Automatic user generated image collecting as database for NEOHAZ 3D flood level mapping project

Julian Krauth
Heidelberg University
Study course: Geography
j.krauth@uni-heidelberg.de

ABSTRACT

This assignments outcome is based as a practical approach to automatically collect potential content in order to contribute to the NEOHAZ 3D flood level mapping project. As input source, several social-media platforms are analysed in terms of their content suitability and availability. The main idea is to create a Python script which filters social-media messages spatially (to area of interest) and semantically (image tags, temporal and qualitative). Furthermore, with this approach, different time aspects for flood level mapping could be analysed (if image quantity is sufficient). A main source of suitable images is the social-network platform Flickr, with its detailed image metadata (precise timestamps, sensor information) as well as additional information as auto-generated tags which could be used to identify buildings and at last the easy-to-operate Python API. The Aim is to create a framework that is capable of collecting images from different areas/times with minor changes to the script, for example a shape-file with a bounding box of another area of interest.

Keywords

NEOHAZ, 3D, Flickr, Flood, Social-Media, VGI

INTRODUCTION

Generating 3D flood-related point clouds is the goal of the NEOHAZ 3D mapping project, in order to provide support in disaster recovery or develop mitigation plans. The necessity of accurate flood information is present, for pluvial urban floods which are not well documented in particular (Griesbaum et al.,2017). The study developed a new low-cost approach to determining local flood elevation, based on which this assignment tries to develop an additional data acquiring method. The workflow shown in figure 1 describes the process from data gathering (Input) to data preprocessing, method development and creating output data. This final assignment tries to extend the current method of gathering (geolocated) flood images via Flickr.

The specific use-case of flood images is time-dependent in terms of gathering pictures of buildings under influence of a high water-level. The acquisition of expert non-flood images of buildings does not have this time-dependency whereas the specific date when a flood image was taken is crucial because the water-level is mostly changing dynamically throughout time. Therefore, in order to get suitable input of building images under flood-related conditions, volunteered geographic information (VGI) is a potential source for the data gathering process. Through the crowd-sourced method with large groups of photographers, the area of interest is spread to the regions where people use the specific social network used.

In this assignment, the social network Flickr is used to develop a process that creates an automatic output of geolocated flood images. Flickr is an image hosting website which provides massive amount of data (10 billion images from 122 million users in 2016, Digital Marketing Ramblings, 2017). A major advantage of Flickr is its detailed metadata description with image size, camera type or geolocation as well as its image tagging options with which the content of a photo is characterized. Conclusively, Flickr provides a python-based API which allows users to collect data from public photos. The Flickr API in this project is used to spatially and semantically preselect images based on the specific requirements of the NEOHAZ project. Upcoming research questions deriving from the 3D flood level mapping project, which this assignment tries to answer, are described in the following chapter.

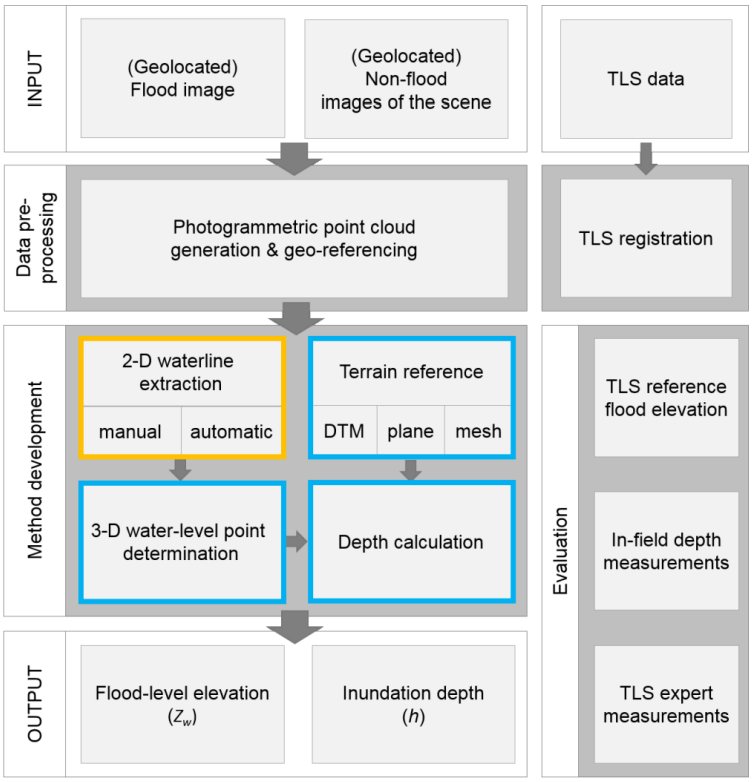


Figure 1. Human Computer Interaction of NEOHAZ 3D flood level mapping project

RESEARCH QUESTIONS

The level of suitable outcome from Flickr is depending on area of interest, level of local social-media usage and amount of available photographers. Following research questions are raised and tried to answer by finishing this project for the case-study of Heidelberg:

- Is the filtered data appropriate for flood level mapping?
- Which filters create the best output?
- Which social-media platform fits our needs the best, a comparison.

Suitability of gathered data for flood level mapping

The script used for this assignment investigates the flood-related posts for the case-study Heidelberg. For the time period of 01.01.2007 until 08.09.2017 a total output amount of 57 images are collected for the administrative boundary of Heidelberg, every picture fulfils the selected filter requirements (spatial, qualitative, thematic and temporal). Of these 57 pictures is only **one** not in original resolution (but still 1024 pixels on largest side), which is relevant for further processing methods. The higher the image quality, the better is the feasibility for mapping purposes. For large and medium size pictures (smaller than original source files), an individual analysis of suitability is necessary, whether or not a photo fits the mapping requirements and therefore are left out in calculation. Another case of suitability check is needed when the thematic context is abstract and not appropriate for flood level mapping. Images with the tag *flood* can contain flood-themed pictures without buildings for example. These false positives have to be manually deselected with optical methods.

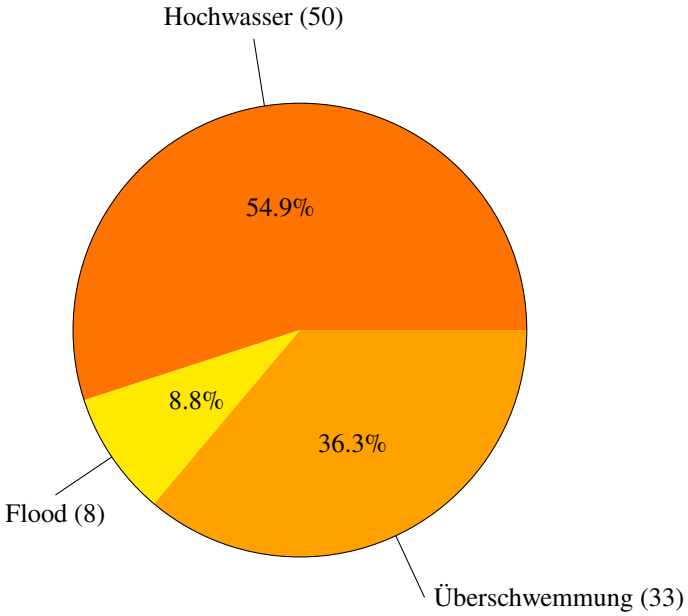
Total Images	Suitable	Thematic issue	Quality issue
57	17	36	4

After manually investigating all output images, conclusively 17 of 57 pictures are suitable for flood level mapping (flood + building displayed in required qualitative condition). 36 of 57 images have a different thematic context and the four remaining photos are present in unacceptable quality (buildings in far background or night-time picture). This low number of image quality issues is dependent by the semantic filtering process where only photos with a suitable resolution are selected. Other than by resolution, image quality is also reliant from exposure to light. Some pictures had a large resolution but were taken at night where less details of the content are visible. Figure 2 shows an example of an image with thematic problems, here, a flood is seen but not the front of the building which might be under water-level.

Figure 2. Not-suitable image from generated Output (thematic context)

Semantic filter quality (tags)

The choice of appropriate filter tags is crucial for the data collection method of this assignment. Nevertheless, good tag selection is not only dependent by subject, moreover, lingual aspects or the distribution of users for specific social-media platforms are a factor. While users from France may use the expression *le déluge* for a flood-related image, Italians may use *l' alluvione* for a same kind of image (country-specific translation of "flood"). The final output created by the script of this assignment described following amount of individual tag uses:



As seen in the circular diagram above, overall more than 57 tags are used (91 of filtered tags, several hundred in total). The filtertags *Flut* and *Floodplain* were not used by users in this research area and time of interest. For each picture, more than one suitable tag is possible, so *Flood* and *Überschwemmung* could both describe the same photo. To eliminate resulting redundancies, a function is called in the script which writes a picture and its metadata only distinctively (described in next chapter: Methodology).

Social-media platform comparison

Flickr, which launched in 2004, is a photo-sharing and platform suitable for casual users as well as high-quality photographers. A main advantage is its available data per picture, a wide range of meta information is connected to each image (e.g. camera type, resolution, tags, autotags or exposure time). Moreover, Flickr provides an easy-to-use API which allows users to collect publicly available data from the web-servers. In September 2016, Flickr had 122 million active users (Digital Marketing Ramblings, 2016). Other than Flickr, the micro-blogging platform Twitter is launched in 2006 and is not specialised solely on photos, users communicate by distributing short text messages. In January 2017, the total amount of active users were 317 millions (Smart Insights, 2017). Similar to Flickr, Twitter provides an API for data collecting. Therefore, a related approach to this assignment is suitable for data gathering, as standalone method or as supplement.
Where Facebook is used by the highest amount of users compared to the platforms mentioned before (1.8 billion in January 2017), their data API restricts data gathering process due to privacy concerns. In 2009, Foursquare launched its venue information platform where users can check-in and rate their current location. Compared to Flickr, Foursquare uses a similar API which enables access to geolocated data. In December 2015 a total amount of 55 million active users were online on foursquare (Digital Marketing Ramblings, 2016). Due to its venue principle, an analysis of flood data is not feasible for this project in practice.
Although this assignment has its focus on gathering data from Flickr data, a comparable approach with help of Twitter data is most likely creating similar output, mainly because pictures are both, in Flickr as in Twitter, described with tags. Facebook is not suitable for technical reasons, whereas Foursquare generates different output.

METHODOLOGY

In order to extract the publicly available image data from Flickr, a python-based API is used. Users are allowed to manipulate, upload, download and replace photo data while beeing registered to Flickr with an authorization key. This key gives permission to access public data whereas the API connection is controlled to prevent exuberant calling requests. The current request limit is set to 3600 calls per hour or one per second (08.09.2017). In case of exceeding the limit, Flickr could deactivate the key to secure the API-usage for other users. Basically, the usage of the Flickr API is free of charge considering a further positive aspect with large community support which is developing scripts and publishing them open-source. The python script used in this assignment is based on the Python Flickr API (Python Software Foundation, 2017). At program start, the script demands two system arguments as input variables:

1. An input Shapefile which defines the bounding box from where the Flickr images are collected (Spatial Filtering). Advantages of this method:
    - No Latitude/Longitude values have to be inserted manually in the code.
    - More complex shapes are possible to inspect, by default the Flickrapi uses only a box with four coordinates. With the help of a Shapefile a more detailed spatial filtering is feasible.
2. An output CSV-file where the automatically filtered images are stored with geolocation and URL-link to the photo-location.

In general, the Flickr database is searched for appropriate data, according to the specifications made in the script (spatial and semantical filtering). Each request is therefor send to the servers via custom authorization key which refers to the amount of requests.

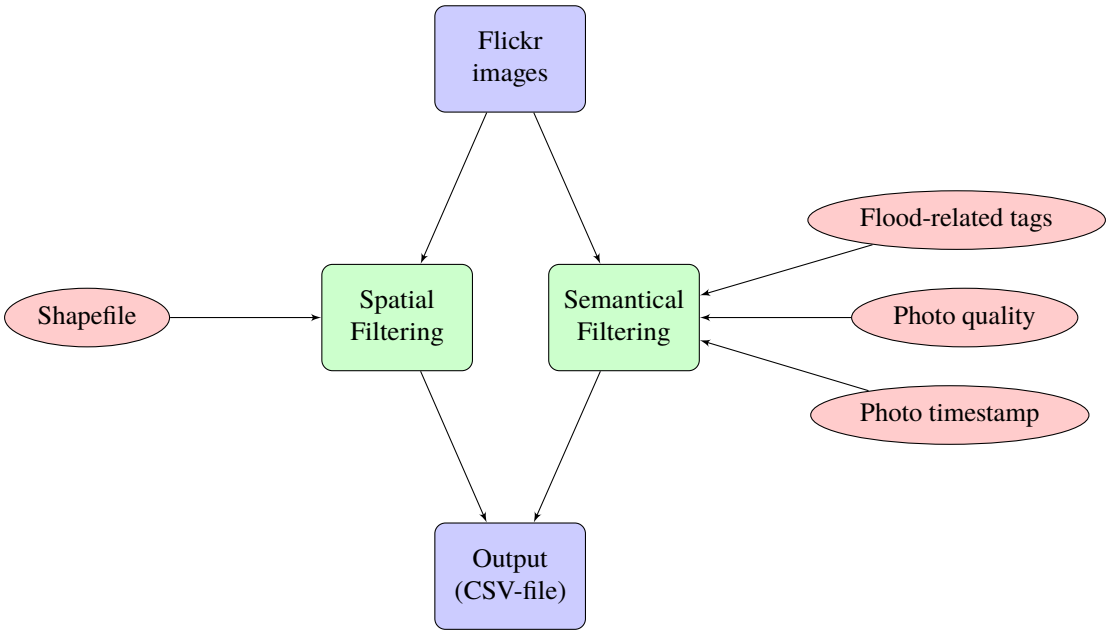


Figure 3. Worflow-diagram for usage of PythonAPI

Spatial Filtering

Boundary of Shapefile extent In detail, the python script reads the input shapefile with a polygon of the research area of interest. Following to reading the file, the specific latitude/longitude coordinates of peak values are automatically returned. In this case, the minimum longitude, minimum latitude, maximum longitude and maximum latitude create a bounding box in which the FlickrAPI search request is called. Basically, the API is searching in a box with four nodes, whereas a polygon can relate to a much more detailed shape. So the filtering of images inside a bounding box is just a preselection in order to remove the necessity of manual input. Later in calculation process, the results are spatially filtered to the exact extent of the polygon in the shapefile:

```
1 #set spatialfilter --> shapefile vs point
2 lyr_in.SetSpatialFilter(point)
3 #if point inside shapefile
4 for feat_in in lyr_in:
```

The spatial filtering process is created with help of the GDAL/OGR Python API, in this example, a spatial filter is set on the input shapefile layer with coordinates of examined points (line 2). After setting a spatial filter, the requested points are checked whether or not the coordinates of the image metadata are spatial aligned with the boundary of the input shapefile polygon.

Semantical Filtering

Photo Quality Apart from spatial filtering the search results, a semantical filtering method has to be implemented, in order to get relevant data for the flood level mapping project (as shown in figure 3). A first step towards a suitable outcome is to ensure the image quality is sufficient regarding the mapping process. Therefor, the script filters select only images with a higher resolution than 800 pixels (on larger side) and choose the most detailed resolution over lower resolutions. Preferably, the images are collected in their source format, so no downsizing method is interfering with the data.

```
1  #url_o = original image size --> best quality
2  if "url_o" in photo:
3          url = photo["url_o"]
4  #url_l --> next best image size (1024px on longest side)
5  elif "url_o" not in photo and "url_l" in photo:
6          url = photo["url_l"]
7  #url_c --> next best image size (800px on longest side)
8  elif "url_o" and "url_l" not in photo and "url_c" in photo:
9          url = photo["url_c"]
10 else:
11         url = ""
```

If the original resolution (Flickr URL-link: "url_o") is not available, a large picture with at least 1024 pixels (on longest side) is selected. In case of non-existent original resolution and large resolution images (Flickr URL-link: "url_l"), the third choice of medium sized photos are collected. Medium sized pictures (Flickr URL-link: "url_c") have 800 pixels on the largest side (Flickr Services, 2017). With this filtering process in terms of image resolution, the best available quality is ensured. The script ignores pictures in less detailed resolutions, because of insufficiency for 3D mapping purposes.

Photo timestamp A crucial factor for well recorded images of flood related sceneries is the time parameter. Usually, a flood occurs only to specific periods of time, commonly existing between several hours up to days. In this period of existence, the water-level is not static. During a flood, a dynamic rise and fall of water-level take place in a short time. The FlickrAPI stores metadata of the time a picture was taken with an accuracy of seconds. Furthermore, the script is modifiable in terms of selecting time intervals from when an image has to be recorded. This feature leads to feasible approaches in practice, where the time of flood occurring is well known, and therefore specificly selectable via the photo timestamp filter option. The API provides a time filter option within the photo-search dialog with which the upload-date or the date when an image was taken can be specified:

```
1 photo_list = flickr.photos.search
2 (min_taken_date='2007-01-01 00:00:01')
```

In this projects example, the filter is set to gather information from pictures that are taken after January, the first in 2007 at 00.00 o´clock and one second.

Flood-related tags Without definition of flood-related tags simply every picture without specific context would be returned, whether or not a flood is shown. The creation of suitable tags is dependant on thematic impressions and locational extent. Thematic parameters are required in case of individual search focus, in this assignments example flood designations. The locational aspect is relevant in terms of lingual expressions of Flickr users. This assignment used a training area in Heidelberg, so some expressions had to be in German, e.g. *Flut* or *Überschwemmung*. These factors have to be considered and manipulated in case of differences in geographic expansion. As example, the Heidelberg case-study was performed with following parameters as tag-search:

- Flood
- Floodplain
- Flut
- Überschwemmung
- Hochwasser

As explained above, the tags are chosen by the most common description of flood events in the researched area of interest, for areas in France, the German expressions have to be replaced with their French equivalents. Searching for multiple tag expressions increases the possibility of redundant output data. It is imaginable that a photo of a building in a floodplain is tagged with: *flood*, *floodplain* and *Hochwasser*. This example would create an equal output three times. To prevent this redundancy from occuring, a simple **set** is generated, where the specific URL of the image is stored:

```
cnt_final = 0
uniqueurl = set()
if url not in uniqueurl:
  cnt_final +=1
  outFile.write(add)
  uniqueurl.add(url)
```

At first, a counter is initiated which counts the final amount of photos that are written in the Output-csv file. For using only distinct pictures, a new empty set is created. The advantage of set in contrast to a list is the storage of distinct values only. Even if more than one equal value has been tried to store in a set only one of it is listed. The if-condition afterwards checks whether a URL of an image is already in the set or not. If it is, no further action is necessary because the same picture has already been stored. For a true condition, the counter adds one to its value, the image properties are written in the output-csv file and the photo URL is stored in the set() in order to prevent a duplication.

RESULTS

The script, developed for this assignment, created an output for flood-related events in Heidelberg, Germany from 01.01.2007 until present (last updated: 08.09.2017). As output, the comma-separated-value format (CSV) is used, which allows list structures with header instance.

fid	uid	tags	date	lat	lon	url
101813	12832@N05	flood heidelberg hochwasser	2013-05-02 22:45:12	49.4151	8.7071	https://..
018712	43122@N02	hochwasser neckar brücke	2014-02-12 12:35:52	49.4153	8.7081	https://..
...	...	...	...	...	...	...

As shown in the table above, the header is filled with fid, uid, characterization tags, recording date, latitude, longitude and specific URL link to image (accessible in web-browser). As seen in column 3 (tags), the prevention of redundancy is necessary. Multiple images have more than one acceptable keyword as tag (acceptable keywords for this case-study: *flood, flut, überschwemmung, hochwasser, floodplain*). In case of no efficient redundancy prevention function, several images would be duplicated inside the output file. The first two columns define the ID of the image itself and the ID of the owner who uploaded the picture. In total, 57 photos passed the filter instances and form the output file. The temporal interval is ranging from May 2009 to January 2015 (with time filter from 2007 to 2017), this can be biased by the amount of (not-)occurring floods.
Furthermore, its noticeable that one user did post a larger set of pictures spread over a day. During the 14th January of 2011, one single user posted 36 images with flood-related content in the area of interest of this assignment. Posting more than half of the overall amount of pictures, this fact has a bias on the contribution of the tags used in the output (see part: Semantic filter quality (tags)).

Figure 4. Suitable image from generated Output

Figure 4 shows an image which is classified as suitable for flood mapping, because the water-level is above the lower buildings edge and the whole buildings front is visible (second building from the right image edge, in the center of the picture). Additionally, this images´ resolution is not compressed by Flickr, here 2048 x 1365 pixels are recorded. This assignments largest picture overall is recorded with 3888 x 2592 pixels, so scaling methods are possible without larger quality loss, even if the building of interest is in the image-background.
Even though, the input-shapefile is spread over the whole city of Heidelberg, the collected images are concentrated on small parts of the inner-city. The small red dots in figure 5 indicate collected photos of flood-related content, unsurprisingly distributed along the river Neckar. South-Eastern to the most images is an outlier, which can be explained by the orography of Heidelberg. The photo was taken at the "Königsstuhl", a popular tourist destination because of its view over the city, which has a notch height of 365 meters. Therefore, a flood is visible even more than a kilometre away from the river. More surprising is the tight expansion on the inner-city parts of Heidelberg. Although the research area is spread over the whole city, just a small area is covered with flood images. Expected

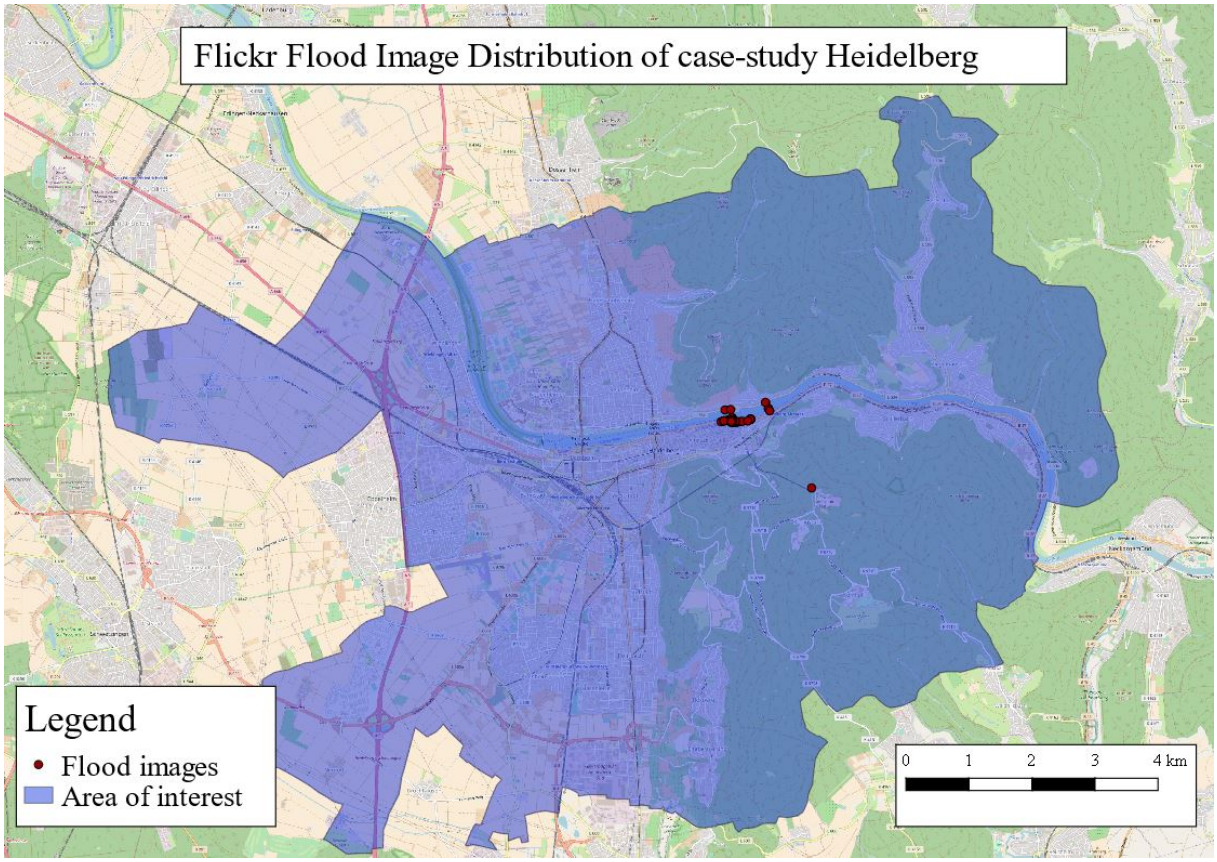


Figure 5. Distribution of flood photos over Heidelberg

was a more heterogeneous distribution along the Neckar river. The distribution can also be biased by population density and social-media usage behaviour of these inner-city areas in contrast to the outskirts of Heidelberg. Other than that, a possible cause for this homogeneity could be a lack of flood protection infrastructure, especially in the areas where the Flickr images occur.

SUMMARY

After investigating the resulting images of the case-study Heidelberg, it is recognizable that not all listed photos are usable for flood level mapping of buildings. Mostly because the pictures´ content is showing flooded sceneries, but often without complete buildings in their background. Therefore, a manual image sorting has to be made on top of the created output, in order to select eligible photos for mapping processes. As the time factor is crucial in flood-related pictures, this approach of VGI-driven image acquisition is well suited because a large group of data collectors can cover up higher amounts of potential research areas. So the data gathering process is shared between many social-media users instead of few experts.
For further research, the potential of other social-media platform sources, for example Twitter, have to be investigated. The possibility of an addition of VGI-based data collection to the present approach of local (semi-)experts taking photos of the research area is the recommended usability for flood mapping. In this case, the advantages of both methods could be brought together. On the one hand, the experts image collection with high-quality pictures of buildings in floodplain but with limited resources in terms of availability in time and human capabilities when a flood occurs, on the other hand, the basic approach of non-expert image collection via social-media posts where a broad user-base share private pictures to the public, though image content and quality has to be further inspected in terms of suitability for certain research purposes.
Another aspect worth further research would be an automatic tag creator inside the script itself. Based on the extent of the source shapefile, basically in which areas the boundaries are, the investigated image tags could be adapted to the specific area of interest, so no manual commands are needed any more. For realisation, just a small country-specific list is necessary which defines word samples based on spatial extent. For example in this case, if the study-area is located in Spain, the Spanish equivalents for flood and floodplain should be used to select flood-related images.

APPENDIX

Python script:

```
#-*- coding: utf-8 -*-
import flickrapi
import sys
import codecs
from osgeo import ogr

#sys.argv[1] --> extent of searched area as shapefile
#sys.argv[2] --> destination of created output as csv
try:
    infile_shape = sys.argv[1]
    outfile_csv = sys.argv[2]

except:
        print "USAGE of this script:
        Shapefile_of_Bounding_Box.shp Output.csv"
        sys.exit()

drv = ogr.GetDriverByName('ESRI Shapefile')
ds_in = drv.Open(infile_shape)
lyr_in = ds_in.GetLayer(0)
extent = lyr_in.GetExtent()

#definition of bounding box of shapefile
#(min Longitude, min Latitude, max Longitude, max Latitude)
shapebbox = "%s,%s,%s,%s"
        % (str(extent[0]),str(extent[2]),str(extent[1]),str(extent[3]))

#My data application:
api_key = "8f8fa0028f7f7dc6d7b1d21fc173c98b"
secret_api_key = "61f983a5fdc9a43e"

#create an API object:
flickr = flickrapi.FlickrAPI(api_key, secret_api_key)

#Search photos from a BBox:
#Documentation at https://www.flickr.com/services/api
#/flickr.photos.search.html
#for flood search enter in tags:
#"flood,Flut,\"Uberschwemmung, Flood, Hochwasser, floodplain"
photo_list = flickr.photos.search(api_key=api_key,
tags="flood,Flut,\"Uberschwemmung, Flood, Hochwasser, floodplain",
min_taken_date='2007-01-01 00:00:01', bbox=shapebbox,accuracy=16,
content_type=7, has_geo=1,
extras= "description, license, date_upload, date_taken, owner_name,
        icon_server, original_format, last_update, tags, geo,
        machine_tags, o_dims, views, media, path_alias, url_sq, url_t,
        url_s, url_q, url_m, url_n, url_z, url_c, url_l, url_o",
        per_page=500,format='parsed-json')


pages = int(photo_list["photos"]["pages"])

#write outfile
outFile = codecs.open(outfile_csv, "w", "utf-8")
header = "fid;uid;tags;date;lat;lon;url\n"
```

```
outFile.write(header)


cnt_source = 0
cnt_final = 0

#write urls in set --> only distinct values --> no redundancy
uniqueurl = set()

for i in range(1, pages+1):
    for photo in photo_list["photos"]["photo"]:
        cnt_source+=1
        #get image content
        fid = int(photo["id"])
        uid = photo["owner"]
        lat = photo["latitude"]
        lon = photo["longitude"]
        #SetPoint_2D method needs float values
        floatlat = float(lat)
        floatlon = float(lon)

        if "tags" in photo:
            tags = photo["tags"]
        else:
            tags = ""

        if "datetaken" in photo:
            date = photo["datetaken"]
        else:
            date = ""

        #url_o = original image size --> best quality
        if "url_o" in photo:
            url = photo["url_o"]
        #url_l --> next best image size (1024px on longest side)
        elif "url_o" not in photo and "url_l" in photo:
            url = photo["url_l"]
        #url_c --> next best image size (800px on longest side)
        elif "url_o" and "url_l" not in photo and "url_c" in photo:
            url = photo["url_c"]
        else:
            url = ""

        point = ogr.Geometry(ogr.wkbPoint)
        #create a temporary point from latlon
        point.SetPoint_2D(0, floatlon, floatlat)
        #set spatialfilter --> shapefile vs point
        lyr_in.SetSpatialFilter(point)

        #if point inside shapefile
        for feat_in in lyr_in:
            #only write an image ONCE in output
            #--> otherwise duplicates when more than one keyword fits
            if url not in uniqueurl:
                cnt_final +=1
                add = "%s;%s;%s;%s;%s;%s;%s\n"
                        % (fid, uid, tags, date, lat, lon, url)
                outFile.write(add)
```

```
114                     uniqueurl.add(url)
115
116
117 print "Points in Bounding Box of Polygon:",cnt_source
118 print "Points in exact contour of Polygon:", cnt_final
119 outFile.close()
```

REFERENCES

L. Griesbaum et al.: Direct local building inundation depth determination in 3-D point clouds generated from user-generated flood images, Nat. Hazards Earth Syst. Sci., 17, 1191-1201, 2017

Digital Marketing Ramblings, available at: https://expandedramblings.com/index.php/flickr-stats/, last access: 21.09.2017

Smart Insights, available at: http://www.smartinsights.com/social-media-marketing/social-media-strategy/new-global-social-media-research/, last access: 21.09.2017

Digital Marketing Ramblings, available at: https://expandedramblings.com/index.php/by-the-numbers-interesting-foursquare-user-stats, last access: 21.09.2017

Python Software Foundation, available at: https://pypi.python.org/pypi/flickrapi, last accessed: 21.09.2017

Flickr Services, available at: https://www.flickr.com/services/api/misc.urls.html, last accessed: 21.09.2017