Project acronym: LADIO Project number: 731970 Work package: Deliverable number and name: D3.2: SfM integration for 360° cameras and camera rigs	Title: SfM integration for 360° cameras and camera rigs Work Package: WP3 Version: 1 Date: August 31, 2017 Author: Pierre Gurdjos
Type: [ ] Report [ ] Demonstrator, pilot, prototype [ ] Website, patent filings, videos, etc. [X] Other	Co-Author(s): Fabien Castan, Tomas Pajdla, Grégoire De Lillo, Benoit Maujean 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 3.2. SfM integration for 360° cameras and camera rigs	

Table of contents

Introduction

The ambition of this deliverable was to support more image input types that can be generated during a shooting session. Basically, we can get videos and this was not natively supported by the pipeline so it was required to manually extract the frames to be able to launch the reconstruction. We can also have a rig of synchronized videos, which provides an important constraint to improve the quality and robustness of the reconstruction. Finally, VFX people can record 360° images to get lighting information for the Computer Graphic assets that need to match with the set. The ambition is to integrate all these kind of images into the 3D reconstruction pipeline.

Video Support

An obvious input that we get from a shooting session is video files. The AliceVision[1] pipeline is based on still images. In the past, we have extracted frames every N images, but we have discovered that it creates many problems. First you have high probability to get blurry images which will dramatically decrease the quality of the final reconstruction, as this notion is not explicitly taken into account in the pipeline. Then as the motion of the camera is not uniform, we don’t get an even distribution of the images in space. You get high density of images when the camera motion is slow and low density when the camera motion is fast.
So we have developed a keyframe selection module with a simple approach. First, we skip images that are too blurry, this avoid to spend unnecessary computation time on them. Then we compute a similarity score with the previous keyframes using the vocabulary tree approach (developed during the POPART project to improve the selection of image pair candidates during the feature matching process). The basic idea consists in extracting SIFT features/descriptors and creating a global image descriptor that can efficiently be compared to many other images. It allows us to ensure that we are not selecting images that are too similar as too close images will not improve the geometric information but will increase the computation time. Then when images are not too blurry and not too similar to the previous keyframes, we search in a range of N frames the less blurry one. And we iterate like that until the end of the video.
The user can also provide a maximum number of keyframes, in this case, we collect all candidates and select the less blurry ones at the end.

If we have a rig of synchronized cameras, we need to select the same keyframes across multiple videos to keep the geometric constraint. We basically apply the same procedure, but we use the lowest blur level and the highest similarity score across the images of the rig.

Rig Support

The ambition behind using a rig of cameras is to reduce the acquisition time which is a one of the main issues on-set. The intuition is also that this new constraint can be used in the SfM pipeline to improve the robustness, completeness and accuracy because the global pose of the rig at a specific time is constrained by all the cameras of the rig and the sub-pose of each camera is constrained by all the images taken with the rig.

The challenge was that the notion of rig needs to be added in the basic data structure used in the whole SfM pipeline, which represents a large refactoring effort. We have added the rig notion to the first step of the pipeline which lists and analyzes all the images metadata to create the intrinsics groups and now initialize the rig groups. Then we modified the incremental pipeline to take the rig into account. We have

[1] https://alicevision.github.io/

not added the rig support into the global pipeline as it is not used in production. We have added the rig notion into the Bundle Adjustment to declare the new constraints between cameras on rig poses and sub-poses.
The previous implementation was computing the triangulation of all new possible tracks directly after the resectioning of each new camera. It means that the newly triangulated features can then be used for the resectioning of the next camera. But it is a problem because these 3D points have never been refined by bundle adjustment. So we changed this behavior to perform the resectioning of all possible cameras with the existing 3D points and then triangulate all new possible tracks from the new cameras. With the notion of rig, after a single resectioning, many cameras can be indirectly localized, so we have to include them in the search for new track candidates for triangulation.
We also added the corresponding unit tests to validate on simple synthetic datasets.

This first implementation allows to retrieve more cameras in most datasets as reported here:

Dataset names	version	valid views	poses	intrinsics	tracks	residuals
eth3d_electro	rig	215	83	5	11917	45333
eth3d_electro_undistort	rig	315	108	5	13919	50458
eth3d_electro	No rig	167	167	5	13617	52103
eth3d_electro_undistort	No rig	170	170	5	13733	51729
mik_desk_hal2	rig	36	12	2	2625	9361
mik_desk_hal2	No rig	36	36	2	4008	21308
capdigital	rig	268	220	4	126178	462633
capdigital	No rig	223	223	4	128720	473312

Table 1 : Rig support test results

This also slightly increases the number of 3D points reconstructed but it is less than expected. The reason is probably that we are setting the subpose of the rig as soon as we have one valid camera to do it. But if this initialization is not good enough, it will pollute many other cameras with a bad initialization.

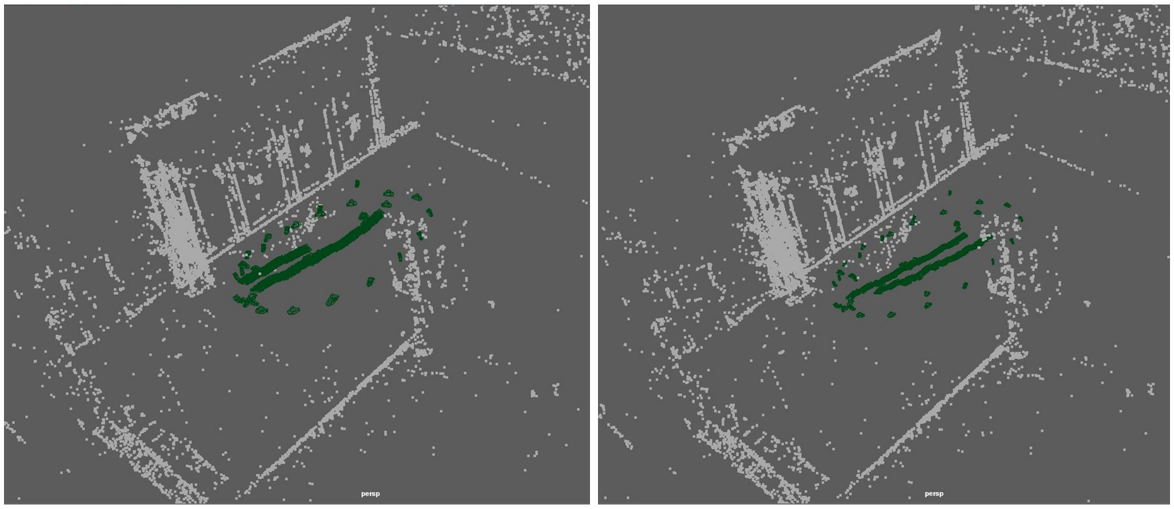

Figure 1 : Dataset eth3d_electro: reconstructed point cloud and cameras, without rig (left), with rig (right)

In the future, we will try to accumulate multiple camera poses before setting the rig pose and sub-poses. In the continuity of the changes in the triangulation, we are also investigating a new robust triangulation method to directly triangulate from more than two views.

The source code is available on our private github:
https://github.com/alicevision/openMVG/commits/dev_rig

360 Cameras

Almost all 360 cameras are composed of multiple sensors and integrate a simple engine to stitch them into 360 images. To get the best geometric information out of them, we have to preprocess the images to retrieve the image from each sensor and then declare these images as a rig of rigidly fixed cameras for the SfM pipeline.
The other challenge is that in most devices, each sensor has a fisheye optics. Fisheye optics are well known to be challenging for SfM pipeline. Our pipeline already uses a specific model for the fisheye distortion, but the pipeline was not estimating it in the matching and resectioning, but was only relying on the Bundle Adjustment (BA) to find these parameters. For standard optics, it works fine but for fisheye optics the initialization for the BA could be too far from the local minimum and may ends up in degenerated models.

Image Preprocessing

We created a new step for the pipeline to preprocess the 360 images. These images can use different mapping methods. The 2 methods commonly used are equirectangular images and dual-fisheye images.

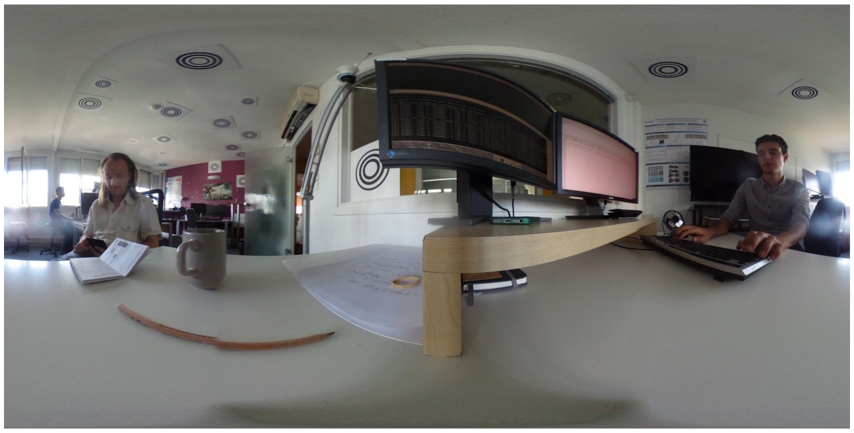

Figure 2 : Equirectangular image from Ricoh Theta (in Mikros Image office)

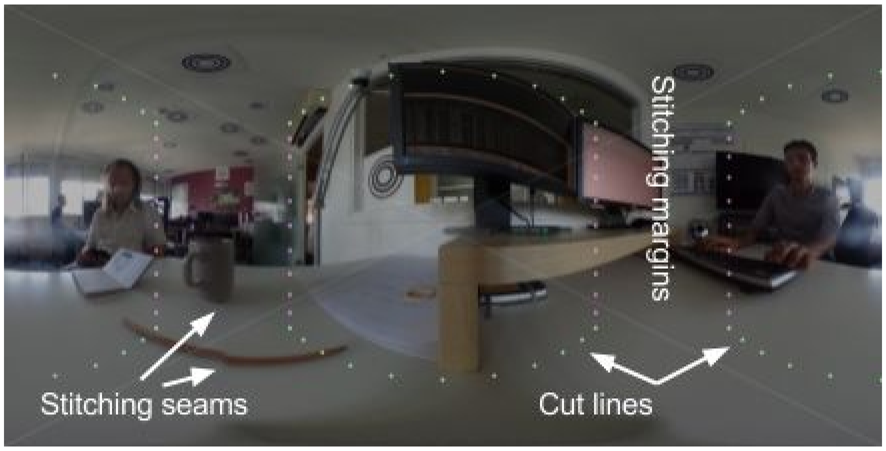

Figure 3 : Same image with cut lines in overlay.
We can see that a margin between images is skipped to avoid stitching areas.

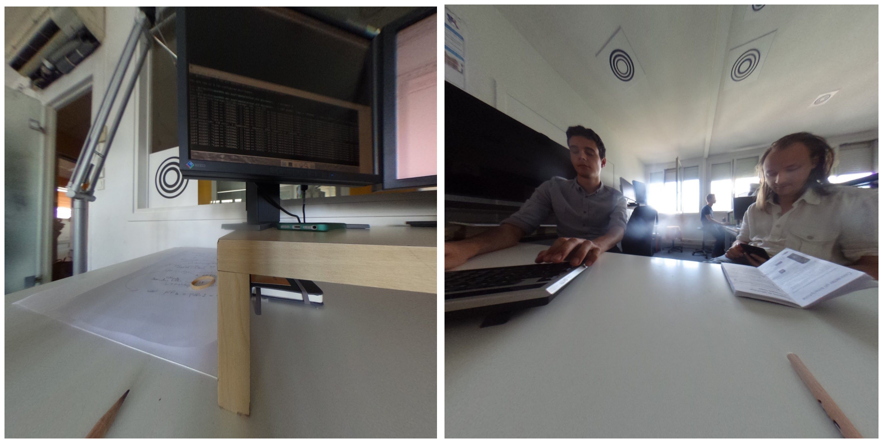

Figure 4 : The 2 output images exported and declared
as a rig of 2 cameras without distortion.

When using equirectangular images, the images from the different sensors are stitched together, so we have to remove the overlapping parts that has been geometrically modified to generate a single 360° image without seams.

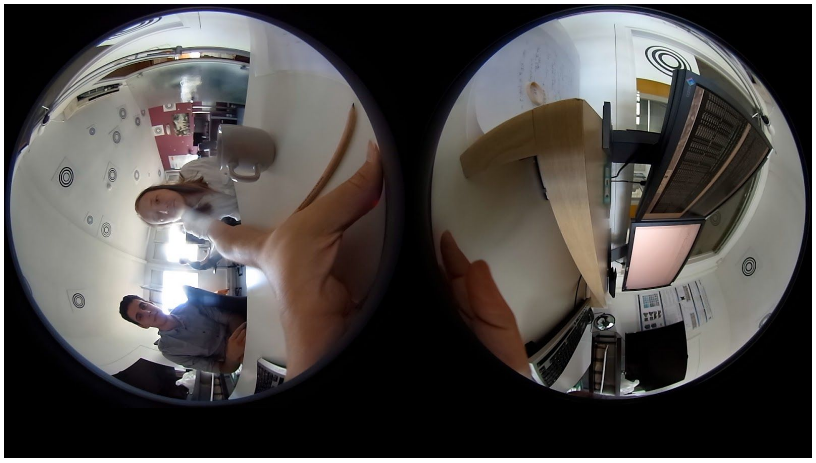
Figure 5 : Dual-fisheye image from Ricoh Theta (in Mikros Image office)

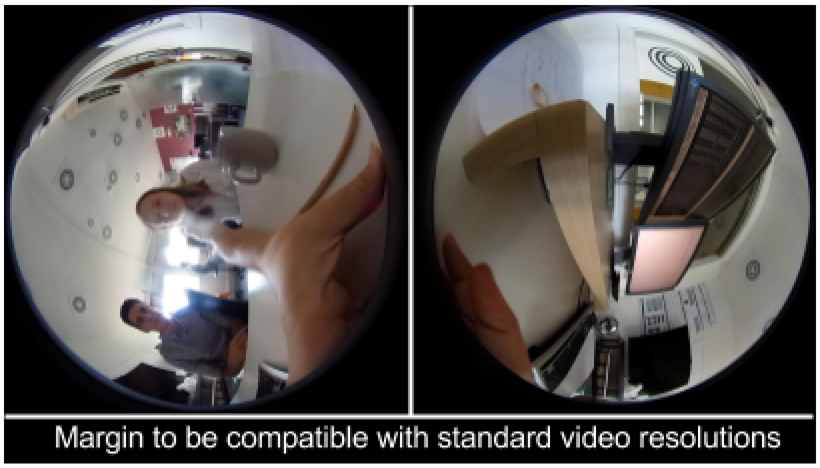

Figure 6 : the 2 output images exported and declared as a rig of 2 cameras with fisheye optics.

When using dual-fisheye images, we just have to split the images into parts. We exposed different options as the layout may change from one device to another.

The source code is available on our private github: https://github.com/alicevision/openMVG/tree/dev_360

Absolute pose estimation with 1 distortion coefficient

As in many SfM pipelines, our pipeline implements a classical resectioning called P3P (Perspective Three Point) method that only estimates the pose based on an initial guess for the focal length.
So we developed 2 new resectioning algorithms, first implemented in Maple and Matlab and now implemented in C++ and integrated into OpenMVG.
First, we implement a P4Pf (Perspective Four Point with unknown focal length) to estimate the focal length at the same time as the camera pose.

The corresponding code is publicly available here:
https://github.com/alicevision/openMVG/blob/popart_develop/src/openMVG/multiview/solver_resection_p4pf.cpp

But for fisheye optics, it is problematic to ignore the dramatic distortion introduced by the short focal length optics. So we implemented a new P5Pfr (Perspective Five Point with unknown focal length and unknown distortion) resectioning algorithm to estimate directly the focal length, the camera pose and one distortion parameter.
The implementation is also publicly available:
https://github.com/alicevision/openMVG/blob/popart_develop/src/openMVG/multiview/solver_resection_p5pfr.cpp

This has resulted in a contribution from CTU to a publication on minimal solvers:
A clever elimination strategy for efficient minimal solvers, Zuzana Kukelova, Joe Kileel, Bernd Sturmfels, Tomas Pajdla, CVPR 2017[2]

Relative pose estimation with 1 distortion coefficient

The last 2 problems to solve to provide an SfM pipeline fully robust to fisheye optics are:

- the ability to use fisheye images to initialize the sequential pipeline with a relative pose estimation algorithm aware of the distortion
- the ability to do the geometric validation of the feature matching step with an algorithm aware of the distortion.

Both elements of the pipeline needs the same algorithm for relative pose estimation taking the distortion into account.
We have analyzed existing algorithms presented in

Z. Kukelova, T. Pajdla. A minimal solution to the autocalibration of radial distortion. CVPR 2007[3]

and

Z. Kukelova, M. Bujnak, T. Pajdla. Real-time solution to the absolute pose problem with unknown radial distortion and focal length. ICCV 2013[4]

for relative pose estimation. Algorithms have been implemented in Macaulay2 to analyze them in exact arithmetics. We are currently working on making the algorithms stable enough to be used in production code as well as possible extendable towards using additional priors on the internal calibration (e.g. on the focal length). This apparently leads to somewhat more difficult computational situation that encountered in previous production implementations. We are currently working on making the solver stable enough to meet production demands.

[2] http://openaccess.thecvf.com/content_cvpr_2017/papers/Kukelova_A_Clever_Elimination_CVPR_2017_paper.pdf

[3] https://www.researchgate.net/profile/Zuzana_Kukelova/publication/4260013_A_minimal_solution_to_the_autocalibration_of_radial_distortion/links/00b7d51ffa8dfba49e000000.pdf

[4] http://www.cv-foundation.org/openaccess/content_iccv_2013/papers/Kukelova_Real-Time_Solution_to_2013_ICCV_paper.pdf

```
R = QQ[k,w,f11,f12,f13,f21,f22,f23,f32,f33,g11,g21,g31,g41,g12,g22,g32,g42,g13,g23,g33,g43,g14,g24,g34,g44,u11,u12,u21,u22]
--- Formulate the epipolar constraints in a classical way
u1 = mat{{u11},{u12}} -- distorted image coordinates
u2 = mat{{u21},{u22}}
v1 = ru2u(u1,k) -- undistorted image coordinates
v2 = ru2u(u2,k)
F  = mat{{f11,f12,f13},{f21,f22,f23},{1,f32,f33}} -- Fundamental matrix
ec = (trn(v2)*F*v1)_(0,0) -- epipolar constraint
--- Formulate the epipolar constraints using lifted coordinates
w1 = mat{{u11},{u12},{1},{u11^2+u12^2}} -- lifted image coordinates
w2 = mat{{u21},{u22},{1},{u21^2+u22^2}}
G  = F | mat{{k*f13},{k*f23},{k*f33}} || mat{{k,k*f32,k*f33,k^2*f33}} -- lifted Fundamental matrix
ec-(trn(w2)*G*w1)_(0,0) -- it works
--- Express all in terms of constraints on coefficients in G
GI = minors(1,mat({{g11,g12,g13,g14},{g21,g22,g23,g24},{g31,g32,g33,g34},{g41,g42,g43,g44}})-G) -- introduce gij unknowns
K  = dia({1,1,w}) -- internal calibration
E  = trn(K)*F*K -- Essential matrix
ID = saturate(ideal(det(E)),ideal(w))
-- ID = ideal(det(F)) + minors(1, 2*E*transpose(E)*E - trace(E*transpose(E))*E) -- Demazure constraints for non-zero w
FD = eliminate(ID,{w}); -- Constraints on F only, w eliminated
GD = GI + FD; -- all constraints on gij, fij, and k
FE = eliminate(GD,{k,f11,f12,f13,f21,f22,f23,f32,f33}); -- constraints on gij only
-- Random linear equations
LE = ideal(random(QQ^8,QQ^16)*trn(matrix({{g11,g21,g31,g41,g12,g22,g32,g42,g13,g23,g33,g43,g14,g24,g34,g44}})))
-- Add linear equations
FL = FE + LE
-- Construct the multiplication matrix for g31 (=k)
Rg = QQ[g11,g21,g31,g41,g12,g22,g32,g42,g13,g23,g33,g43,g14,g24,g34,g44]
use Rg -- gij unknowns only
FLg = substitute(FL,Rg) -- change to Rg
dim FLg -- must be 0
A = Rg/FLg -- Factor ring is an algebra
B = sort(lift(basis(A),Rg)) -- basis of the algebra in Rg
use Rg -- back to Rg
r = (g41*B) % FLg; -- reduced polynomials
M = (coefficients(r,Monomials => B))_1; -- multiplication matrix = coefficients of the reduced polynomials wrt B
r-B*M -- must be 0
-- Solve it in complex numbers
use RR
MR = substitute(M,RR)
(e, v) = eigenvectors(MR)
```

Snippet of Macaulay2 code formulating the camera relative pose problem with radial distortion.