First person hyperlapse video

Project Part 2 : Computational Photography and Capture

Maud Buffier

Key Idea

Obtain a hyper-last video from a “first person” video

Idea from a Microsoft paper (1) and video :
https://www.youtube.com/watch?v=SOpwHaQnRSY

Key stapes

- *Extract frames from a video*
- *Use Stucture of Motion to infer a point cloud from the world feature as well as the camera positions across the video*
- *Smooth the path of the camera*
- *Synthetize frames seen by the virtual cameras on the new path*
- *Reconstruct a video from those*

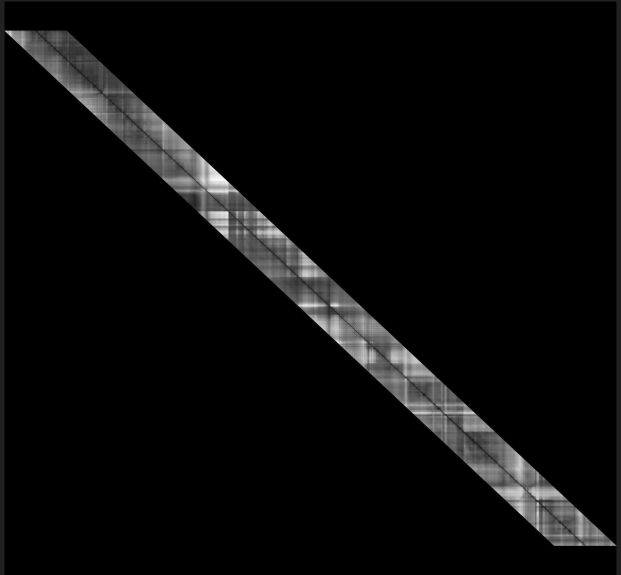
How did I proceeded ?

- Use of a short video to test and improve my algorithm

- Apply it on a final longer video to see the result

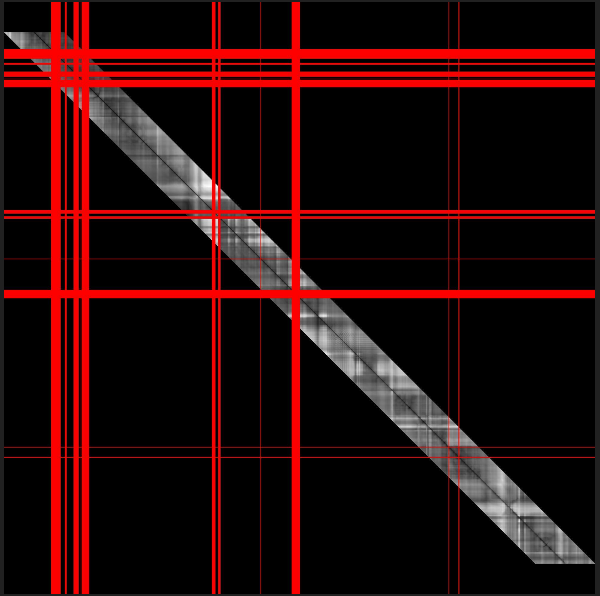
1er step : Extract frames from the video

- Compute a distance matrix for all the frames (code in distanceFrame.m)
- This step was required only for the final video (I only compared 100 frames around the current one to avoid 5 hours of computation)



Extract frames from the video

- Remove frames with too much similarities (when the person stopped for example, it doesn't add any informations)



Extract frames from the video

- We only keep 1 frame every 3 for avoid a high the computational cost in the SfM algorithm (code in fromVideoToImages.m)

We now have a good collection of frames to infer a point cloud

2de step : Structure from motion

- I used the OpenMVG library (2)

- It performs features detection, matching and robust triangulation to obtain a point cloud from multiple images

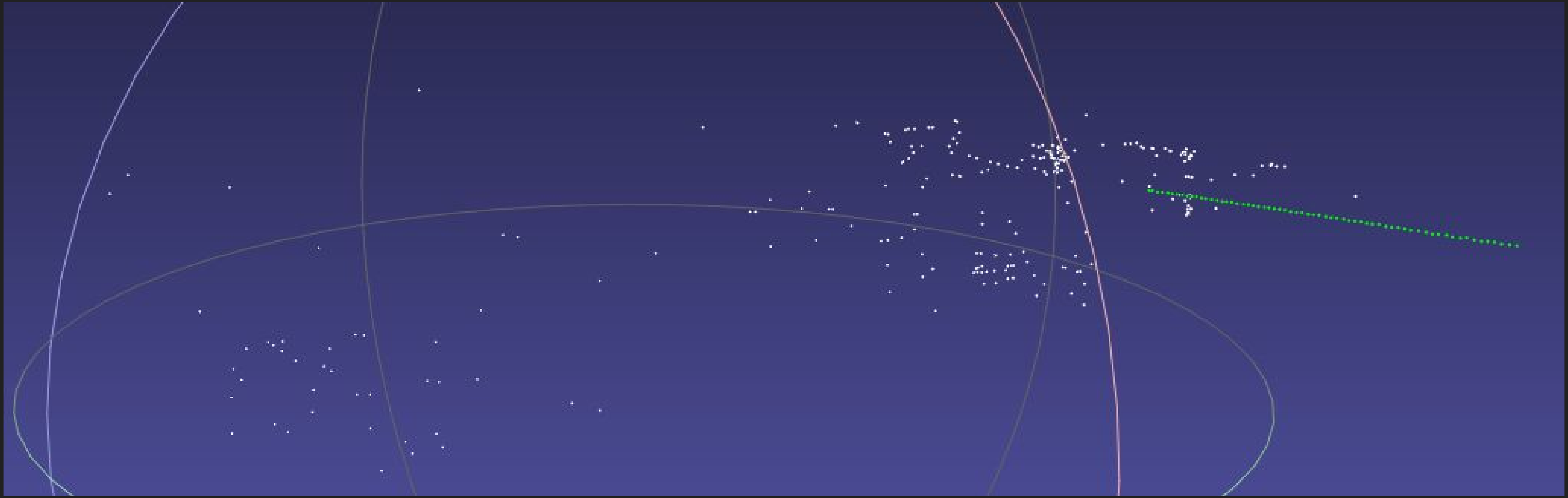
- The code is in (SfM_PoseEstimation.py), extended from the tutorial to perform global structure from motion using OpenMVG

Structure from motion : OpenMVG pipeline

- Find features points in the images
- Match those points across images
- Compute a global reconstruction by triangulation
- Perform a last bundle adjustment over the resulting positions to assure a good result
- Store the point cloud and camera positions in a JSON file

Structure from motion : Result

Here is the point cloud visualized in meshlab (green for the camera positions and white for the point cloud)



Step 3 : Conversion to use into matlab

However, the result of OpenMVG needs to be adapted to my project :

- Extraction of the intrinsic parameters in the JSON file
- The openMVG result gives for each point, what cameras see it. But, I need the opposite for my purpose : points seen by a given camera
- The parameters for the camera are : the center and the rotation and not the translation and rotation so I need to deduce the translation with respect to the center of the world

The code to perform that is in conversion.m (use of the JSON parser from (3))

Step 4 : Smooth the path of the camera

To obtain a smooth path, I need to smooth both the camera positions and orientations (responsible for the shaking)

- For both, I used a parametric spline interpolation (the parameter t was created as the cumulative difference between each points)

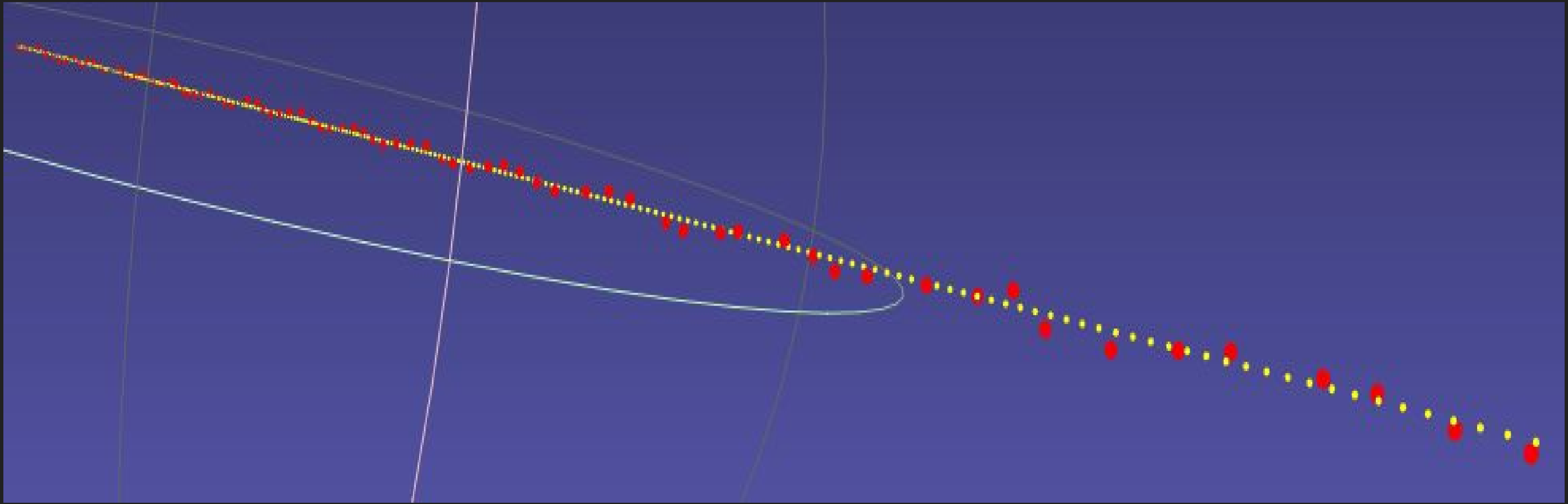
- I fit a spline to each axes using the parameter t and then reconstruct the global curve

- For the rotation, I transform the 3*3 rotations into quaternions using (4) to be able to interpolate them

- The code is in spline3dCurveInterpolation.m for the center and spline4dCurveInterpolation.m for the rotation

Smooth the path of the camera : result

Here is the result for my test video for the camera center positions (red the old one and yellow the new one)



Step 5 : Synthesize new frames

Here is how I proceeded :

- Find the closest camera (C) from this virtual camera position (V) using a kd-tree
- Reproject the points seen by C in the image frame of V using the extrinsic matrix of V
- I now have point positions in the image frame of C and V

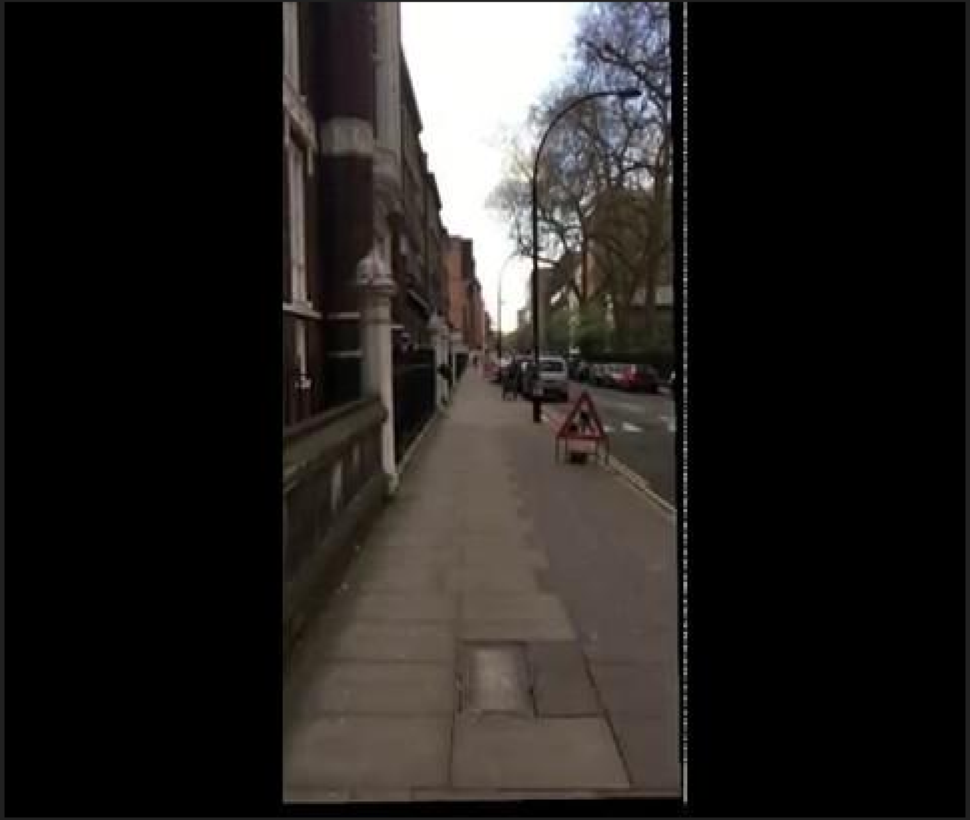
Hence, I can synthesize new frames by 3 differents techniques :

- Estimate an homography from the displacement and warp the image seen by C using this homography
- Infer sparse flow motion from those points, interpolate it to obtain a dense flow and wrap the image of C with it
- Use the last technique to obtain 2 images : one from C and another one from the second closest camera and cross-disolve the images

Synthesize new frames : homography

The code is in mainHomography.m

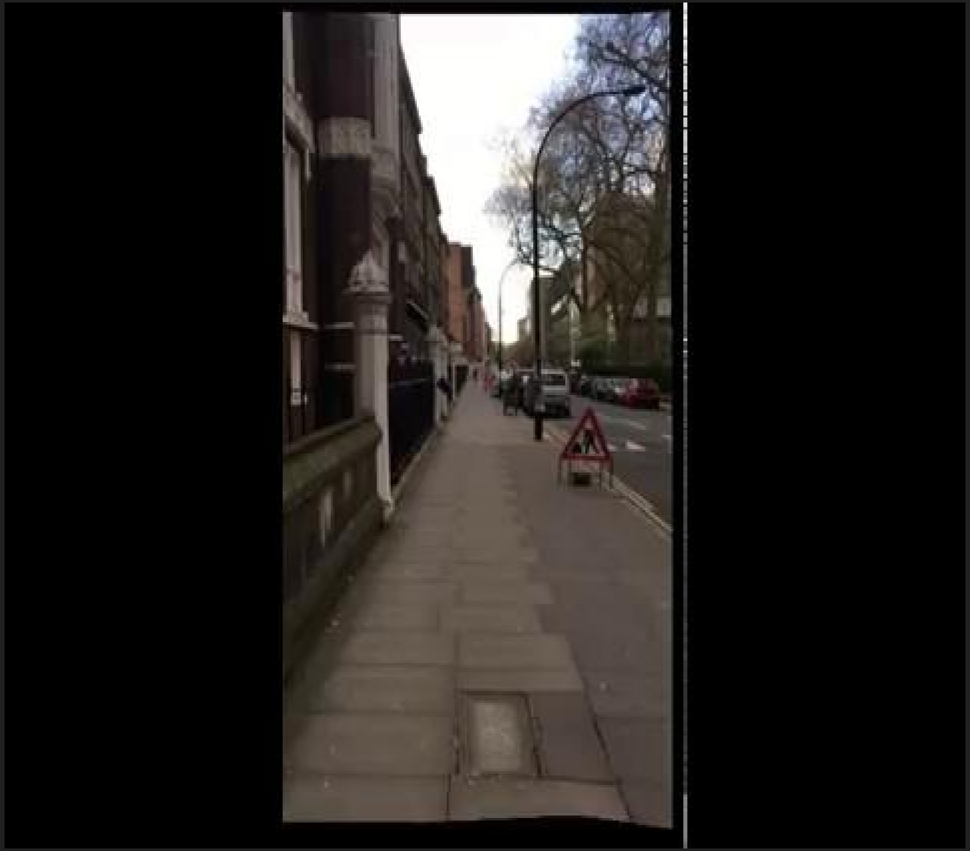
- From the positions in C and V, I compute the best homography
- I synthesize each image by warping C using this homography
- Here is the result :



Synthesize new frames : flows

The code is in mainFlow.m

- From the positions in C and V, I infer the optical flow for this points
- From this sparse flow I use a 2D interpolation to find a dense flow
- I warp the image of C using this flow
- Here is the result :



Synthesize new frames : flow and blend

The code is in mainBlending.m

- I compute 2 images using the flow technique : one from the closest camera (C1) and the other from the second closest camera (C2)

- The distance between V and C1/C2 gives me weights to perform a blending

- The final image is computed by blending the 2 images accordingly to their weights (distance of C1/C2 from V)

- Here is the result :

Advantages / drawbacks of those techniques

- The homography allows to move the entire image, hence less artifacts appear inside the images. However, it lets less places to improvements
- I created the blending solution to avoid “jumps” in the final video when going from a view point to a another which occur in the flow version
- I try to blur the flows to avoid important artifacts but they still occur when the displacement of point is wrong

Final video result :

What to improve ?

- The major problem is the distortion in some part of the images. In the article (1), they tackle this problem by looking at different frames to find undistorted patches for all the virtual image frame. Then, they stitch together the undistorted regions from various frames using a markov random field.

- The moving car at the back are also responsible for the failure of the SfM algorithm. Indeed, it’s normally made for static scenes.

- The original longer video contained some curves and changes of direction but I have never succeeded to make the result of OpenMVG taking that into account. Once it lost track of the first set of feature points, it seems to stop computing tracks. Maybe because the overlap during “the turn” wasn’t sufficient to connect 2 streets together.

- The quality is poor because I had to reduce the images to ⅓ to apply SfM in a reasonable amount of time

Sources

Here are my sources for this project :

1. Kopf, Johannes and Cohen, Michael and Szeliski, Rick, *First-person hyperlapse video* in ‘ACM Transactions on Graphics (Proc. SIGGRAPH 2014)’

2. Pierre Moulon and Pascal Monasse and Renaud Marlet and Others *OpenMVG. An Open Multiple View Geometry library* found in https://github.com/openMVG/openMVG

3. Joe Hicklin, *Parse JSON text* in Matlab File Exchange

4. Przemyslaw Baranski *Quaternions* in Matlab File Exchange

Thank you !

Maud Buffier