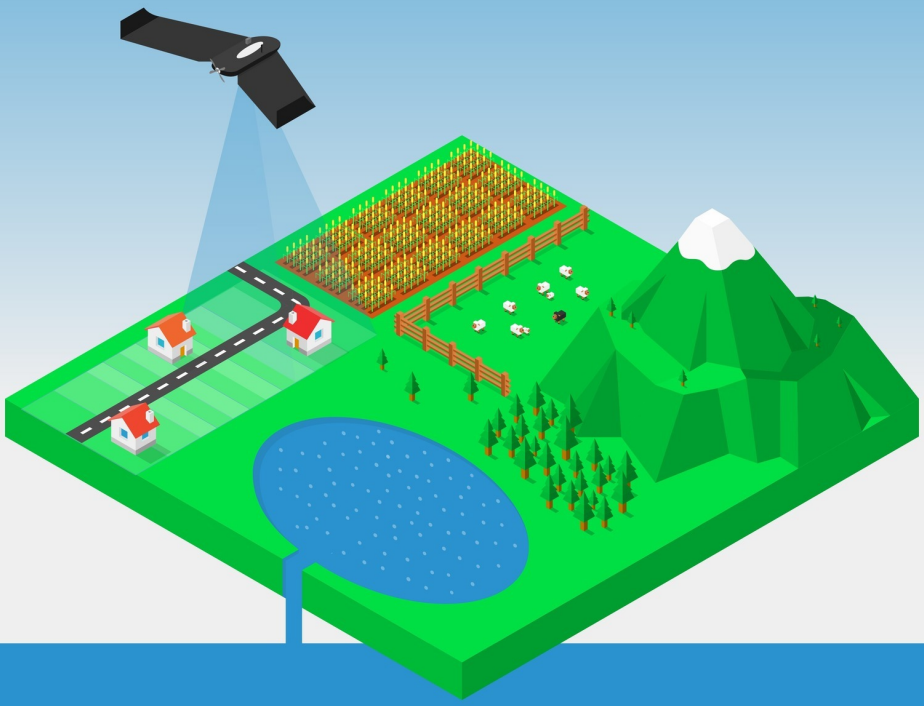
A Practical Guide to Drone Mapping
Using Free and Open Source Software



# OpenDroneMap
## The Missing Guide

Piero Toffanin

# Contents

*Preface* vii
*Acknowledgement* x
*Gold Supporters* x
*Silver Supporters* xi

I Introduction

Why OpenDroneMap? 3
What You Can Do with OpenDroneMap 5
The Key To Becoming a Successful User 8

II Getting Started

Installing The Software 13
Hardware Requirements 15
Installing on Windows 16
Installing on macOS 26
Installing on Linux 29
Basic Commands and Troubleshooting 32
Hello, WebODM! 34
Processing Datasets 36
Dataset Size 36
File Requirements 37
Process Tasks 38

Output Results 42
Share With Others 43
Export To Another WebODM 44
Manage Plugins 44
Change The Look & Feel 44
Create New Users 44
Manage Permissions 44
How Does WebODM Process Images? 45
The Processing Pipeline 46
Load Dataset 47
Structure From Motion 47
Multi View Stereo 51
Meshing 52
Texturing 54
Georeferencing 56
Digital Elevation Model Processing 57
Orthophoto Processing 58
Task Options in Depth 61
build-overviews 64
cameras 64
crop 65
debug 66
dem-decimation 66
dem-euclidean-map 67
dem-gapfill-steps 68
dem-resolution 70
depthmap-resolution 71
dsm 72
dtm 72
end-with 73
fast-orthophoto 74

gcp 77
help 77
ignore-gsd 77
matcher-distance 79
matcher-neighbors 80
max-concurrency 81
merge 82
mesh-octree-depth 82
mesh-point-weight 85
mesh-samples 87
mesh-size 89
min-num-features 89
mve-confidence 92
opensfm-depthmap-method 94
opensfm-depthmap-min-patch-sd 94
orthophoto-bigtiff 98
orthophoto-compression 98
orthophoto-cutline 99
orthophoto-no-tiled 101
orthophoto-resolution 102
pc-classify 102
pc-csv 109
pc-ept 109
pc-filter 109
pc-las 110
rerun 111
rerun-all 111
rerun-from 111
resize-to 112
skip-3dmodel 112
sm-cluster 114

smrf-scalar 114
smrf-slope 114
smrf-threshold 114
smrf-window 114
split 114
split-overlap 115
texturing-data-term 115
texturing-keep-unseen-faces 122
texturing-nadir-weight 124
texturing-outlier-removal-type 127
texturing-skip-global-seam-leveling 130
texturing-skip-hole-filling 132
texturing-skip-local-seam-leveling 132
texturing-skip-visibility-test 135
texturing-tone-mapping 135
time 136
use-3dmesh 136
use-exif 137
use-fixed-camera-params 137
use-hybrid-bundle-adjustment 138
use-opensfm-dense 140
verbose 140
version 140
Ground Control Points 141
Creating a GCP file using POSM GCPi 145
Using GCP files 150
How GCP files work 150
Flying Tips 152
Fly Higher 152
Fly on Overcast Days 153
Fly Between 10am and 2pm 153

Fly at Different Elevations and Capture Multiple Angles 153
Fly on Calm Days 154
Increase Overlap 155
Set Drone to Hover While Taking Images 155
Check Camera Settings 156

III Advanced Usages

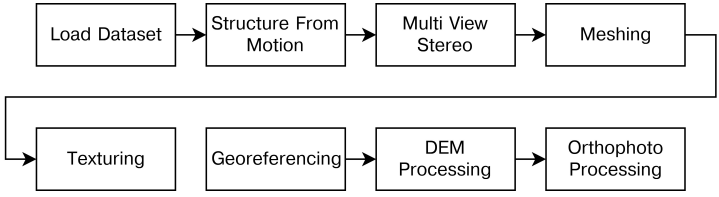
The Command Line 159
Command Line Basics 160
Using ODM 162
Processed Files Owned By Root 163
Add New Processing Nodes to WebODM 164
Batch Geotagging of Images Using Exiftool 165
Further Readings 166
Docker Essentials 167
Docker Basics 167
Managing Containers 169
Managing Images 172
Managing Volumes 174
Docker-Compose Basics 177
Managing Disk Space 179
Changing Entrypoint 180
Assigning Names To Containers 180
Jumping Into Existing Containers 181
Making Changes Without Rebuilding Images 182
Camera Calibration 184
Option 1: Use an Existing Camera Model 186
Option 2: Generate a Camera Model From a Calibration Target 188
Taking Pictures of a Calibration Target 189

Extracting a Camera Profile 190
Manually Writing a cameras.json File 193
Bonus: Checking Your LCP File by Manually Removing Geometric Distortion 196
Processing Large Datasets 200
Split-Merge Options 201
Local Split-Merge 204
Distributed Split-Merge 206
Using Image Groups and GCPs 210
Limitations 212
The NodeODM API 213
Launching a NodeODM Instance 215
NodeODM Configuration 216
Using the API with cURL 219
Remove a Task 221
API Specification 222
Automated Processing With Python 242
Getting Started 243
Example 1: Hello NodeODM 244
Example 2: Process Datasets 245
Concluding Remarks 248
API Reference 248
*Glossary* 257
*About the Author* 261

# 6

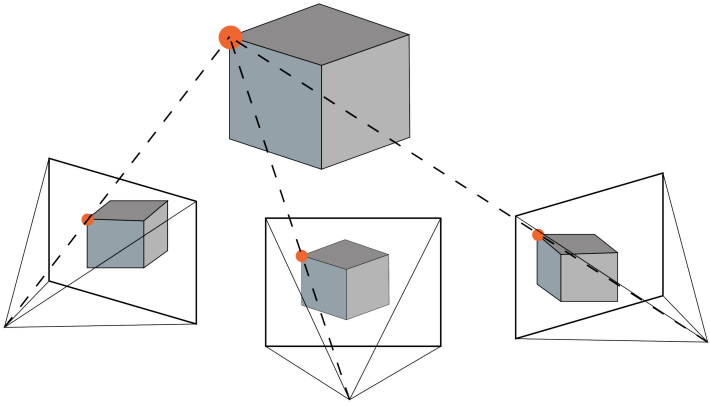
# The Processing Pipeline

Going from images to 3D models and orthophotos is a process best visualized as a series of incremental steps. Each step relies on the work of previous steps.




*ODM's processing pipeline*

In this chapter we will explore an overview of the pipeline. We will not cover too many details, as each step's behavior can be tweaked by changing the task options. We will discuss in detail of how task options affect the inner workings of each step in the next chapter.



*The SFM problem. What kind of camera took these pictures and where was the camera when the pictures were taken?*

Photogrammetry is not a new field and its history dates back hundreds of years[14]. It's just that we've recently discovered that computers can be really good (and fast) at it. For those interested in learning more about SFM, coursera.org has some really good lectures[15]. ODM uses a software package called OpenSfM[16] (Open Structure From Motion) for efficiently solving the SFM problem.

**Input**: images + GCP (optional)
**Output**: camera poses + sparse point cloud + transform

[14] History of Photogrammetry: http://wayback.archive-it.org/all/20090227061949/http://www.ferris.edu/faculty/burtchr/sure340/notes/History.pdf

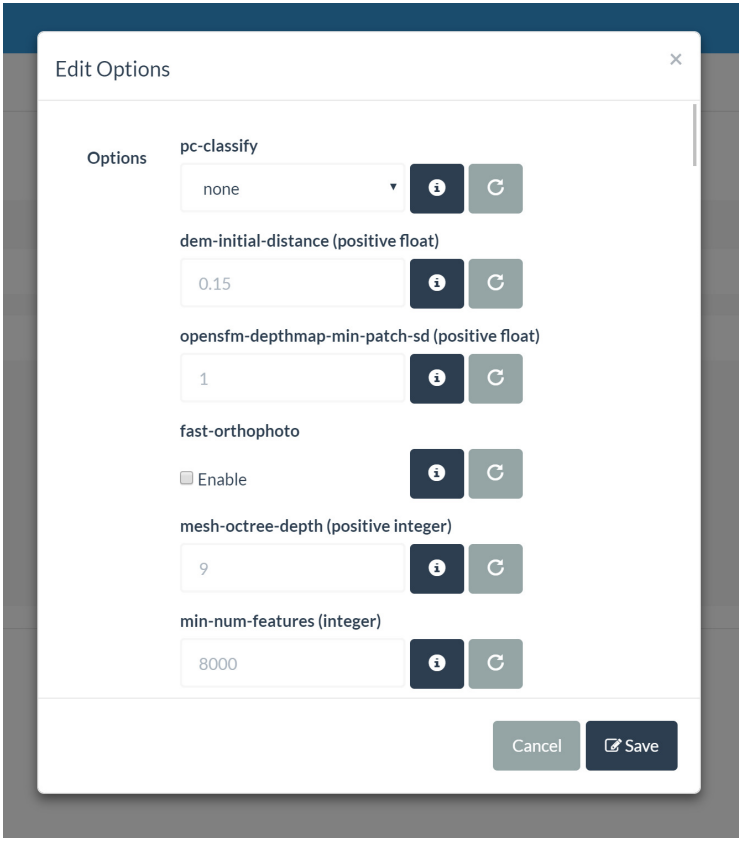
[15] Robotics: Perception: https://www.coursera.org/learn/robotics-perception

[16] OpenSfM: https://github.com/mapillary/OpenSfM/

# 7

# Task Options in Depth

There are several components involved in the data processing pipeline. Each component has several adjustable settings that influence the output. The software exposes a subset of these available knobs through various options. When creating a task, a user can choose to tweak one or more options to change the behavior of the pipeline.




*Options as shown in WebODM when creating a task*

If the list seems overwhelming, just remember that this is a subset of all possible options that could be available from the various components of the data pipeline! This should raise some curiosity. Hidden features and processing capabilities could be hiding in the source code of OpenDroneMap, in the form of an option not yet exposed! The software exposes only those options that seem to have the biggest impact on results, or

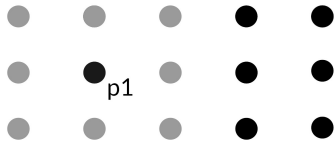
those necessary to handle different workflows. But many, many more options, under the hood, remain unexposed in order to keep their number somewhat manageable.

Tuning options is more art than science. There are no clear guidelines on how to tune options to achieve optimal results. That's mostly because the best options for a certain dataset do not automatically transfer over to another. Given the big variety of possible scenes, cameras and mission planning strategies, it would be immensely time consuming to write an exhaustive guide. Plus, the tools are evolving quickly, so by the time such guide would be written, it would already be obsolete. But fear not!

This chapter is about understanding in detail what each option does. By the end of the chapter you'll be able to quickly improve your results, explain why certain models turn out the way they do and know what to tweak if the results don't turn out the way you want.

A few of these options might be missing from the graphical interface and might be available only from the command line. This is because sometimes the option does not make sense in the context of the interface workflow, or it's simply not supported.

Feel free to jump around and use this chapter as a reference. As the software gets better, some of these options might disappear from future versions and new ones will be introduced. The list below is taken from the software as of June 26th 2019. I will try my best to keep up with updates to the software and I plan to start a second edition of the book after it's published. In alphabetical order:




*Dots represent approximate image locations, extracted from EXIF tags. When the matcher-neighbors is set to 8, only the 8 nearest neighbors (highlighted in gray) are considered for matching with image p1*

For datasets with lots of overlap, it can be beneficial to increase this value since it's likely that valid matches will not be taken into consideration and decrease the accuracy of the reconstruction. It can be set to zero to disable it. If no location information is embedded in the EXIF tags of the images, this option is disabled. This option works in conjunction with the **matcher-distance** option.

## max-concurrency

By default the program will attempt to use all available CPU resources. There are scenarios where this might not be desirable, for example on shared servers or when wanting to use the computer for other tasks while processing. This option attempts (but does not guarantee) to limit the maximum number of CPU cores that will be used at the same time.

## merge

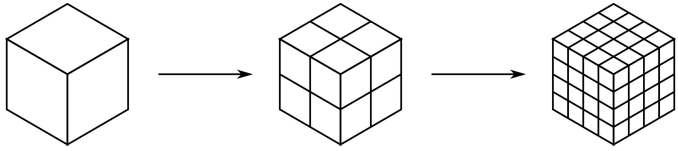
This option controls what assets should be merged during the merge step of the split-merge pipeline. By default all available assets are merged, but users can choose to merge only specific ones. We cover this option in more detail in the *Processing Large Datasets* chapter.

## mesh-octree-depth

When it comes to generating 3D models, this is probably the most important option. It specifies a key variable for the Screened Poisson Reconstruction[25] algorithm, which is responsible for generating a mesh from the point cloud. The details of the algorithm are fascinating, but probably outside the scope of this chapter. For the curious ones, the best description I could find is available on Wikipedia under the *Surface Reconstruction* section at https://en.wikipedia.org/wiki/Poisson%27s_equation

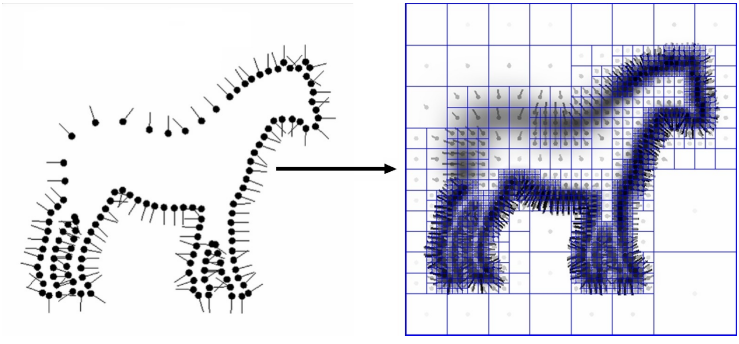
To understand how this option affects the output, it helps to visually understand the concept of an octree. First, octree means *eight-tree* (okta is *eight* in Greek). Why eight? Because at each level (or *depth*) of the tree, each box (or *node or branch*) of the tree is divided in eight parts. At the first level there's only one branch. At the second level there's 8. At the third there's 64 and so forth.

[25] Screened Poisson Reconstruction: watertight surfaces from oriented point sets. http://www.cs.jhu.edu/~misha/MyPapers/ToG13.pdf



*An octree with depth 1, 2 and 3*

Lower depths in an octree allow finer details to be captured.



*Points and resulting octree.*
*Image from http://www.cs.jhu.edu/~misha/Code/*

The practical aspect of this option is that the higher the value, the finer the resulting mesh will be. The trade-off is exponentially longer run-time and memory usage. The default value of 9 works well for a lot of different cases. Flat areas can benefit from lower values (6-8) and urban areas can improve by setting this value higher (10-12). When increasing this option, **mesh-size** should also be increased as finer meshes require more triangles.

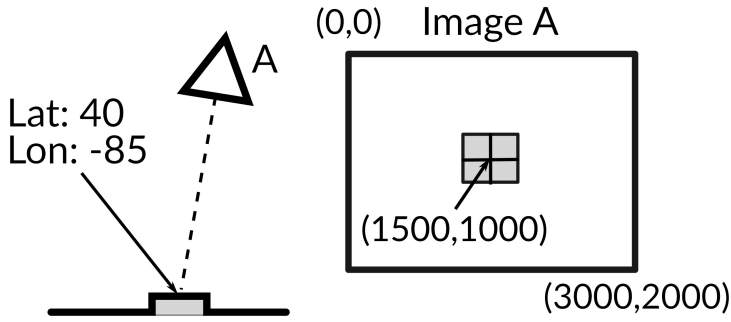



*Figure 1: A GCP marker is photographed by camera A to produce Image A. In a second step, the pixel location of the marker (1500,1000) from Image A can be manually tagged with its real world coordinates (latitude 40, longitude -85).*

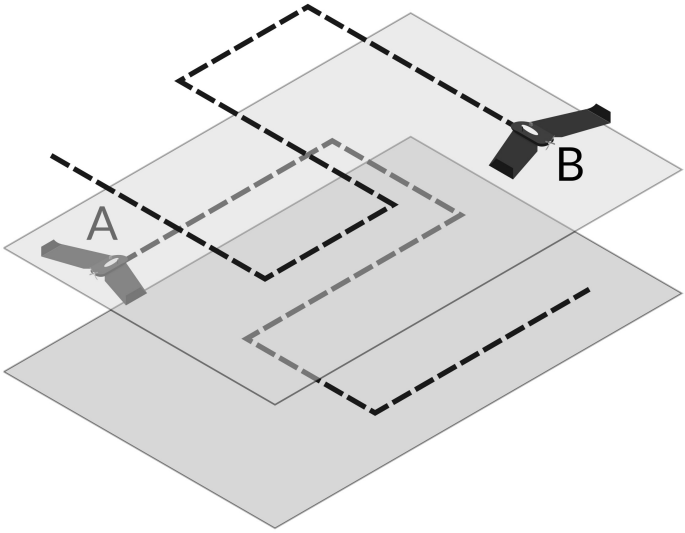
Using ground control points can increase the georeferencing accuracy of a reconstruction, since measurements of static (non-moving) objects using a high precision GPS are often better than those obtained from the GPS of moving UAVs.

The ideal number of ground control points ranges between 5 to 8, placed evenly across the area to be flown. Adding more than 8 ground control points does not necessarily result in increased accuracy.

If the same marker is visible from multiple images, it should be tagged multiple times for each image. Ideally each marker should be tagged at least 3 times. Another way to think of it is to capture each marker on at least 3 images. This is so that the marker's location can be triangulated during computation.

Ground control points can be used by providing an additional text file along with the input images. The file follows a simple format:

- The first line indicates the spatial reference system (SRS) of



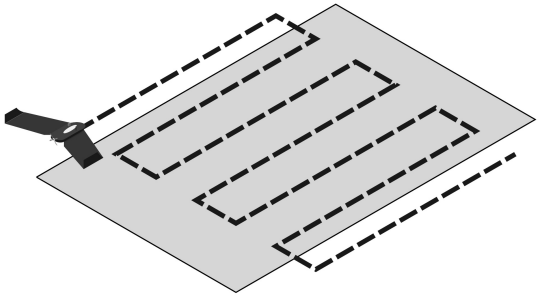

*Flying cross-pattern at different elevations with B capturing nadir images and A capturing images at a slight angle (or vice-versa) is a better for camera calibration*

We discuss this in more details in the *Camera Calibration* chapter.

## Fly on Calm Days

When it's windy your drone will have a harder time stabilizing the camera and can produce images that are blurrier. Fly when it's calm for better results.

High overlap, near nadir images are not recommended[48]. This is important to remember, as most mission planning software will create exactly this type of pattern.

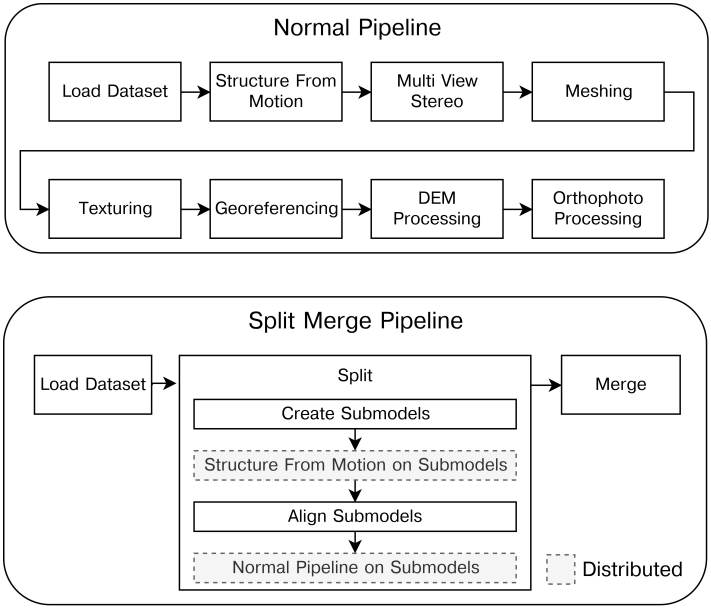


*Typical flight path from mission planning software. Not great for self-calibration*

This doesn't mean a person should never fly this pattern. It just means that when flying this pattern, people need to be aware that the reconstruction will not be as accurate. Over large areas, this error accumulates and typically results in a *doming* effect.

*Point cloud exhibiting doming. The terrain appears arched instead of straight*

[48] Camera Calibration Considerations for UAV Photogrammetry: https://www.isprs.org/tc2-symposium2018/images/ISPRS-Invited-Fraser.pdf



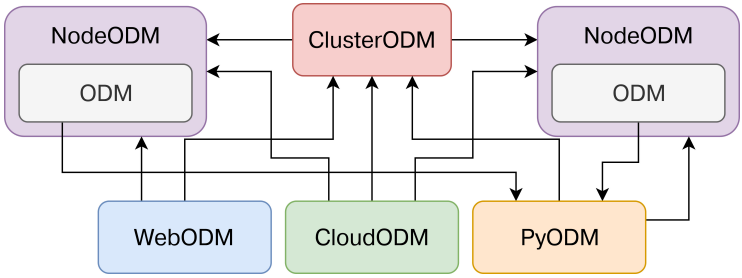

*Split-merge pipeline. Step 2 and 4 of the split section can be performed in parallel on separate machines when using distributed split-merge*

## Split-Merge Options

### split

Split-merge can be used either from the command line (ODM) or from WebODM and is turned on/off by setting the **split** option. Split-merge will be turned on anytime the following condition is true:

```
Split Option < Number of Images
```




*OpenDroneMap projects use the NodeODM API to communicate with each other*

It's interesting to note the NodeODM API has also been adopted by projects outside the OpenDroneMap ecosystem. For example, the NodeMICMAC project[55] has successfully implemented an aerial image processing pipeline using the open source MicMac photogrammetry engine as an alternative to ODM. This move has allowed the project to re-use and work in sync with all of the other tools within the OpenDroneMap ecosystem.

In this chapter we'll learn to manually launch a NodeODM instance, explore its web interface and do some manual interaction with the API using cURL, a program that allows us to make web requests. Familiarity with the NodeODM API can give users a better understanding of the network interactions that happen between the various projects.

The API has different versions and tries to be backward compatible with previous implementations whenever a new version is released. The most up-to-date specification is available online[56], while a copy of the latest specification at

[55] NodeMICMAC: https://github.com/dronemapper-io/NodeMICMAC/

[56] NodeODM specification: https://github.com/OpenDroneMap/NodeODM/blob/master/docs/index.adoc

multiple overlapping image pairs. MVS programs expect that information about cameras has already been computed.

**NodeODM:** A lightweight REST API to access aerial image processing engines such as ODM or MicMac

**Noise:** An unwanted interference. When applied to point clouds, it indicates points that should not be present or that were missed during outlier filtering

**Nadir**: The direction pointing directly below a particular location

**ODM:** A command line toolkit to generate maps, point clouds, 3D models and DEMs from drone, balloon or kite images.

**OpenSfM:** Open source structure from Motion library written in Python. The library serves as a processing pipeline for reconstructing camera poses and 3D scenes from multiple images

**Orthophoto**: An image that has been *orthorectified*, warped in such a way that distances and scales are uniform

**Photogrammetry**: the process of obtaining reliable information about physical objects and the environment through the process of using photographic images.

**Preemptive Matching:** During the Structure From Motion process, the act of reducing the possible number of pair candidates by using the location information stored in the EXIF tags of the images

**PyODM**: A Python SDK for adding aerial image processing capabilities to applications

**RTK:** Real Time Kinematics. A satellite navigation technique used to enhance the precision of position data derived from satellite-based positioning systems

**SDK:** Software Development Kit. A set of libraries, tools and examples that help software developers to build software with