École Polytechnique Fédérale de Lausanne

Automated georegistration of 3D models

Master Semester Project at

Digital Humanities Laboratory

Author: Alicia Soria Gómez

Supervisor: Albane Descombes

Contents

Chapter 1

Introduction

1.1 Context

Nowadays, there are numerous applications being developed in the paradigm of computer vision, such as autonomous navigation, monitoring, facial recognition,inventory management, medical imaging, online education, interactive entertainment, image based 3D reconstruction, virtual and augmented reality apps, etc. The list goes on, but all of them share a basic requirement in order to perform correctly. This necessity is to develop and perform an accurate geolocalization of the 2D-3D points involved in the service, so that actions are taken in the correct position, and mapping is precise.

In this context, photogrammetry appears. Photogrammetry can be understood as the opposite of photography, while photography consists on projecting a 3D scene into one or more 2D projections, photogrammetry aims to gather a set of unordered 2D images and transform them into a 3D model.

There are multiple programs and commands that enable the user to perform the transformation of 2D images into 3D models in a simple manner, such as AliceVision Meshroom (AliceVision 2021) which firstly employs the Structure From Motion flow to compute representative descriptors in several images, then calculates the camera movement through the set of images and finally computes a 3D cloud of points; and Multi View Stereo process, that densifies the cloud, calculates a surface mesh over the cloud and finally texturizes the mesh.

Some other examples of softwares specialized in photogrammetry pipelines are COLMAP, MicMac, 3DF Zephyr, DroneDeploy, Photomodeler, etc. (all3dp 2021)

Finally, the last example of a photogrammetry pipeline belongs to OpenMVG, that performs SFM pipeline, and OpenMVS which is responsible for the MVS process. In this project, OpenMVG and OpenMVS are employed.

Photogrammetry is a precise method for representing and measuring our surrounding environment, therefore, it is crucial to give the 3D models a proper scale and position in space. Otherwise, the results would be 3D models in uncorrelated spaces.

An example of an EPFL startup that works in this scope is Pix4D. They propose the Pix4Dmapper software, which allows the user to employ a drone as the mapping tool and from a mobile application, it is possible to control its flight to ensure that images are correctly overlapped for the photogrammetry process (all3dp 2021). In this process, a GPS is needed to measure exactly the position of some control points necessary to later geolocalize the 3D points of the cloud.

1.2 Related work

Generally, in order to localize a 3D model, a GPS device is employed when taking the images that will be then introduced into the photogrammetry pipeline. Some softwares such as Pix4Dmapper and Agisoft Metashape (Agisoft 2022) work with GPS coordinates that are obtained from a GPS mapping system embedded into the camera that captured the images. In the case of Pix4D it would be embedded into Parrot drones. (PIX4D 2022)

However, it is not always possible to obtain GPS coordinates from a GPS device, for example when images have been obtained from historical sources, internet videos, etc. In this case, there have been various approaches to solve the geolocalization necessity when there is no geographic metadata available, that are based on computer vision algorithms.

Firstly, methods based on image-retrieval, which aim to match a reference image to the dataset of images available for the 3D reconstruction (Wang, Wilson and Snavely

2013). Secondly, there are methods that synthesize new 2D views from a 3D model, for example, an aerial view (Liu, Li and Dai 2017). Thirdly, algorithms that create a vocabulary tree filled with image descriptors that are then matched to one or more 3D points (Sattler et al. 2015). And fourthly, there are approaches based on co-visibility relations between 3D points that are likely seen from the same set of images (Liu, Li and Dai 2017). In this project, mainly the first two approaches will be studied.

1.3 Goals

The goal of this study is to develop an automatic or semi-automatic pipeline that allows the georegistration of 3D points in a cloud model. To achieve this objective, several approaches are taken and ideally the best one is presented as the final result.

Chapter 2

Methods

This chapter is devoted to explain the main algorithms and techniques chosen throughout the project. The following sections will be illustrated: firstly, the gathering of the data and its visualization, secondly, the introduction to the base tools employed in the different stages of the project that are recurrent throughout its evolution, and thirdly, the different investigation approaches that were explored to obtain the desired results.

2.1 Obtaining and visualizing the data

Throughout this project, the raw data employed is mainly of three types: 3D models, 2D images (can be hand-taken pictures, Internet retrieved images, aerial maps,...), and 2D cadaster maps.

2.1.1 3D models

Firstly, 3D models of different locations such as the Louvre museum, San Gimignano, Vatican Piazza in Figure 2.1, Colmar, etc. are provided by the DHLAB. Each of these models has been constructed employing photogrammetry techniques available in OpenMVG.

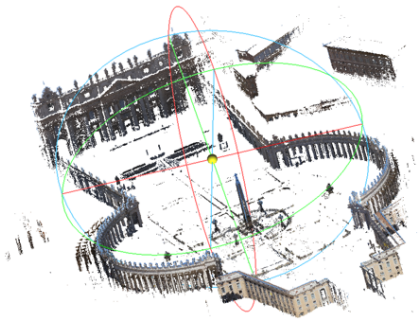


Figure 2.1: Visualization of the 3D model of the Vatican Piazza with CloudCompare

They include a folder with the images on which the 3D model is based, a 'structure from motion' file with

some intrinsic parameters related to the camera position, and some extrinsics related to the different poses captured by the camera. Finally the 3D points are stored in a Polygon File Format (.ply). In order to visualize the models, the CloudCompare software (CloudCompare 2020) is employed, and is shown in Figure 2.1.

2.1.2 2D image database

Secondly, 2D images (either afoot-taken or aerial) are obtained from various databases. These can be either provided by the DHLAB in order to create the 3D models, or either retrieved from GoogleStreetView (GoogleStreetView 2019), Google Maps (GoogleMaps 2019), Map Tiler (MapTiler 2022), Open Street Map (OpenStreetMap 2021) and from the Institut National de l'Information Géographique et Forestière (IGN) (IGN 2021).

2.1.3 2D cadaster models

Thirdly and finally, the 2D cadaster maps are retrieved from various webpages which provide more or less details on their representation. In particular, the preferred sources are Geofabrik (Geofabrik 2020), for not-too-detailed maps, or Open Street Map, Map Tiler and Paris OpenData (OpenData 2021) for more accurate maps. For the visualization of the 'shapefiles' (.shp) that contain the cadasters, the software Qgis is employed. Qgis, as its name indicates, supports Geographic Information System processing, and allows the user to manage geographic information, monitor routes, change map properties, locate markers, and specially, raster other maps on top of a cadaster model.

2.2 Base tools

Each of the research steps in this project employs different tools, however, there are several that are transversal to the various stages. Specifically, I will introduce OpenCV (OpenCV 2021), its particularization into OpenMVG commands and the most important Python libraries that are recurrent in this study.

2.2.1 OpenCV

The first software, OpenCV, is a collection of packages, libraries and in general tools intended to solve computer vision tasks. Among its many features, in this project the ones employed were those that enable to find visual features in images and to match two images, finding the common features in both images. There have been several options explored to find features in images, such as the Harris Corner Detection, that extracts corners and infers features from an image; the SIFT [1] finding algorithm, which finds descriptors regardless of size and orientation by calculating histograms of gradients in the pixels; the BLOB [2] algorithm to find contours with some special characteristics such as centroid, color, mean and standard deviation of the surrounding pixels and finally the ORB [3] method, that combines the strengths of several other methods with many modifications to enhance performance, this one permitted to find the greater number of descriptors per image in comparison with the other algorithms. An example of the ORB descriptors found in San Gimignano (Italy) aerial view can be seen in Figure 2.2. Specially, the ORB algorithm has been employed to match two images and find the common descriptors in both of them. An advantage is that it can be easily imported into a Jupyter Notebook as the library `cv2`.

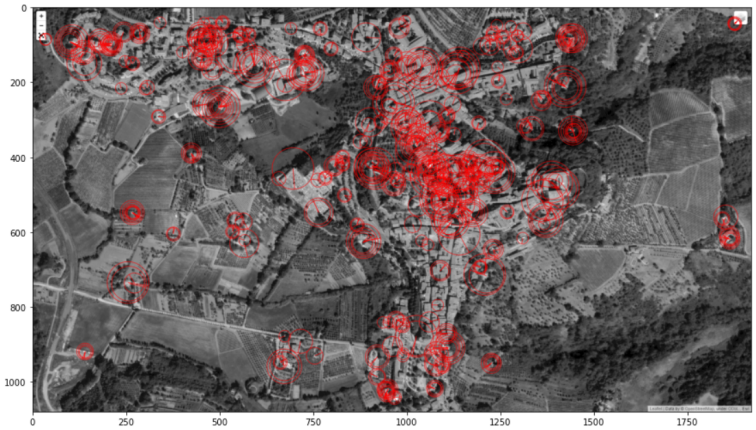


Figure 2.2: ORB descriptors in San Gimignano

[1] Structural Invariant Feature Transform
[2] Binary Large Object
[3] Oriented FAST and Rotated BRIEF

2.2.2 OpenMVG

A particularization of OpenCV is OpenMVG (Multi View Geometry), that provides a pipeline applicable to a collection of images. Firstly, it computes the 'sfm' file with the camera movement parameters; secondly, it calculates the SIFT descriptors in each image and exports a file with the points as well as a visualization of them; thirdly, it matches all of the images between them to extract the closest matches, and produces a summary graph with the closest images that share the greatest amount of SIFT descriptors; and finally, it is possible to build a 3D model from this structure from motion pipeline, which can be visualized in CloudCompare for instance, as well as the camera position with respect to the 3D model. An example of the 3D model of half of the Bâtiment Vortex (a FMEL residence in Lausanne) is shown in Figure 2.3. Hence, this software is particularly interesting for 3D model computation, matching between a set of images, and extracting camera movement parameters.

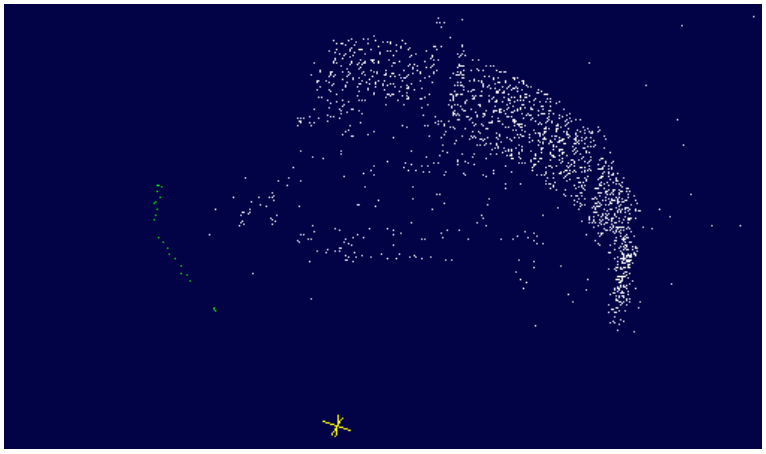


Figure 2.3: SFM 3D result from Bâtiment Vortex

2.2.3 Python scripts

And lastly, another software that has been employed repeatedly is the Jupyter Notebooks with Python libraries to automatize the OpenCV and OpenMVG pipelines, to access and download images and maps from different sources, and to analyze different results and visualize them interactively. The main libraries employed are the following: `ee` from Google Earth Engine to authenticate a Google user, `folium` to obtain the Google Earth Engine aerial maps, `cv2` to implement the OpenCV pipeline,

`google_streetview.api` and `.helpers` to download afoot-taken images from Google Streetview, `geopy.geocoders` to transform a location in string format into coordinates, `streetview_functions` to download panoramic images from piazzas for instance, `defisheye` in order to remove the fish eye distorsion, and other for analysis, visualization and interactiveness such as `numpy`, `plotly.express` and `matplotlib`.

The following table provides a reference link to the main libraries employed throughout this project.

Library	Reference
`ee`	https://developers.google.com/earth-engine/
`folium`	https://python-visualization.github.io/folium/
`cv2`	https://pypi.org/project/opencv-python/
`google_streetview`	https://pypi.org/project/google-streetview/
`geopy`	https://geopy.readthedocs.io/en/stable/
`streetview_functions`	https://github.com/robolyst/streetview
`defisheye`	https://pypi.org/project/defisheye/
`numpy`	https://numpy.org/
`plotly.express`	https://plotly.com/python/plotly-express/
`matplotlib`	https://matplotlib.org/

Chapter 3

Results

During the evolution of the project, there were three main approaches considered, which concluded in different results that were more or less satisfactory. The first one consists of afoot-matching, which refers to employing the images created to construct the 3D model and match them to a reference image which is supposed to be geolocated, in this case it can be seen that the 3D model is not directly employed.

The second approach consists in actually employing the 3D model visualized in CloudCompare to find matches with a reference image. This involved creating a dense mesh over the 3D points and investigating the intrinsic/extrinsic parameters in the 'sfm' file. However, in this case, the most straightforward way to geolocalize points was by employing the aerial visualization of the 3D model, hence the third approach.

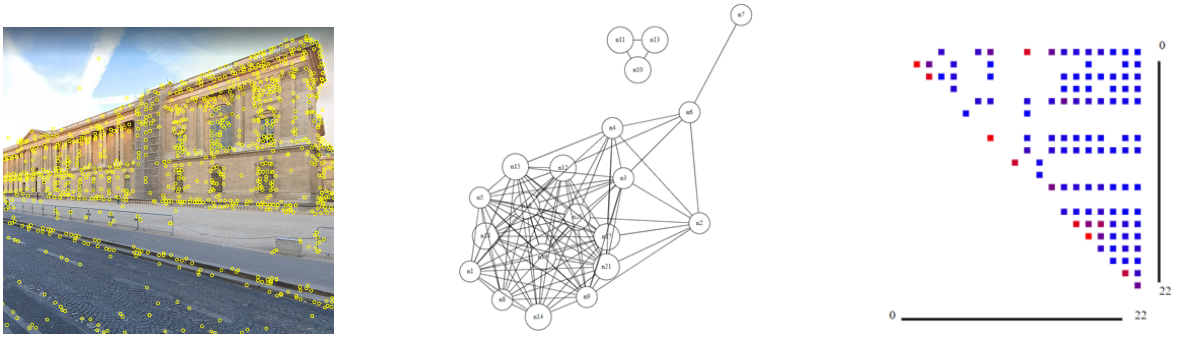
The third approach consists of capturing the 3D aerial visualization as an image and studying the possibilities of aerial matching. Here, the 3D aerial view is geolocalized by employing reference maps from open source repositories such as Google Maps with the `folium` package.

3.1 Afoot matching

The first approach consists in having a query database of images from a given place, for instance, the Louvre Museum, which are not exactly localized, but roughly. The idea is to introduce into the database a large enough number of reference images taken

from an open source site, for example, GoogleStreetView, that are localized, as we know the orientation of the camera and its exact position, which is given by the site. The objective is to find the closest query image for each reference image, and in the end, be able to localize all of the images in the query database. The pipeline is the following:

$$Get\ query\ database > Get\ reference\ image > Find\ closest\ match > Automatize$$



(a) SIFT descriptors (b) Matches as graph (c) Matches as matrix

Figure 3.1: OpenMVG outputs

As a query database, a given set of Louvre images from the DHLAB was employed; and as a reference, several images downloaded from GoogleStreetView were tested individually. For the matching process, the software OpenMVG was used which outputs representation of SIFT descriptors over the analyzed images and computes matches between all of the images, as it can be seen in Figure 3.1. Later, for the automatization process, a Jupyter Notebook was written in which `geopy.geocoders` is employed to transform an address into coordinates, `google_streetview` is applied to download a set of images close to the given coordinates, by slightly modifying the heading (north, south, etc, orientation), field of view (essentially is the zoom) and pitch (up or down), as well as the exact location of the coordinates. Finally, the Linux commands defined to call OpenMVG functionalities are also introduced in the Python Notebook for efficiency purposes, by calling `subprocess.run()`, and the specific command.

The results in this approach are not great, nevertheless, as it is an arduous job to find a reference image that is similar to the queries in the database in an automatic manner simply by inputting the rough address of the building due to inaccuracies in every step. Firstly, the `geopy` library returns a set of coordinates that can be either inside

the building or in any location, and then the retrieval of enough images requires more parameters, for instance, take that the coordinates are of the center of the building, then it would be necessary to know the exact width of the building to make a complete tour around it and find similar images to that of the query database. Therefore, a more specific method was designed which only tackles 'piazzas' or squares.

3.1.1 Panoramic exploration

In this case, the approach was to take a panoramic image of the interior of the square with the Python function `streetview_functions.download_panorama_v3` which calls GoogleStreetView, and is already georeferenced. Then, this panoramic is defisheyed with the library `defisheye` and matched against the query database with OpenCV ORB function, and depending on where on the reference panoramics the descriptors were located, each query image could be oriented. This method has two main drawbacks, firstly, it is uniquely useful for round buildings or squares, and secondly, the results are quite coarse, as the matching descriptors against a panoramic image that has been defisheyed are not great. Therefore, the second approach is taken.

Figure 3.2: Panoramic image of Madrid center downloaded with the Google Street View API

3.2 3D to 2D matching

This section was inspired by the paper *Efficient Global 2D-3D Matching for Camera Localization in a Large-Scale 3D Map* (Liu, Li and Dai 2017), in which the 3D points of

a model are geolocalized taking as a reference certain images by employing 'covisibility', which defines the probability of two descriptors being in the same image from the database used to create the 3D model.

For this purpose, a detailed study on the 3D models and their associated files was performed. Note, that the model analyzed is a 'dense' 3D model which has a mesh layer of textures associated to the points, made available by the DHLAB for San Gimignano, Italy. For this model, the extrinsic and intrinsic parameters are studied and the camera movement is represented in 3D. However, during the exploration, another idea occurred related to capturing the aerial view of the 3D model, and this approach was followed due to the promise of more straightforward results. Hence, the pipeline results in the following:

Obtain dense 3D > Explore extrinsics, intrinsics > Camera movement > Aerial view

The 3D dense model can be obtained using the library OpenMVS (OpenMVS 2021) which takes a 'sfm' file from a 3D point cloud created with OpenMVG, and outputs a densified file, with more points per view which are then merged into a consistent point cloud. Later, the dense model is employed to calculate a mesh surface and give it texture. However, for the project, a previous texturized model from San Gimignano was already available.

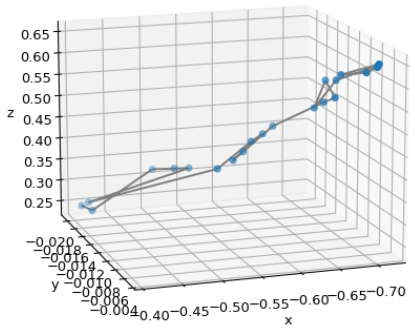


Figure 3.3: Extrinsics

The second step is to explore the parameters in the 'sfm' file. The intrinsics describe the inherent camera parameters such as the model (pinhole, for example), the shape of the images (px,px), the focal length and the principal point, which corresponds to the center of the image ideally. Meanwhile, the extrinsics give information on poses: the rotation matrix and the center of the camera for that pose; the centers represent the camera movement in Figure 3.3. These parameters contain a huge amount of information and they will be used in future lines, but during the evolution of this project, they were set aside as another idea occured, which was to capture the aerial view of the texturized 3D model in order geolocalize it.

3.3 Aerial matching

As seen in '3D to 2D matching' in section 3.2, it can be possible to work with a 3D aerial view in order to assign localization coordinates. With this objective in mind, a detailed exploration of aerial matching is performed. It is also interesting to find relations between map features and cadastre features, which can be then extrapolated to any kind of map or file that contains geo-registered points. Firstly, the possibility to match a cadastre as a reference, with an aerial map is explored, and later the download of open-source aerial maps is automatized. The pipeline that results is the following:

$$Obtain\ cadastre\ and\ aerial\ map > Find\ matches > Automatize$$

In a second step, the 3D aerial view is included in the study in order to find coordinates that are found in a reference aerial map and assigned to matched descriptors in the 3D aerial map, which is the query. As a result, the pipeline obtained is as follows:

$$Include\ 3D\ and\ aerial\ view > Find\ coordinates > Automatize$$

In the first step, a cadastre map of EPFL is obtained from OpenStreetMap and visualized in Qgis, additional to an aerial image downloaded directly from MapTiler. There is a very interesting feature in Qgis that allows to raster a map or any image on top of a shapefile that contains a cadastre, therefore effectively obtaining the overlap of both reference and query and apparently solving the problem, as can be seen in Figure 3.4. The drawback is that the user is required to input manually a large amount of control points, in the order of 20, which should be coincident in the cadaster and the aerial map. This is subject to human error when inputting the control points with the computer mouse, and it is a tedious job. Therefore, OpenMVG from the Linux command line is employed to find features and matches in both maps.

Figure 3.4: Qgis geolocalization of a raster image

A problem arises in this approach, and it is that matches are not found directly between an aerial map and a cadastre, due to their differences, henceforth, the solution to

the problem is to employ an auxiliary intermediate map between the query and the reference. In this case, it is pretty straightforward, as MapTiler offers the possibility to visualize/download the same map either in aerial view or in cadastre mode. As a result, the matching structure from query to reference results in actually being: query aerial view > auxiliar reference aerial view > cadastre reference. This can be seen in Figure 3.5

(a) Query-aerial matching to reference-aerial

(b) Reference cadastre

Figure 3.5: Aerial - cadastre matching

It is possible to see that in order to match a query to a cadastre, an intermediate referenced view is required. This approach is the one considered in the second step, where the 3D aerial view is included.

Also in this step, a Python program is developed to automatize the download of aerial images from other sources, for example Google EE.

In the program, it is possible to input a coarse location of where the query should be , in this case, be San Gimignano, Italy. Then the object Nominatim from `geopy.geocoders`

converts it to coordinates, which are then plotted with `plotly.express` in order to check that it is returning the correct coordinates. The program then requests an authentication into Google with the `ee` library and inputs the coordinates into a `folium` object which returns the satellite view of a map around the coordinates.

In this operation, a drawing package from `folium` is employed which returns the coordinates of the corners of a rectangle over the map, which automatically provides the geolocalized corners of the reference map. At this point, the reference view only has the corners localized, therefore, a grid of the same shape as the reference map (pixels by pixels) is created, and each pixel is assigned one coordinate.

In the second step, OpenCV with ORB method is employed by using the `cv2` library, to find coincident descriptors between a downloaded and referenced aerial map and the 3D synthesized aerial view. Fortunately, the ORB method works better than OpenMVG, finding in the order of 150 ORB descriptors agains the 0 SIFT descriptors found by the OpenMVG pipeline.

Later, the coordinate of the reference pixel is associated with the query pixel and all of the matched descriptors are localized, as can be seen in Figure 3.6. This is a very interesting result, as the query 3D aerial view now contains geo-referenced pixels, which can be in later steps associated to 3D points in the upper view of the 3D model and therefore localized (not their height, of course).

Figure 3.6: Assigned coordinates to San Gimignano query

Chapter 4

Conclusion and future lines

This project is, as a conclusion, the result of the exploration of several different approaches in order to find point coordinates in a 3D model. In this aspect, it is important to highlight that each approach has had different results which decided if that particular path was worth following or it was more efficient to change to a different approach. In particular, the first image-based approach of afoot-taken images posed several problems when trying to escalate the Python program to generic locations and not determined to a specific building or museum. Therefore, a less generic approach was taken, the square-investigation which provided less escalation problems, but which, on the other hand, introduced noise and distorsion into the reference panoramic image which reduced the number of descriptors found in the image.

The second approach, which tried to make most of the information contained in a 3D point-cloud model basically resulted in an investigation of camera parameters and Structure From Motion development, and gave the idea of acquiring a synthesized aerial view of the 3D model, in particular of the San Gimignano model.

Hence, the third approach is based on matching a synthesized view against a georeferenced map, and this provides more satisfactory results, as around 150 ORB descriptors were computed and geolocalized, and a semi-automatic script was developed to enable the user to access all files and on-line maps directly from the Jupyter Notebook environment. Of course, not always a 3D model of such high-quality is available, and aerial views cannot be synthesized so easily.

Therefore, the project concludes with the exploration of several approaches, and the final design of a semi-automatic Python pipeline which enables to geolocalize the found ORB descriptors in a query synthesized aerial view from a 3D model.

Of course, the investigation on this topic is not finished and there are several future lines that could be followed. For instance, the evolution of this same study line, and continue to assign the coordinates found to the actual 3D points in the 3D aerial view, and find a process that allowed to calculate the height of the localized points, maybe through afoot-taken images of the area.

Other study line could be to continue with the panoramic exploration, trying to undistort as maximum the panoramic image and employ the extrinsic parameters in the 'sfm' files to assign orientation and location to each query image against the panoramics. Finally, as a third example of future line, another approach could be taken, for instance the creation of a vocabulary tree with descriptors that are then matched to 3D points for instance.

Bibliography

AliceVision (2021). *AliceVision Meshroom*. AliceVision. URL: https://alicevision.org/ (visited on 11th Dec. 2021).

all3dp (2021). *Best Photogrammetry Software of 2021*. all3dp. URL: https://all3dp.com/1/best-photogrammetry-software/ (visited on 5th Jan. 2022).

Agisoft (2022). *Metashape*. Agisoft. URL: https://www.agisoft.com/ (visited on 5th Jan. 2022).

PIX4D (2022). *PIX4Dmapper*. PIX4D. URL: https://www.pix4d.com/ (visited on 5th Jan. 2022).

Wang, Chun-Po, Kyle Wilson and Noah Snavely (2013). 'Accurate Georegistration of Point Clouds Using Geographic Data'. In: *2013 International Conference on 3D Vision - 3DV 2013*, pp. 33–40. DOI: 10.1109/3DV.2013.13.

Liu, Liu, Hongdong Li and Yuchao Dai (2017). 'Efficient Global 2D-3D Matching for Camera Localization in a Large-Scale 3D Map'. In: *2017 IEEE International Conference on Computer Vision (ICCV)*, pp. 2391–2400. DOI: 10.1109/ICCV.2017.260.

Sattler, Torsten et al. (2015). 'Hyperpoints and Fine Vocabularies for Large-Scale Location Recognition'. In: *2015 IEEE International Conference on Computer Vision (ICCV)*, pp. 2102–2110. DOI: 10.1109/ICCV.2015.243.

CloudCompare (2020). *CloudCompare Project*. CloudCompare. URL: https://www.cloudcompare.org/main.html/ (visited on 30th Oct. 2021).

GoogleStreetView (2019). *GoogleStreetView*. Google LLC. URL: https://www.google.com/streetview/ (visited on 5th Dec. 2021).

GoogleMaps (2019). *Google Maps*. Google LLC. URL: https://www.google.com/maps/ (visited on 24th Nov. 2021).

MapTiler (2022). *Map Tiler*. maptiler. URL: https://www.maptiler.com/ (visited on 15th Dec. 2021).

OpenStreetMap (2021). *OpenStreetMap*. OpenStreetMap Foundation. URL: https://www.openstreetmap.org/#map=6/40.007/-2.488 (visited on 29th Dec. 2021).

IGN (2021). *Remonter Le Temps*. IGN. URL: https://remonterletemps.ign.fr/ (visited on 1st Dec. 2021).

Geofabrik (2020). *geofabrik*. Geofabrik GmbH Karlsruhe. URL: https://www.geofabrik.de/ (visited on 19th Dec. 2021).

OpenData, Paris (2021). *opendata*. OpenData. URL: https://opendata.paris.fr/pages/home/ (visited on 12th Nov. 2021).

OpenCV (2021). *opencv*. opencv. URL: https://opencv.org/ (visited on 13th Dec. 2021).

OpenMVS (2021). *openmvs*. openmvs. URL: https://openmvs.org/ (visited on 1st Dec. 2021).