imagens, das quais 84 continham pontos de controlo. Neste levantamento a percentagem de imagens com GCP é de cerca de 63%, o que é um valor alto face a outros levantamentos que foram feitos. No entanto este valor

deve-se ao facto de neste levantamento a área a mapear ser pequena face ao número de marcadores que foram colocados. Numa área com cerca de 22000 m2 encontravam-se 4 marcadores. No levantamento efectuado na Barragem de Sambade a 13/12/2019, por exemplo, a percentagem de imagens com GCP foi de 20%. Nesse levantamento foram utilizados 11 marcadores no entanto a área a mapear era muito superior.

Processando as imagens do levantamento efetuado no Parque das Nações com o *drone* DJI Phantom 4 Pro, utilizando o algoritmo GCP Finder obtivemos uma taxa de identificação na ordem dos 71%, ou seja, das 84 imagens que continham marcadores foram identificados 60. Estes resultados podem ser melhorados com uma melhor parametrização do modulo ArUco assim como um melhor pré-processamento das imagens.

5.2.2 Precisão da identificação do centro dos GCPs nas imagens

Neste segundo teste é possível comparar o erro obtido na identificação do centro do GCP feita pelo *software*.

É selecionada uma amostra de imagens e verifica-se qual é a precisão obtida quando é identificado o centro do ponto de controlo através do algoritmo GCP Finder. O erro é determinado com base no número de píxeis que o centro real do ponto de controlo dista da identificação feita pelo *software*.

Foi obtido um erro médio no primeiro dataset de 1.10 cm traçando diagonais sobre o marcador e calculando a sua intersecção. Utilizando a mediada dos lados foi obtido um erro de 0.99 cm. No segundo dataset o erro médio foi de 0.86 cm com a técnica das diagonais e de 0.70 cm utilizando a mediana dos lados.

Com os resultados obtidos conclui-se que a melhor forma de obter o centro dos GCPs nas imagens é através da mediana dos lados e foi dessa forma que o cálculo do centro do GCP foi implementado no algoritmo GCP Finder. Na figura 5.3 é possível observar a identificação feita utilizando as duas técnicas.

De forma a avaliar o rigor do algoritmo GCP Finder irá ser calculado a *Accuracy*, *Precision* e *Recall* do algoritmo utilizando dois datasets.

Accuracy descreve a relação entre as imagens que foram bem categorizadas entre todas as imagens que foram processadas. Ou seja, quantas imagens que não tinham pontos de controlo foram selecionadas como não tendo pontos de controlo e quantas com pontos foram selecionadas como tendo pontos de controlo?

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{5.1}$$

Na primeira etapa foi obtido *accuracy* de 89% no dataset D3 e 84% no dataset D4.

Na segunda etapa foi obtido *accuracy* de 78% no dataset D3 e 81% no dataset D4.

Estes valores são justificados pela não identificação de alguns pontos de controlo mesmo quando eles estão presentes nas imagens. Embora não seja o ideal, o que não se

Figura 5.3: Identificação do centro de um ponto de controlo feita pelo algoritmo GCP Finder. A vermelho tem-se a identificação utilizando a mediana dos lados, e a verde utilizado a intersecção das diagonais. A imagem foi ampliada e capturada a 60 m de altura.

quer que aconteça é a identificação incorreta de pontos de controlo pois dessa forma a precisão do levantamento iria diminuir drasticamente.

A *Precision* descreve a relação entre os *true posives* e os *false positives*, por outras palavras, quantas das imagens que foram classificadas como tendo um ponto de controlo realmente o têm?

$$Precision = \frac{TP}{TP + FP} \tag{5.2}$$

Na primeira etapa foi obtido *precision* de 79% no dataset D3 e 80% no dataset D4.

Na segunda etapa foi obtido *precision* de 100% no dataset D3 e 100% no dataset D4.

Os valores obtidos na etapa 1 são justificados pela falta de precisão no cálculo da altura do drone, que irá influenciar o cálculo do GSD, assim como o cálculo dos limites da imagem. Obtendo por vezes classificações erradas quando existem pontos de controlo perto dos limites da imagem. No entanto a etapa 2 é a mais crítica pois influencia toda a precisão do levantamento. Os pontos identificados na 2º etapa são escritos diretamente para o ficheiro de coordenadas GCP que irá servir de configuração para o modelo gerado

pelo ODM. É importante este valor ser o mais elevado possível e nos dois datasets que foram utilizados, foi obtido *precison* de 100%, o que é excelente.

Recall descreve a relação entre os *true positives* e os *false negatives*. Por outras palavras, daquelas imagens que realmente contêm um ponto de controlo, quantas delas foram selecionadas como imagens contendo um ponto de controlo?

$$Recall = \frac{TP}{TP + FN} \tag{5.3}$$

Na primeira etapa foi obtido *recall* de 98% no dataset D3 e 100% no dataset D4.

Na segunda etapa foi obtido *recall* de 38% no dataset D3 e 70% no dataset D4.

Embora tenham sido obtidos bons valores na primeira etapa, na segunda isso não se verifica. Uma melhor parametrização da biblioteca ArUco para a deteção dos pontos de controlo irá melhorar estes valores.

Para estes testes foram utilizados os dados do levantamento topográfico efetuado no Parque das Nações no dia 17/12/21 com dois *drones*. Um DJI Phatom 3 e um DJI Phantom 4.

As matrizes de confusão, podem ser consultadas nas tabelas 5.2, 5.3, 5.4 e 5.5.

Tabela 5.2: Matriz de confusão da etapa 1 usando o dataset D3.

		Valor Previsto	
	D3	Sim	Não
Valor Verdadeiro	Sim	44	1
	Não	12	64

$$Accuracy = \frac{44 + 64}{44 + 64 + 12 + 1} = 0.8926$$

$$Precision = \frac{44}{44 + 12} = 0.7857$$

$$Recall = \frac{44}{44 + 1} = 0.9778$$

Tabela 5.3: Matriz de confusão da etapa 1 usando o dataset D4.

		Valor Previsto	
	D4	Sim	Não
Valor Verdadeiro	Sim	87	0
	Não	22	25

$$Accuracy = \frac{87 + 25}{87 + 25 + 22 + 0} = 0.8358$$

$$Precision = \frac{87}{87 + 22} = 0.7982$$

$$Recall = \frac{87}{87+0} = 1$$

Tabela 5.4: Matriz de confusão da etapa 2 usando o dataset D3.

		Valor Previsto	
	D3	Sim	Não
Valor Verdadeiro	Sim	16	27
	Não	0	78

$$Accuracy = \frac{16+78}{16+78+0+27} = 0.7769$$

$$Precision = \frac{16}{16+0} = 1$$

$$Recall = \frac{16}{27} = 0.3721$$

Tabela 5.5: Matriz de confusão da etapa 2 usando o dataset D4.

		Valor Previsto	
	D4	Sim	Não
Valor Verdadeiro	Sim	60	24
	Não	0	50

$$Accuracy = \frac{60+50}{60+50+0+24} = 0.8201$$

$$Precision = \frac{60}{60+0} = 1$$

$$Recall = \frac{60}{60+24} = 0.7142$$

5.2.3 Tempo de execução do *software* para imagens com e sem meta-informação

As imagens que possuem a meta-informação necessária para a conclusão da primeira etapa de processamento irão beneficiar numa redução do tempo total de processamento do algoritmo. Meta-informação como o *yaw*, o *pitch* e a geolocalização de cada imagem são essenciais para a primeira etapa de processamento do algoritmo GCP Finder, descrita no início do capítulo 4. Embora esta etapa seja opcional, ela reduz imenso o tempo de processamento total das imagens. No levantamento com o *drone* DJI Phantom 4 Pro, feito no Parque das Nações, se for considerada a meta-informação disponível nas imagens, obtém-se um tempo total de processamento na ordem dos 2 min e 54 s utilizando um MacBook Pro de 2015 com um Intel Core i5 a 2,7 GHz dual core e 8 GB de RAM DDR3.

Desta forma foram selecionadas apenas 84 imagens das 134 que foram carregadas. Se não estivesse disponível meta-informação nas imagens ou se essa informação estivesse incompleta, o tempo de processamento seria de 4 min e 29 s, pois o algoritmo iria ter procurar por GCPs em todas as imagens que foram carregadas, neste caso 134 imagens. No entanto este valor ainda é elevado. Isto deve-se ao facto do algoritmo apenas selecionar imagens com pontos de controlo e estas estarem presentes em menor número face ao número de imagens total. No levantamento feito à Barragem de Sambade, apenas 20% das imagens continham pontos de controlo. Um valor bem diferente dos 63% de imagens com pontos de controlo no levantamento no Parque das Nações.

5.3 Discussão de resultados

Ao longo do desenvolvimento da dissertação a interface GCP Finder foi sendo constantemente melhorada até chegar ao seu estado atual. Com o levantamento efetuado na Barragem da Lapa foi possível comprovar a relevância prática do GCP Finder. Existem inúmeras vantagens em utilizar a ferramenta GCP Finder em comparação com o *software* POSM GCPi disponibilizado por default pelo WebODM, como por exemplo a deteção automática de pontos de controlo. O POSM GCPi necessita de intervenção humana para identificar a localização dos pontos de controlo nas imagens ao contrário do GCP Finder. Outra vantagem trata-se do tempo total de interação com estas interfaces de pré processamento das imagens (POSM GCPi e GCP Finder). Apesar de não ter sido possível obter dados quantitativos que comprovem as vantagens em utilizar o algoritmo GCP Finder é de esperar que este seja mais rápido e eficiente do que um operador humano.

6

Conclusões e trabalho futuro

6.1 Conclusões

Com o aparecimento de novas tecnologias associadas a drones, faz todo o sentido passar a integra-las na verificação periódica das barragens. Algo que é feito até ao momento de uma forma mais manual e trabalhosa, pode ser otimizada graças ao avanço tecnológico que houve nos últimos anos e que nos permite obter um modelo tridimensional de forma mais prática e eficiente.

Esta dissertação tem o objetivo de tornar mais rápido e prático a aferição de danos nas barragens oferecendo uma interface simples e apelativa para a identificação automática de GCPs. Esta interface tanto pode ser utilizada pelos engenheiros do LNEC, que fazem visitas regulares às barragens para a aferição de deslocamentos ou fissuras, mas também por qualquer pessoa que se interesse por uma solução *open source* de identificação automática de pontos de controlo.

O algoritmo desenvolvido ao longo desta dissertação está disponível na página do GitHub mencionada na secção 4.4 e espera ser uma contribuição e ajuda para o LNEC, que agora tem ao seu dispor o GCP Finder, uma alternativa ao POSM GCPi, que vem instalado por default com o WebODM mas que não permite uma identificação automática dos pontos de controlo.

Na secção 2.3.1 do capítulo 2, é mencionado o projeto *opensource* Find-GCP de identificação de marcadores ArUco desenvolvido por professores da Universidade de Tecnologia e Economia de Budapeste. Este projeto embora semelhante em alguns aspetos com o algoritmo desenvolvido nesta dissetação, está carente de uma interface Web, sendo neste caso o utilizador obrigado a utilizar a linha de comandos, o que pode não ser habitual para a grande maioria dos utilizadores. O projeto supramencionado não faz qualquer tipo de processamento aquando da identificação dos ArUco markers nas imagens. O algoritmo GCP Finder desenvolvido no âmbito desta dissertação, pelo contrário, faz uma pré-seleção das imagens a identificar de forma a reduzir substancialmente o tempo geral de processamento. Esta técnica é explicada com mais detalhe no início do capítulo 4.

O levantamento topográfico efetuado a barragens tornou-se muito mais simples devido ao *software* desenvolvido.

6.2 Trabalho Futuro

O projeto desenvolvido está disponível no GitHub e de forma a continuar a desenvolver o mesmo, sugerem-se as seguintes melhorias:

- Criar uma interface para aferir a precisão da identificação do centro da imagem e ajusta-la se necessário.
- Melhorar a parametrização do modulo ArUco de forma a obter um maior número de GCPs identificados e um melhor valor de *accuracy*, *precision* e *recall*.
- Realizar um novo levantamento em breve (idealmente até Maio de 2023) de forma a ser termo de comparação com o levantamento efetuado no dia 19/05/2022.
- Fazer o setup de mais nós de processamento, no WebODM disponível no servidor do LNEC.

Com a criação de uma interface para validar a identificação feita pelo algoritmo GCP Finder, iria ser possível aumentar ainda mais a precisão do levantamento efetuado pois quaisquer erros durante a fase de identificação dos GCPs iriam ser eliminada nesta fase. O ponto menos positivo desta abordagem seria o acrescimento de tempo despendido nesta etapa. Além disso seria relevante concluir a implementação do algoritmo GCP Finder na versão Web, pois como foi referido na subsecção 4.2.1, o botão utilizado para guardar as imagens que contêm pontos de controlo numa pasta à parte, não está disponível neste momento.

Com o intuito de melhorar a identificação feita pelo algoritmo GCP Finder e de aumentar o número de GCPs encontrados deve-se proceder a uma melhoria na parametrização do modulo ArUco. Utilizando datasets recolhidos por vários drones e com diferentes combinações de parâmetros em conjunto com filtros de imagem, acredita-se conseguir obter resultados ainda melhores, mais precisos e de forma mais rápida.

Seria interessante comparar os dados deste novo levantamento com os dados do levantamento efetuado à Barragem da Lapa este ano. Sendo esta uma barragem que teve um grande assentamento no passado, seria interessante acompanhar o estado da barragem e ver se há evolução no futuro.

Neste momento apenas está disponível um nó de processamento no WebODM instalado no servidor do LNEC e dado que o ODM suporta múltiplos nós de processamento, seria interessante tirar partido dessa opção, configurando mais nós e desta forma utilizar datasets maiores sem aumentar o tempo de processamento geral.

Bibliografia

[1] V. Aerial. *What is Ground Sample Distance (GSD)?* Online; Last accessed 09 January 2022. Abr. de 2021. URL: https://visionaerial.com/what-is-ground-sample-distance/ (ver p. 24).

[2] P. Aero. *Things To Know About Ground Control in Drone Surveying*. Last accessed 4 May 2021. Dez. de 2016. URL: www.propelleraero.com/blog/things-to-know-about-ground-control-in-drone-surveying (ver p. 16).

[3] Agência Portuguesa do Ambiente. *"Barragens de Portugal,"Last accessed 15 July 2021*. URL: https://apambiente.pt/prevencao-e-gestao-de-riscos/barragens-de-portugal (ver p. 1).

[4] Agisoft. *"Buy,"Last accessed 14 June 2021*. URL: www.agisoft.com/buy/online-store (ver p. 8).

[5] L. Agisoft. *Agisoft Metashape User Manual, Professional edition, Version 1.7. Agisoft LLC, St. Petersburg, Russia*. 2021 (ver pp. 8, 9).

[6] Agisoft HelkDesk. *"Coded targets and Scale bars", Last accessed 16 June 2021*. URL: https://agisoft.freshdesk.com/support/solutions/articles/31000148855-coded-targets-and-scale-bars (ver p. 9).

[7] amigos-maker. *What is Flask used for?* Nov. de 2019. [Online]. URL: https://dev.to/amigosmaker/what-is-flask-used-for-2do5 (ver p. 30).

[8] Bootstrap. *"About,"Last accessed 7 December 2021*. URL: https://getbootstrap.com/docs/5.1/about/overview/ (ver p. 30).

[9] T. E. o. E. Britannica. *Aerial photography*. Mar. de 2019. [Online]. URL: www.britannica.com/technology/aerial-photography (ver p. 6).

[10] O. G. Consortium. *OGC GeoTIFF Standard*. URL: https://www.ogc.org/standards/geotiff (acedido em 13/07/2022) (ver p. 17).

[11] S. Doggett. *How to Create Orthomosaic Maps Using DroneDeploy*. Dronegenuity. Dez. de 2019. [Online]. URL: www.dronegenuity.com/create-orthomosaics-dronedeploy (ver p. 8).

[12] S. Doggett. *What Is An Orthomosaic? Orthomosaic Maps & Orthophotos Explained*. URL: https://www.civilsimplified.com/resources/what-is-total-station (acedido em 15/05/2022) (ver p. 5).

[13] DroneDeploy. *"Automated GCP Detection with DroneDeploy,"Last accessed 12 June 2021*. URL: https://support.dronedeploy.com/docs/automated-gcp-detection-with-dronedeploy (ver pp. 8, 9).

[14] DroneDeploy. *"Drone Mapping Software,"Last accessed 19 June 2021*. URL: www.dronedeploy.com (ver p. 9).

[15] DroneDeploy. *"GCP Request Checklist,"Last accessed 14 June 2021*. URL: https://support.dronedeploy.com/docs/gcp-request-checklist-1 (ver p. 8).

[16] DroneDeploy. *"Unlock the Power of Drone Data,"Last accessed 12 June 2021*. URL: www.dronedeploy.com/pricing.html (ver p. 8).

[17] DroneMapper. *"DroneMapper,"Last accessed 13 June 2021*. URL: https://dronemapper.com/pricing (ver pp. 8, 9).

[18] DroneMapper. *"DroneMapper,"Last accessed 13 June 2021*. URL: https://dronemapper.com (ver pp. 8, 9).

[19] DroneMapper. *"DroneMapper,"Last accessed 16 June 2021*. URL: https://dronemapper.com/software_downloads (ver p. 9).

[20] DroneMapper. *MicMac*. Last accessed 12 June 2021. 2019. URL: https://github.com/dronemapper-io/micmac (ver p. 8).

[21] A. Engenharia. *Levantamento topográfico: Importância e aplicações*. URL: https://https://aeroengenharia.com/levantamento-topografico (acedido em 12/05/2022) (ver p. 2).

[22] N. Fiedler. "Distributed Multi Object Tracking with Direct FCNN Inclusion in RoboCup Humanoid Soccer". Tese de doutoramento. Jun. de 2019 (ver p. 14).

[23] S. Garrido-Jurado et al. "Automatic generation and detection of highly reliable fiducial markers under occlusion". Em: *Pattern Recognition* 47.6 (2014), pp. 2280–2292. DOI: https://doi.org/10.1016/j.patcog.2014.01.005 (ver p. 10).

[24] GeoSLAM. *Point Clouds for Beginners*. URL: https://geoslam.com/point-clouds/ (acedido em 29/03/2022) (ver p. 5).

[25] Ground Control Points. *"Creating Quality GCPs for Mapping Contour Lines,"Last accessed 12 July 2021*. URL: www.groundcontrolpoints.com/mapping-contour-lines-using-gcp (ver p. 17).

[26] J. T. Hastings e L. L. Hill. "Georeferencing". Em: *Encyclopedia of Database Systems*. Ed. por L. LIU e M. T. ÖZSU. Boston, MA: Springer US, 2009, pp. 1246–1249. ISBN: 978-0-387-39940-9. DOI: doi.org/10.1007/978-0-387-39940-9_181 (ver p. 15).

[27] A. Jain, M. Mahajan e R. Saraf. "Standardization of the Shape of Ground Control Point (GCP) and the Methodology for Its Detection in Images for UAV-Based Mapping Applications". Em: *Advances in Computer Vision*. Ed. por K. Arai e S. Kapoor. Cham: Springer International Publishing, 2020, pp. 459–476. ISBN: 978-3-030-17795-9 (ver p. 1).

[28] O. Kalachev. *ArUco markers generator!* URL: https://chev.me/arucogen/ (acedido em 16/01/2022) (ver p. 11).

[29] J. Kuze. *Ground Control Points (GCPs) for aerial photography*. Last accessed 17 May 2021. Fev. de 2015. URL: https://diydrones.com/profiles/blogs/ground-control-points-gcps-for-aerial-photography (ver p. 17).

[30] langeroo. *odm_gcp_apriltag*. Last accessed 29 June 2021. 2021. URL: https://github.com/langeroo/odm_tools/tree/main/odm_gcp_apriltag (ver p. 19).

[31] D. Nugent. *Designing the perfect Apriltag*. URL: https://optitag.io/blogs/news/designing-your-perfect-apriltag (acedido em 13/07/2022) (ver p. 17).

[32] O. A. ODM. *OpenDroneMap/ODM GitHub Page 2020*. Last accessed 3 May 2021. 2021. URL: https://github.com/OpenDroneMap/ODM (ver pp. 8, 9, 13, 14, 19).

[33] E. Olson. "AprilTag: A robust and flexible visual fiducial system". Em: *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, mai. de 2011, pp. 3400–3407 (ver p. 13).

[34] OpenStreetMap US. *"OpenDroneMap" Oct. 22, 2017*. Last accessed 3 May 2021. URL: www.youtube.com/watch?v=gHftqakS5lw (ver pp. 15, 19).

[35] M. Pierrot-Deseilligny et al. "Micmac, apero, pastis and other beverages in a nutshell". Em: *Institut Géographique National* (2014) (ver p. 14).

[36] Pix4D. *"Image acquisition,"Last accessed 12 July 2021*. URL: https://support.pix4d.com/hc/en-us/articles/115002471546-Image-acquisition (ver p. 6).

[37] Pix4D. *"Maximum number of images per project and processing time", Last accessed 13 June 2021*. URL: https://support.pix4d.com/hc/en-us/articles/202558769-Maximum-number-of-images-per-project-and-processing-time (ver p. 8).

[38] Pix4D. *"PIX4Dcapture", Last accessed 10 June 2022*. URL: https://www.pix4d.com/product/pix4dcapture (ver p. 10).

[39] Pix4D. *"PIX4Dcloud - FAQ", Last accessed 13 June 2021*. URL: https://support.pix4d.com/hc/en-us/articles/215858503-PIX4Dcloud-FAQ (ver p. 8).

[40] Pix4D. *"Pix4dmapper Photogrammetry Software", Last accessed 8 June 2021*. URL: www.pix4d.com/product/pix4dmapper-photogrammetry-software (ver pp. 8–10).

[41] Pix4D. *"Pix4dmapper", Last accessed 13 June 2021*. URL: www.pix4d.com/pricing (ver p. 8).

[42] Pix4D. *Ground control points: why are they important?* Pix4d. Dez. de 2019. [Online]. URL: www.pix4d.com/blog/why-ground-control-points-important (ver p. 17).

[43] Pix4D. *Image acquisition*. Online; Last accessed 13 July 2021. Jan. de 2020. URL: https://support.pix4d.com/hc/en-us/articles/115002471546-Image-acquisition (ver p. 7).

[44] Pix4D. *Yaw, Pitch, Roll and Omega, Phi, Kappa angles*. Online; Last accessed 2 December 2021. Fev. de 2017. URL: https://support.pix4d.com/hc/en-us/articles/202558969-Yaw-Pitch-Roll-and-Omega-Phi-Kappa-angles (ver p. 25).

[45] J. L. Schönberger e J.-M. Frahm. "Structure-from-Motion Revisited". Em: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016, pp. 4104–4113. DOI: doi.org/10.1109/CVPR.2016.445 (ver p. 6).

[46] Z. Siki e B. Takács. "Automatic Recognition of ArUco Codes in Land Surveying Tasks". Em: *Baltic Journal of Modern Computing* 9 (jan. de 2021). DOI: doi.org/10.22364/bjmc.2021.9.1.06 (ver p. 10).

[47] Z. SIKI e B. TAKÁCS. *Find-GCP*. Last accessed 23 May 2021. 2021. URL: https://github.com/zsiki/Find-GCP (ver pp. 10, 19).

[48] C. Simplified. *What is Total station?* URL: https://www.civilsimplified.com/resources/what-is-total-station (acedido em 15/05/2022) (ver p. 6).

[49] spqr. *"Autodetecting Ground Control Points"Last accessed 15 July 2021*. URL: https://community.opendronemap.org/t/autodetecting-ground-control-points/5014 (ver p. 19).

[50] J. Stoermer e P. Stoermer. *geoBits*. https://github.com/dronemapper-io/aruco-geobits (ver p. 8).

[51] *The OpenCV Reference Manual*. 2.4.13.7. OpenCV. Jul. de 2018 (ver p. 11).

[52] P. Toffanin. *OpenDroneMap: The Missing Guide*. first. UAV4GEO, 2019 (ver pp. 17, 5, 8, 13, 15, 16, 18, 19).

[53] D. Unger et al. "Accuracy of Unmanned Aerial System (Drone) Height Measurements," em: *International Journal of Geospatial and Environmental Research* 5.1 (2018), p. 13. URL: https://dc.uwm.edu/ijger/vol5/iss1/6 (ver p. 24).

[54] E. Van Rees. *OpenDroneMap: a Toolkit for Processing Aerial Drone Imagery*. Last accessed 3 May 2021. Dez. de 2017. URL: www.commercialuavnews.com/infrastructure/opendronemap-toolkit-processing-aerial-drone-imagery (ver p. 15).

I

Código Fonte

Listagem I.1: Script em Python da ferramenta GCP Finder.

```
1
2
3  #=================================================================
4  #       Script em Python da ferramenta GCP Finder
5  #=================================================================
6
7  import os
8  import cv2
9  import sys
10 import math
11 import json
12 import time
13 import glob
14 import shutil
15 import exiftool
16 import geopy.distance
17 from cv2 import aruco
18 from pygeodesy.sphericalNvector import LatLon
19 from Image_ import Image_
20 from GroundControlPoint import GroundControlPoint
21 from Statistics import Statistics
22
23
24 class GCPFinder:
25     found = "Ponto de Controle encontrado"
26     not_found = "Ponto de Controle NAO encontrado na imagem"
27     keywords = ["EXIF:Model", "MakerNotes:Yaw", "MakerNotes:
           CameraPitch", "XMP:RelativeAltitude", "File:ImageWidth",
28                 "File:ImageHeight", "EXIF:FocalLength", "EXIF:
                       GPSLatitude", "EXIF:GPSLongitude", "EXIF:
                       GPSLatitudeRef","EXIF:GPSLongitudeRef"]
29
```

```
30      def __init__(self):
31
32          self.save_gcp_path = os.path.dirname(os.path.abspath("
                GCP_Finder.py"))
33          self.total_images = 0
34          self.SENSOR_WIDTH = 0
35          self.lista_de_GCP_fixos = {}
36          self.images_with_gcp = []
37          self.image_list = []
38          self.missing = False
39          self.save_images = 0
40
41          if len(sys.argv) < 4:
42              print(
43                  'Usage: python GCP_Finder.py images_source_path
                        coordinates_source_path border | [OPTIONAL]
                        save_images_path')
44              sys.exit(1)
45          if len(sys.argv) == 5:
46              self.save_images = 1
47              # add '/' in the end of path if it's not present
48              self.save_images_path = sys.argv[4] + "/" if sys.argv
                    [4][-1] != "/" else sys.argv[4]
49
50          self.images_source_path = sys.argv[1] + "/" if sys.argv[1][-1]
                != "/" else sys.argv[1]
51
52          self.coordinates_source_path = sys.argv[2]
53          self.border = int(sys.argv[3])  # search gcp in (1-border)% of
                the image, remove border% border around the image
54
55      def run(self):
56
57          start = time.time()
58          self.read_gcp_file()
59
60          # upload all filenames
61          for filename in glob.glob(self.images_source_path + '*'):
62              self.image_list.append(filename)
63
64          total_images = len(self.image_list)
65
66          if total_images == 0:
67              sys.exit("Images directory is empty.")
68
69          stats = Statistics(total_images)
```

```
70
71          with exiftool.ExifTool() as et:
72              metadata = et.get_tags_batch(self.keywords, self.
                    image_list)
73
74          for i in range(0, total_images):
75
76              if len(metadata[i]) != 12 or self.check_metainfo(
77                      metadata[i]) is False:  # Its required 12 specific
                            parameters to process the gcp location
78                  self.missing = True
79
80          if not self.missing:
81
82              print("\nGPS information found!\n")
83              print("Proceding to search for images that probably have a
                    GCP in it.\n")
84
85              for i in range(0, total_images):
86
87                  current_image = self.make_image(metadata[i])  # create
                        a new instance of Class Image
88
89                  print("Current image:", current_image.get_filename())
90
91                  self.SENSOR_WIDTH = self.get_drone_info(Image_.
                        get_drone_model())
92
93                  if self.SENSOR_WIDTH == 0:
94                      sys.exit("Sensor Width is 0")
95
96                  distance = current_image.get_altitude() / math.tan(
                        current_image.get_pitch_angle() * math.pi / 180)
97                  distance = distance / 1000  # m to km
98                  self.show_info(current_image)
99                  origin = geopy.Point(current_image.get_latitude(),
                        current_image.longitude)
100                 destination = geopy.distance.GeodesicDistance(
                        kilometers=distance).destination(origin,
                        current_image.get_horizontal_angle())
101                 lat2, lon2 = destination.latitude, destination.
                        longitude
102
103                 print(self.is_gcp_nearby((lat2, lon2), current_image,
                        stats))
104                 print()
```

```
                stats.save_statistic(1, "meta")

        else:
            print("\nGPS information not found!\n")
            print("Proceding to search for GCPs in all images.\n")

        self.write_gcp_file_header()
        self.aruco_detect(stats)
        end = time.time()
        print("Elapsed time", round(end - start, 1), "s")

    def aruco_detect(self, stats):
        marker_found = 0

        # if there is metadata in the images, we search for gcps in
            the ones selected
        number_of_images = len(self.images_with_gcp)
        # if there isn't, we process every uploaded image
        if number_of_images == 0:
            number_of_images = len(self.image_list)
            stats.update_aruco(number_of_images)
            with exiftool.ExifTool() as met:
                meta = met.get_tags_batch(self.keywords, self.
                    image_list)
        else:
            stats.update_aruco(number_of_images)
            with exiftool.ExifTool() as met:
                meta = met.get_tags_batch(self.keywords, self.
                    images_with_gcp)

        for k in range(0, number_of_images):
            vec = []
            image_meta = meta[k]
            image_filename = image_meta["SourceFile"]
            state = False

            frame = cv2.imread(image_filename)
            gray_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

            # Dictionary with 16 bits markers and ids from 0 to 49
            aruco_dict = aruco.Dictionary_get(aruco.DICT_4X4_50)
            parameters = aruco.DetectorParameters_create()

            parameters.cornerRefinementMaxIterations = 20
            parameters.cornerRefinementMethod = 0
            parameters.polygonalApproxAccuracyRate = 0.1
```

```
148             parameters.cornerRefinementWinSize = 5
149             parameters.cornerRefinementMinAccuracy = 0.08
150             parameters.perspectiveRemovePixelPerCell = 4
151             parameters.maxErroneousBitsInBorderRate = 0.04
152             parameters.adaptiveThreshWinSizeStep = 2
153             parameters.adaptiveThreshWinSizeMax = 21
154             parameters.perspectiveRemoveIgnoredMarginPerCell = 0.4
155             parameters.minMarkerPerimeterRate = 0.008
156
157             corners, ids, rejectedImgPoints = aruco.detectMarkers(
                    gray_frame, aruco_dict, parameters=parameters)
158
159             if ids is not None:
160                 for j in range(len(ids)):
161                     c = corners[j][0]
162                     center_point = [c[:, 0].mean()], [c[:, 1].mean()]
163                     vec.append(center_point)
164
165                 if ([-1], [-1]) not in vec:
166                     state = self.addLine(vec, image_filename, ids)
167
168                 if state:
169                     print("Marker found!", self.image_list[k])
170                     marker_found = marker_found + 1
171                     stats.save_statistic(1, "gcp_found")
172                     if self.save_images == 1:
173                         self.save_images_to_folder(image_filename)
174             else:
175                 print("Marker not found in image", self.image_list[k])
176
177             stats.save_statistic(1, "aruco")
178
179         print("\nFound", marker_found, "markers out of", stats.
                get_total_images(), "images uploaded.")
180
181     def addLine(self, pixels, filename_, gcp_ids):
182         sucess = False
183         im = os.path.split(filename_)
184         img_name = im[-1]
185         s = 0
186         for m in gcp_ids:
187             n = m[0]
188             try:
189                 gcp = self.get_gcp_info(n)
190                 # longitude, latitude, altitude, imagem_pixel_X,
                        image_pixel_Y, image_name, gcp id
```

```
191                 line = str(gcp.get_long()) + " " + str(gcp.get_lat())
                        + " " + str(gcp.get_alt()) + " " + str(
192                      pixels[s][0][0]) + \
193                         " " + str(pixels[s][1][0]) + " " + img_name + "
                                " + str(n) + "\n"
194
195                 gcp_file_location = self.save_gcp_path + "/gcp_list.
                        txt"
196                 f = open(gcp_file_location, 'a')
197                 f.write(line)
198                 f.close()
199
200                 s += 1
201                 sucess = True
202             except KeyError:
203                 print("Incorrect reading. Do not print." + " False
                        identification with ID ->", n, "in " + img_name)
204             return sucess
205
206     def check_metainfo(self, metainfo):
207         correct_meta = True
208         for word in self.keywords:
209             if word not in metainfo:
210                 correct_meta = False
211
212         return correct_meta
213
214     def save_images_to_folder(self, image_path):
215         # guardar imagem numa pasta a parte
216         img_ = os.path.split(image_path)
217         img_name, img_extension = img_[-1].split('.')
218         os.makedirs(os.path.dirname(self.save_images_path), exist_ok=
                True)
219         shutil.copy(image_path, self.save_images_path + img_name + "."
                + img_extension)
220
221     def write_gcp_file_header(self):
222         gcp_file_location = self.save_gcp_path + "/gcp_list.txt"
223         f = open(gcp_file_location, 'w+')
224         f.write(self.lista_de_GCP_fixos.get(next(iter(self.
                lista_de_GCP_fixos))).get_format_())
225         f.close()
226
227     def read_gcp_file(self):
228
229         try:
```

```
230             f = open(self.coordinates_source_path, 'r')
231         except OSError:
232             print("Could not open/read file:", self.
                    coordinates_source_path)
233             sys.exit()
234 
235         header = f.readline()
236         for ln in f:
237             line = ln.split()
238             if len(line) > 0:  # read in format ID Lat Long Alt
239                 gcp = GroundControlPoint(int(line[0]), float(line[1]),
                            float(line[2]),float(line[3]), header)
240                 self.lista_de_GCP_fixos[gcp.get_id()] = gcp
241 
242     @staticmethod
243     def get_border_scale(b):
244         if type(b) not in [int, float]:
245             raise TypeError("The border percentage has to be an int or
                        float")
246         if b < 0:
247             raise TypeError("The border percentage has to be positive"
                    )
248         if b >= 100:
249             raise TypeError("The border percentage cannot be equal or
                    greater than 100%")
250         elif b == 0:
251             return 1
252         img_without_border = abs((b / 100) - 1)
253         scale_raw = math.sqrt(img_without_border)
254         scale = (abs(scale_raw - 1)) / 2
255         return scale
256 
257     def get_distance_to_corners(self, image):
258         border_estimated = self.get_border_scale(self.border)
259 
260         ground_sample_distance = (image.get_altitude() * self.
                SENSOR_WIDTH) / (
261         image.get_focal_length() * image.get_image_width())  # m/pixel
262 
263         bor = (image.get_image_width() * border_estimated), image.
                get_image_height() * border_estimated
264 
265         center_point = image.get_image_width() / 2, image.
                get_image_height() / 2
266 
```

```
267         dist = math.sqrt((center_point[0] - bor[0]) ** 2 + (
                center_point[1] - bor[1]) ** 2)  # distance in pixels
268 
269         final_distance = dist * ground_sample_distance  # real
                distance in meters
270 
271         return final_distance
272 
273     @staticmethod
274     def get_drone_info(model):
275         f = open('drones_DB.json')
276         data = json.load(f)
277 
278         return data[model]
279 
280     def show_info(self, image):
281 
282         gsdW = (image.get_altitude() * self.SENSOR_WIDTH) / (
283                 image.get_focal_length() * image.get_image_width())  #
                        m/pixel
284 
285         print("Altitude:", image.get_altitude(), "m")
286         print("Sensor width:", self.SENSOR_WIDTH, "m")
287         print("Focal lenght:", image.get_focal_length(), "m")
288         print("Image width:", image.get_image_width(), "px")
289         print("Ground Sample Distance:", round(gsdW * 100, 5), "cm/
                pixel")
290 
291     @staticmethod
292     def make_image(meta):
293         pitch_angle = abs(meta["MakerNotes:CameraPitch"])  #
                pitch_angle has to be positive
294         image_width = meta["File:ImageWidth"]
295         image_height = meta["File:ImageHeight"]
296         focal_length = meta["EXIF:FocalLength"] / 1000  # mm to m
297         horizontal_angle = float(meta["MakerNotes:Yaw"])  # 0 yaw e
                calculado em sentido clockwise
298         altitude = float(meta["XMP:RelativeAltitude"][1:-1])  # raw
                altitude value
299         filename = meta["SourceFile"]
300         model = meta["EXIF:Model"]
301         lati = meta['EXIF:GPSLatitude']
302         lon = meta['EXIF:GPSLongitude']
303         latRef = meta['EXIF:GPSLatitudeRef']
304         longRef = meta['EXIF:GPSLongitudeRef']
305 
```

```
306         # avoid division by 0 -> if pitch_angle == 0, drone is looking
                to horizon
307         if pitch_angle == 0:
308             pitch_angle = 0.000001
309 
310         if latRef == "S":
311             lati = -lati
312         elif longRef == "W":
313             lon = -lon
314 
315         return Image_(pitch_angle, image_width, image_height,
               focal_length, horizontal_angle, altitude, filename,
316                       model,
317                       lati, lon)
318 
319     def get_gcp_info(self, id__):
320         return self.lista_de_GCP_fixos[id__]
321 
322     # receive the gps coordinates of the focal point in the image and
           the image itself
323     def is_gcp_nearby(self, centerCoord, img, stats):
324         top_right, bottom_right, bottom_left, top_left = self.
               get_corner_coordinates(img, centerCoord[0],centerCoord[1])
325 
326         image_path = img.get_filename()
327         find = False
328         # assuming list GCP already in DD format
329         for gcp in self.lista_de_GCP_fixos:
330 
331             p = LatLon(self.lista_de_GCP_fixos[gcp].get_lat(), self.
                    lista_de_GCP_fixos[gcp].get_long())
332             b = LatLon(top_right.latitude, top_right.longitude),
                    LatLon(bottom_right.latitude, bottom_right.longitude),
333                     LatLon(bottom_left.latitude, bottom_left.longitude),
                            LatLon(top_left.latitude, top_left.longitude)
334 
335             if p.isenclosedBy(b):
336                 find = True
337                 stats.save_statistic(1, "contains_gcp")
338                 if image_path not in self.images_with_gcp:
339                     self.images_with_gcp.append(image_path)
340 
341         if find:
342             return self.found
343         else:
344             return self.not_found
```

```
345
346     def get_corner_coordinates(self, img, centerLat, centerLong):
347
348         # angle between the top and the right cornor of the image (it
                will be 45 degrees if the image its a square)
349         angle = math.atan((img.get_image_width() / 2) / (img.
                get_image_height() / 2)) * (180.0 / math.pi)
350
351         NE = angle  # starting from North which is 0
352         SE = 180 - angle
353         SW = 180 + angle
354         NW = 360 - angle
355
356         dist = self.get_distance_to_corners(img)
357
358         dist = dist / 1000  # in km
359
360         center_point_coord = geopy.Point(centerLat, centerLong)
361
362         top_right = geopy.distance.GeodesicDistance(kilometers=dist).
                destination(center_point_coord,img.get_horizontal_angle() +
                 NE)
363         bottom_right = geopy.distance.GeodesicDistance(kilometers=dist
                ).destination(center_point_coord,img.get_horizontal_angle()
                 + SE)
364         bottom_left = geopy.distance.GeodesicDistance(kilometers=dist)
                .destination(center_point_coord,img.get_horizontal_angle()
                + SW)
365         top_left = geopy.distance.GeodesicDistance(kilometers=dist).
                destination(center_point_coord,img.get_horizontal_angle() +
                 NW)
366
367         return top_right, bottom_right, bottom_left, top_left
368
369
370 if __name__ == '__main__':
371     finder = GCPFinder()
372     finder.run()
373
374 #=========================================================================
```