Project acronym: POPART Project number: 644874 Work package: Real-Time Camera Tracking Deliverable number and name: D2.2: Software for basic frame query	Title: Software for basic frame query Work Package: WP2 Version: 1 Date: 31/07/2015
	Author: Simone Gasparini
Type: [ ] Report [ ] Demonstrator, pilot, prototype [ ] Website, patent filings, videos, etc. [X] Other	Co-Author(s): To: Albert Gauthier, Project Officer
Status: [ ] Draft [ ] To be reviewed [ ] Proposal [X] Final / Released to EC	Confidentiality: [X] PU – Public [ ] CO – Confidential [ ] CL - Classified
Revision: Final	
Contents: Deliverable 2.2: Software for basic frame query.	

Deliverable description

The goal of this deliverable is to provide some preliminary tools for querying the database in order to recover the pose of the camera. The approach relies on the creation of a 3D visual database in an off-line step from a collection of views of the scene. The implemented localization algorithm receive as input a sequence of images (either from a video or a set of images) and it processes each image independently by querying the database. The database returns a set of images from the initial dataset that are visually similar to the query image. By matching the features extracted from the query image and those of the similar images the pose can be estimated: since each image of the dataset has 2D-3D associations between each feature and the associated 3D point (from the off-line SfM step), knowing the 2D-2D correspondences between the query image and the image of the dataset allows to recover the 2D-3D associations needed to estimate the pose.

This is the general pipeline for the image-based localization algorithm that has been developed. For this first release of the code we didn't focus on the computational performances of the algorithms (i.e. real-time constraint) but more on the quality of the results and the accuracy of the localization. We developed different approaches and methods that adopts this paradigm in order to test them.

Camera localization

A first naive and simple method consists on taking the first best result of the query and use it for the camera localization: the best matching image is matched with the query image and the 3D-2D correspondences are then computed and used to estimate the camera position. This methods, even if it is very basic and simple works in many cases, but it is not robust to outliers and wrong matches. This method is implemented in the class `NaiveTracker`, with the method `track()`.

A different approach consists in retrieving the N best matches for the query images, and from them retrieve the 3D points of the scene that are visible by the N images. Then the 3D points that are less visible (ie they are visible by less than k images) are discarded. For each of the remaining points, a mean SIFT descriptor is computed from the SIFT features associated to that point in the images in which the point is visible. Then a FLANN matching between these mean SIFT features and the features of the query image is performed, thus retrieving the 3D-2D point associations in order to estimate the pose. This method should be more robust to outliers as it intrinsically considers the results images that are likely to belong to the same cluster of images depicting the same part of the scene. This method is implemented in the class `NaiveTracker`, with the method `trackUsingCluster()`

Code description

This camera localization algorithms have been developed on top of the previous library delivered as Deliverable D1.1. The algorithms for the camera localization are placed in `./library/vision/cameraTracking`. In `./applications/voctree/src` there are a couple of sample applications that can be used to test the algorithms.

trackToSfmdata

This is an utility that takes as input a vocabulary tree, a OpenMVG scene in json format (the `sfm_data.json` file), and either a video, a folder containing some images or a single image. For each of these input, be it a frame of the video or an image, it queries the database and it tries to estimate the pose of the camera using the results of the query. The results are exported in a .json file with the same structure as the sfm_data files of OpenMVG and, optionally, in an Alembic file (.abc)

Usage

```
Options:
  -h [ --help ]                Print this message
  -r [ --results ] arg (=4)    Number of images to retrieve in database
  -t [ --mediatype ] arg (=2)  Input media type (0 image, 1 image sequence, 2 video)
  -c [ --calibration ] arg     Calibration file
  -v [ --voctree ] arg         Filename for the vocabulary tree
  -w [ --weights ] arg         Filename for the vocabulary tree weights
  -d [ --sfmdata ] arg         The sfm_data.json kind of file generated by OpenMVG
  -s [ --siftPath ] arg        Folder containing the .desc. If not provided, it will be assumed
                               to be parent(sfmdata)/matches
  -m [ --mediafile ] arg       The folder path or the filename for the media to track
   -e [ --export ] arg (=trackedcameras.json)   Filename for the SfM_Data export file (where
                               camera poses will be stored).
```

For the calibration file, the format `.cal` is assumed:

```
[3,3]((fx, 0, ppx),(0,fy, ppy),(0,0,1))
[2](imWidth, imHeight)
[4](k0, k1, p1, p2)
```

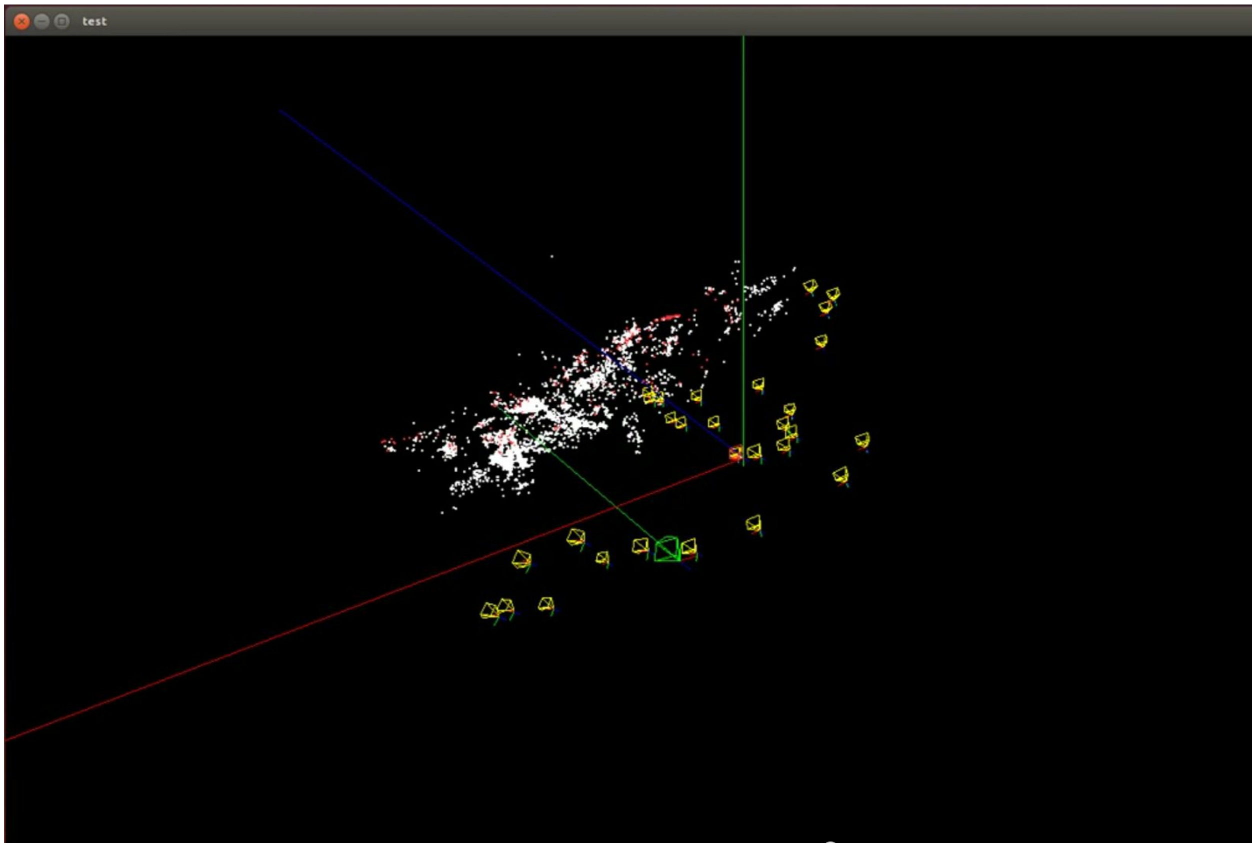
where `fx` and `fy` are the focal lengths, `(ppx, ppy)` is the principal point, `imWidth, imHeight` are the size of the image, and the `ki` are the radial distortion coefficients and the `pi` are the tangential distortion coefficient.

Alembic export

Alembic is a 3D file format which supports the common geometric representations used in the industry, including polygon meshes, subdivision surface, parametric curves, NURBS patches and particles. Alembic also has support for transform hierarchies and cameras. Thanks to the collaboration with MIK, the trackToSfmdata application can now export the localized camera in Alembic format. This is a feature that is in further developement and that can be tested in the [alembic] branch of the repository.

testGUI

This application works exactly like trackToSfmdata but it also display the tracking results in a OpenGL gui.



A screenshot of the testGUI application showing the 3D points of the scene in white, the camera positions for the images of the dataset (in yellow), and the position of the camera beeing localized (in green)

Note: for better results one can use a forked version of OpenMVG which allows to save the color for each 3D points. To do so clone the forked repository and compile the fiatcolor branch:

```
git clone --recursive -b fiatcolor git@github.com:simogasp/openMVG.git
```

Compile OpenMVG with the usual options as described in BUILD and then, for this repository, run cmake with the option `WITH_OPENMVGFIATCOLOR`, e.g.

```
cmake .. -DWITH_OPENMVGFIATCOLOR=ON
```

Usage

```
  -h [ --help ]                         Print this message
  -r [ --results ] arg (=4)             Number of images to retrieve in
                                        database
  -t [ --mediatype ] arg (=0)           Input media type (0 image, 1 image
                                        sequence, 2 video)
  -c [ --calibration ] arg              Calibration file
  -v [ --voctree ] arg                  Filename for the vocabulary tree
  -w [ --weights ] arg                  Filename for the vocabulary tree
                                        weights
  -d [ --sfmdata ] arg                  The sfm_data.json kind of file
                                        generated by OpenMVG
  -s [ --siftPath ] arg                 Folder containing the .desc. If not
                                        provided, it will be assumed to be
                                        parent(sfmdata)/matches
  -m [ --mediafile ] arg                The folder path or the filename for the
                                        media to track
  -e [ --export ] arg (=trackedcameras.json)
                                        Filename for the SfM_Data export file
                                        (where camera poses will be stored).
                                        Default : trackedcameras.json
```

Code download

The code can be downloaded here
https://drive.google.com/file/d/0B1xIYr4Ku8luNGNZdGZ5ZmRESm8/view?usp=sharing

and it will be available in opensource at:
https://github.com/poparteu/cameraLocalization