CENTER FOR
MACHINE PERCEPTION

CZECH TECHNICAL
UNIVERSITY IN PRAGUE

RESEARCH REPORT

ISSN 1213-2365

Building Streetview Datasets for Place Recognition and City Reconstruction

Petr Gronát[1], Michal Havlena[1], Josef Šivic[2],
Tomáš Pajdla[1]

[1]{gronapet, havlem1, pajdla}@cmp.felk.cvut.cz
[2]josef.sivic@ens.fr

[1]Center for Machine Perception, FEE, CTU in Prague, Czech Republic
[2]INRIA, WILLOW, Laboratoire d'Informatique de l'Ecole Normale Superieure

CTU–CMP–2011–01

January 26, 2011

Available at
ftp://cmp.felk.cvut.cz/pub/cmp/articles/gronat/Gronat-TR-2011-01.pdf

The work was supported by the SGS grant.

Research Reports of CMP, Czech Technical University in Prague, No. 1, 2011

Published by

Center for Machine Perception, Department of Cybernetics
Faculty of Electrical Engineering, Czech Technical University
Technická 2, 166 27 Prague 6, Czech Republic
fax +420 2 2435 7385, phone +420 2 2435 7637, www: http://cmp.felk.cvut.cz

Building Streetview Datasets for Place Recognition and City Reconstruction

Petr Gronát[1], Michal Havlena[1], Josef Šivic[2], Tomáš Pajdla[1]

January 26, 2011

Abstract

Google Maps API combined with Street View images can serve as a powerful tool for place recognition or city reconstruction tasks. In this paper, we present a way how to build geotagged datasets of perspective views from Google Maps. Given the initial GPS coordinates, the algorithm can build a list of panoramas in a certain area, download corresponding panoramas, and generate perspective views. In more detail, each panorama on Google Maps Street View contains meta data from which the GPS location and the direction of the view can be extracted. Moreover, the information about the neighbouring panoramas can be obtained as well, hence, a list of panoramas covering a certain area can be built. Downloading panoramas from the list and combining it with the meta data, each downloaded panorama is cut into a set of overlaping perspective views and stored while the camera GPS location, yaw, and pitch are coded in the filename of the perspective view. The geotagged database is subsequently used for place recognition and structure from motion 3D reconstruction.

1 Introduction

Google Maps API is an interface allowing to integrate Google Maps services into external websites. In 2007, Google Maps Street View was released and new features appeared in Google Maps API that provide panoramic views of cities at ground level. We utilize the API and its Google Maps Street View JavaScript panorama objects to build a geotagged database of perspective images in a defined area of the city. For each panorama, this database contains seven overlaping perspective views per 360° in two different pitches, hence, totally fourteen perspective views are generated per panorama. Moreover,

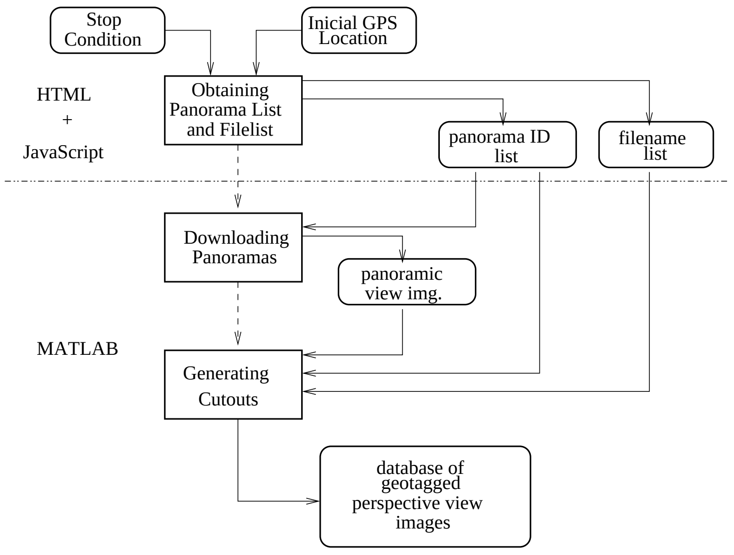


Figure 1: Flowchart of building the geotagged image database.

the information about the GPS location and the geographic North orientation are extracted from JavaScript panorama objects and are coded into the filenames of the database of the perspective views. For developers, both JavaScript and Adobe Flash API are available, in our work we use HTML and JavaScript API combined with a MATLAB script for downloading.

The geotagged database is later used for place recognition [3] or for 3D scene reconstruction respectively using CMP Structure-From-Motion Web Service as demonstrated below.

2 Building the Database

As mentioned above, we utilize the Google Maps Street View to build our database. The actual process which is captured in Figure 1 consists of three consecutive steps: (i) obtaining panorama list, (ii) downloading panoramas, and (iii) generating cutouts. The panorama list building block utilizes the HTML page with JavaScript whereas the panorama downloading and cutout blocks are implemented in MATLAB.

2.1 Obtaining Panorama List and Filelist

We use a HTML page with JavaScript and Google Maps API to obtain the IDs of the panoramas which should be downloaded. The panorama IDs and other meta data are held in `panoData` JavaScript objects. First, the panorama closest to the user-defined start position is searched for using method `panoClient.getNearestPanorama()`, its ID is read from `panoData.location.panoId`, and inserted into a queue. In the main loop, the panorama ID from the head of the queue is used as a query for the `panoData` object using `panoClient.getPanoramaById()` and added to the panorama list. Then, the IDs of the neighboring panoramas are read from `panoData.links` and inserted into the queue. Finally, the whole process is repeated until the queue is empty. An additional hastable is used to prevent inserting duplicate panorama IDs in the panorama list, so panorama list generation proceeds in a wave from the start position and terminates either when all the linked panoramas, i.e. the whole city segment, have been processed or when the server rejects the query.

Another list containing future filenames of the perspective views and the orientation of geographic North direction is created in parallel as shown in Figure 1. This list is subsequently used by a MATLAB script for generating cutouts as described below. We wish to store perspective images in the `latitude_longitude_yaw_pitch.JPG` format in our database, *yaw* being the angle between the view direction and the geographic North. We generate seven overlapping perspective views with *yaw* being uniformly distributed per 360° starting with $yaw := 0°$. For each *yaw*, we generate two perspective views with two different *pitch* angles, namely $-4°$ and $-28°$ to capture both the street level and high buildings.

However, the panorama center on GoogleMaps is oriented towards the direction of the motion of Google car, i.e. direction 0° points down the street. We recover the information about the absolute direction of the motion of the car which is stored in `panoData.Projection.pano_yaw.deg` to obtain panorama orientation w.r.t. the geographic North. Having this information for each panorama, the desired geographic *yaw* can be converted to the relative direction of view w.r.t. panorama center. This angle is also stored in the list for each filename and is subsequently used by our MATLAB cutout script, see Section 2.3.

The script for panorama list and filelist building is described in Algorithm 1 in deeper detail.

2.2 Downloading Panoramas

The $360° \times 180°$ panorama in the equirectangular projection model is stored on Google server at different zoom levels. We download panoramas at zoom level 4 which are represented by $6,656 \times 3,328$ pixels large images stored in 13×7 tiles, 512×512 pixels large each. In the case that a certain panorama is not available at zoom level 4 we download panorama at zoom level 3 which

Algorithm 1 Obtaining Panorama List and Filelist

Input Wave seed location $startLatLng$.
Output Panorama list $idFile$ and filelist $cutFile$.

```
1: open $idFile$ and $cutFile$
2: create empty queue $process$, empty hashtable $visited$, set $numId := 0$
3: set $panoData :=$ panoClient.getNearestPanorama($startLatLng$)
4: enqueue $panoData.location.panoId$ into $process$
5: insert $panoData.location.panoId$ into $visited$
6: repeat
7:   dequeue $nextPanoId$ from $process$
8:   set $panoData :=$ panoClient.getPanoramaById($nextPanoId$)
9:   if $panoData.code \neq 200$ then {server rejected connection}
10:    return
11:  end if
12:  add line [panoData.location.panoId panoData.location.latlng] in $idFile$
13:  for selected $yaw$ angles do {$0°$, $1/7*360°$, $2/7*360°$, $3/7*360°$, $4/7*360°$, $5/7*360°$, $6/7*360°$}
14:    set $shiftYaw := (360 - panoData.Projection.pano\_yaw\_deg + yaw) \bmod 360$
15:    for selected $pitch$ angles do {$-4°$, $-28°$}
16:      add line [numId  shiftYaw  pitch  panoData.location.lat_panoData.location.lng_yaw_pitch.JPG] in $cutFile$
17:    end for
18:  end for
19:  for all $linkedPanoId$ in $panoData.links[].panoId$ do
20:    if $linkedPanoId$ is not in $visited$ then
21:      enqueue $linkedPanoId$ into $process$
22:      insert $linkedPanoId$ into $visited$
23:    end if
24:  end for
25:  increase $numId$
26: until $process$ is empty
27: close $idFile$ and $cutFile$
```

size is $3,328 \times 1,664$ pixels and which consist of 6.5×3.5 tiles each having the resolution of 512×512 pixels. Such panorama is subesequently resampled using bilinear interpolation to have the same resolution as other panoramas downloaded at zoom level 4.

The downloaded tiles are simply stacked together to obtain the panoramatic view using a MATLAB script and the list of panorama IDs described in section 2.1. The actual download link is `http://cbk1.google.com/cbk?output=tile&zoom=4&x=X&y=Y&cb_client=maps_sv&fover=2&onerr=3&panoid=ID` where `X`, `Y`, and `ID` are set according to our needs.

The result of tile stacking is demonstrated in Figure 2(a) where the consequent panorama can be seen. The center of the stacked panorama corresponds to the direction of the motion of Google car and the North direction in panorama can be recovered from the built filelist.

2.3 Generating Cutouts

Subsequently, perspective views are cutout from the downloaded panoramic images. A given panorama can be mapped onto a surface of a unit sphere using the transformation from image points to unit vectors of their rays which can be formulated as follows. For the equirectangular image having the dimensions I_W and I_H, a point $\mathbf{u} = (u_i, u_j)^\top$ in the image coordinates is transformed into a unit vector $\mathbf{p} = (p_x, p_y, p_z)^\top$ in spherical coordinates such that:

$$p_x = \cos\phi \sin(\theta - \theta_0), \quad p_y = \sin\phi, \quad p_z = \cos\phi\cos(\theta - \theta_0), \ . \tag{1}$$

where angles θ and ϕ are computed as:

$$\begin{aligned} \theta - \theta_0 &= \left(u_i - \frac{I_W}{2}\right)\frac{2\pi}{I_W}, && (2) \\ \phi &= \left(u_j - \frac{I_H}{2}\right)\frac{\pi}{I_H}\,. && (3) \end{aligned}$$

We generate 936×537 pixels large perspective images with focal length 502.36 pixels corresponding to HFOV 86° by projecting the surface of the unit sphere to its tangent planes in 14 different directions. Technically, we do bilinear interpolation in source image coordinates.

The inputs for script are panorama ID, 14 filenames, and relative angles from the previously built filename list corresponding to the particular panorama. Given the panorama ID, the appropriate panoramatic view for cutout is loaded. Angle $\theta_0 = 0°$ in Figure 3 corresponds to the center of the stacked panorama. Relevant angles θ_0 for generating cutouts are read from the filelist and perspectiove views are generated according to equations (2) and (3) and stored under appropriate filenames. An example of the cutouts together with the downloaded panoramic image can be seen in Figure 2.

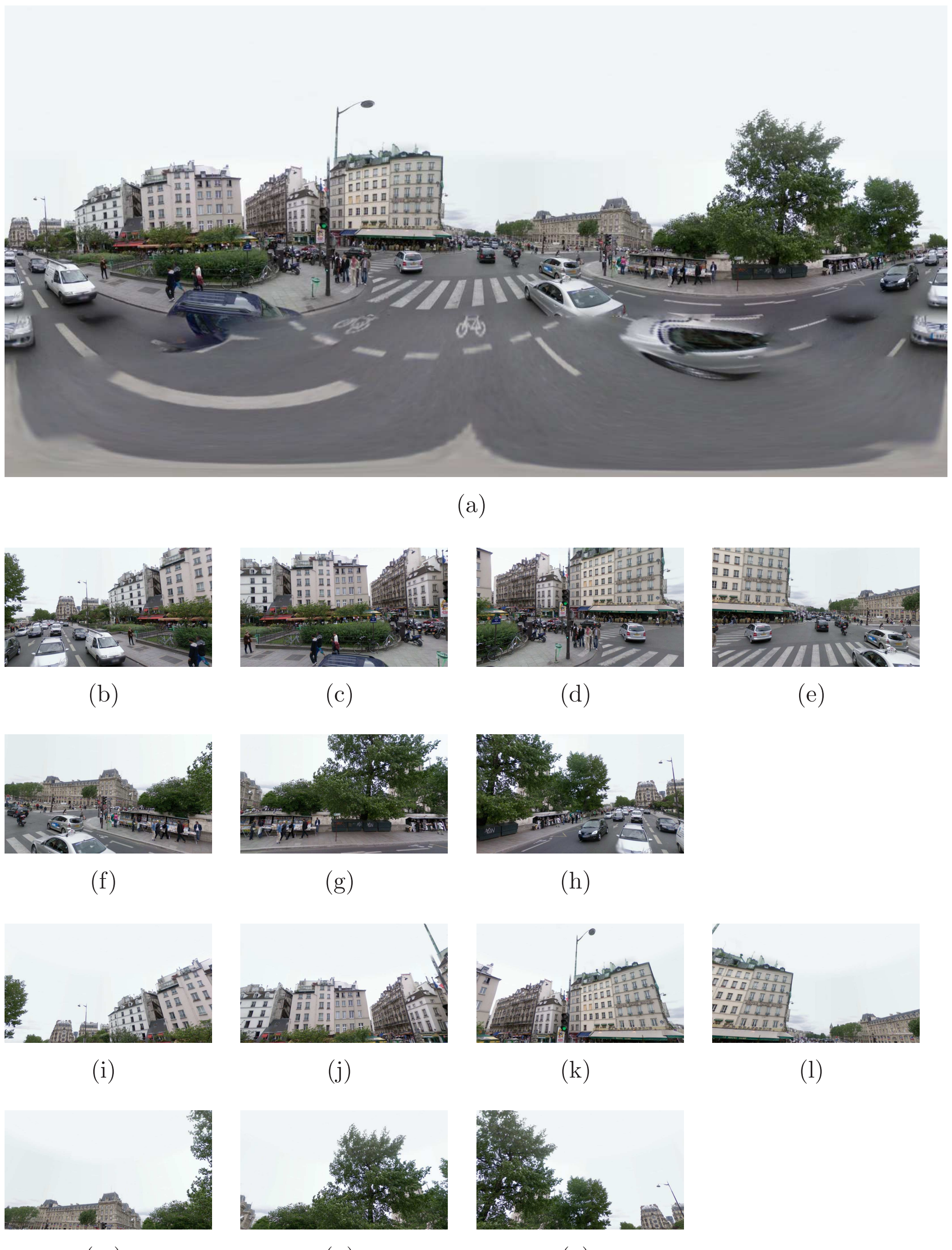


Figure 2: Panorama (a) and fourteen perspective cutouts with pitch -4° (b)–(h) and -28° (i)–(o).

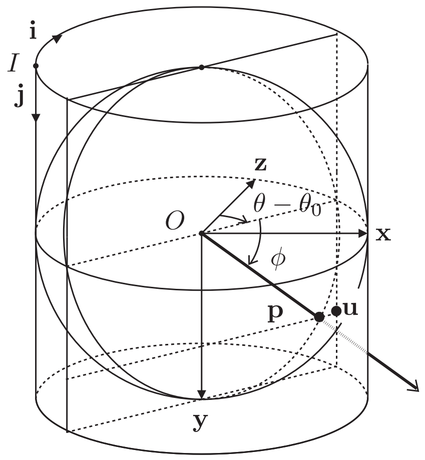


Figure 3: Transformation between a unit vector **p** on a unit sphere and a pixel **u** of the equirectangular image. The coordinates p_x, p_y, and p_z of the unit vector **p** are transformed into angles θ and ϕ. Column index u_i is computed from the angle θ and row index u_j from the angle ϕ.

3 Our Datasets

Three geotagged databases were downloaded using the downloading package described above. Namely, we built databases for Prague where the inicial position was set to GPS (50.081644, 14.416367), Paris with start position at Notre-Dame GPS (48.854466, 2.347617) and finally Pittsburgh with inicial point at GPS (40.44146, -79.995369).

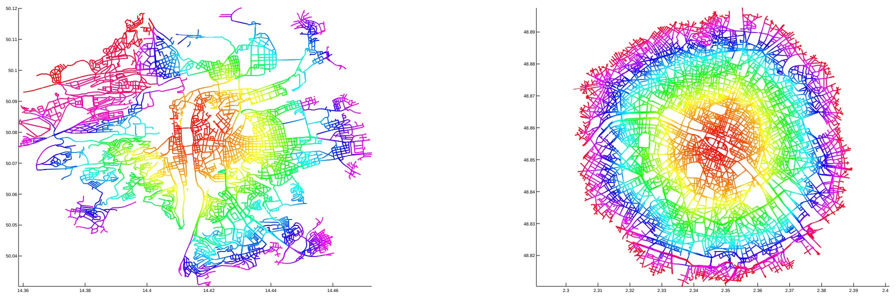
The sizes of the datasets are summarized in Table 3 whereas locations of the downloaded panoramas are shown in Figure 3.

4 Place Recognition

Our goal is to recognize and to localize the visual content in a query image using the built geotagged database [3]. This task is challenging due to changes in scale, viewpoint, and lightening conditions between database images and

Table 1: Downloaded datasets and their sizes.

Dataset	No. of Locations	No. of Views $[\cdot 10^3]$	Size $[GB]$
Prague	$60,354$	845	86
Paris	$86,166$	$1,206$	140
Pittsburgh	$38,111$	533	48



(a) (b)

Figure 4: Maps of the downloaded panorama locations, Prague (a) and Paris (b). The color represents the distance "in the wave" from the inicial point.

the query image. Given the query image of a particular part of the city, i.e. a building or a street, our objective is to recognize one or more images in our databese having the same visual content, i.e. there should be a significant scene overlap in the images.

Being short, the procedure is as follows [5]. SURF features [1] are computed for each image in the database. We build a visual vocabulary using all features over some random subset of the databaset. Using this vocabulary, each SURF feature is mapped into a particular word, hence, each image in the database is represented by some amount of visual words. Visual words are assigned to each document in the database, thus, for each document the tf-idf vector can be computed. That means that having the visual vocabulary and thre computed SURF features, the image can be represented by a single tf-idf vector. Given the query image, SURF features are computed, mapped into a dictionary, and the tf-idf vector is evaluated. This vector is compared with those corresponding to the database images, highly correlated vectors represent couples of images having high probability of visual overlap.

In more detail, first, SURF features are computed for every image in the database which produces 64-dimensional feature space of data. Using the approximative k-means algorithm [4], a user-defined number of clusters is assigned to data. Each cluster center represents a so called visual word. Now, each SURF descriptor is assigned to the closest cluster i.e. a visual word is assigned to each SURF descriptor. Since now, each image is represented by its visual words. Analogicaly to the text retrieval tasks, the tf-idf vector $\mathbf{t_{di}}$ is computed for the i-th image in the database.

Given a query image, the procedure is as follows. SURF descriptors are computed for the given query image and visual words are assigned. The tf-idf vector $\mathbf{t_q}$ for the query image is computed and subsequently compared with the tf-idf vectors of the database images. We are looking for candidate images i in the database such that $i = argmax_i(\mathbf{t_{di}^T} \cdot \mathbf{t_q})$ and the best candidates are selected. Subsequently, we employ spatial verification of visual words via homography and RANSAC for each candidate and select the best one. Since every image in our database is geotagged, the GPS coordinates of the query image visual content can be retrieved.

5 City Reconstruction

The goal is to reconstruct some part of the 3D structure of a city using the geotagged database. However, our objective is not to reconstruct the whole city at the same time from the whole database because that would be inconceivably expensive or even impossible. Instead of it, we manually define a certain area in the map which should be reconstructed. Having the GPS locations of downloaded panoramas, we hand-label panoramas close to the area that may have a visual overlap with it. We select only view directions relevant to the defined area for each panorama. Appropriate perspective view cutouts are subsequently parsed into the SFM pipeline and a part of the 3D scene is reconstructed.

In more detail, we manually localize the visual content in the map for each query image from the recognition task. The border of this visual content, the scene, is highlighted by a polygon whose vertex coordinates are stored for each query image. The situation is illustrated in Figure 5(a) in which the query image and a map cutout with a white triangle bounding the visual content region of the query image can be seen. Locations of available panoramas are displayed by red circles in Figure 5(b).

A hand-labeled list of panorama locations having a great potential to have visual overlap with the query image is built. We remove the perspective views not heading inside the polygon for each panorama location from this list. The result of the selection is shown in Figure 5(b), where yellow arrows represent directions of the selected perspective view cutouts from the geotagged database.

Using this list, we are able to reconstruct the scene using a structure from motion pipeline. We utilize Bundler [6] in the CMP SfM Web Service [2] for 3D reconstruction. A detailed user guide can be found on the website. The MATLAB script is being used to generate the XML files containing the filelists of perspective views or the focal lengths respectively. These XML files are passed to the SFM pipeline to proceed the reconstruction. The results are displayed in Figure 5. Whereas the view in Figure 5(c) is approximately

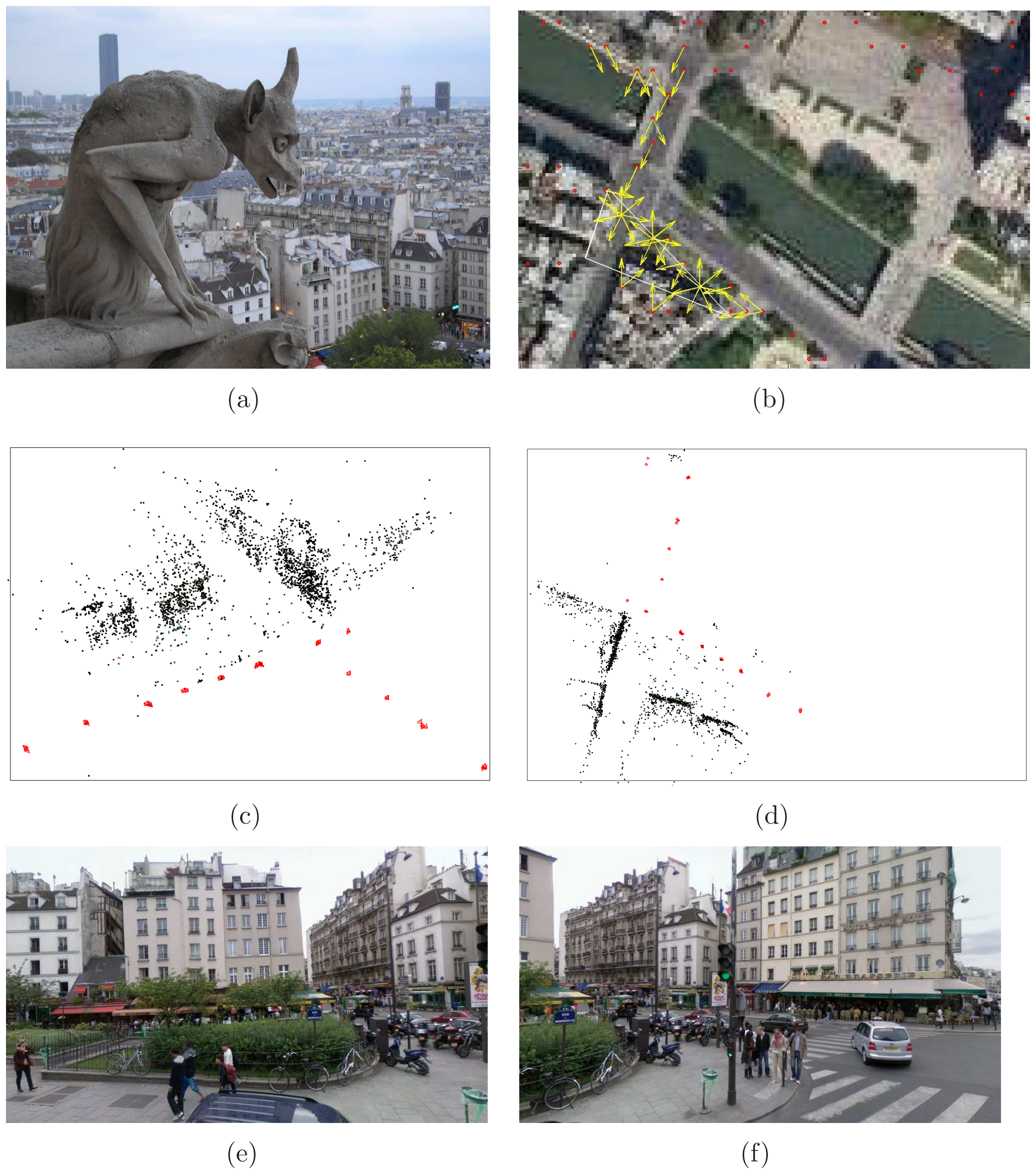


Figure 5: (a) Query image. (b) A map cutout and a hand-labeled white polygon covering the visual region with the query image (a). Red circles show the available panorama locations, yellow arrows represent perspective view directions having visual overlap with the region inside the white polygon. (c) SFM reconstruction – perspective view. Reconstructed camera positions are shown in red. (d) SFM reconstruction – top view aligned with the map cutout. (e),(f) Panorama perspective cutouts capturing the reconstructed junction.

aligned to the perspective of the query image, in Figure 5(d) there is a top view of the 3D reconstruction aligned with the map cutout there.

6 Conclusion

We presented a toolbox for building geotagged datasets of perspective view images. The useage of these datasets for place recognition task was discuseed. Moreover, the geotagged dataset was used for the city recnostruction task such that subset of relevant views is selected and parsed to the SFM pipenline, the results were shown in example.

References

[1] H. Bay, A. Ess, T. Tuytelaars, and L.J. Van Gool. Speeded-up robust features (SURF). *CVIU*, 110(3):346–359, June 2008.

[2] Jan Heller, Michal Havlena, Akihiko Torii, and Tomáš Pajdla. CMP SfM web service v1.0. Research Report CTU–CMP–2010–01, Center for Machine Perception, K13133 FEE Czech Technical University, Prague, Czech Republic, January 2010.

[3] J. Knopp, J. Sivic, and T. Pajdla. Avoding confusing features in place recognition. In *Proceedings of the European Conference on Computer Vision*, 2010.

[4] Marius Muja and David G. Lowe. Fast approximate nearest neighbors with automatic algorithm configuration. In *International Conference on Computer Vision Theory and Application VISSAPP'09)*, pages 331–340. INSTICC Press, 2009.

[5] J. Sivic and A. Zisserman. Video Google: Efficient visual search of videos. In *Toward Category-Level Object Recognition (CLOR)*, pages 127–144, 2006.

[6] N. Snavely, S. Seitz, and R. Szeliski. Modeling the world from internet photo collections. *IJCV*, 80(2):189–210, 2008.