Calibration

Tutorial

October 1, 2021

1 Intrinsics estimation

We follow the example of the repository **multiview_calib/examples/intrinsics**.

1.1 Setting up the camera

The camera has to be in the same mode/configuration as in the final system. That is, same zoom, focus, aperture, resolution, etc.. If the camera will be used in video mode in the final system you should used the same mode here. Auto-focus, auto-stabilization or other such features that dynamically alter the geometry of the image must be disabled.

1.2 Preparing the calibration pattern

The calibration pattern used in the example has 6 inner corners along the height, 8 inner corners along the width and the squares are of size 30×30 mm. However, any other checkerboard can be used as the number of calibration points nor the size of the squares are important.

The first step consist in printing the calibration pattern. It is a good idea to print it with the highest resolution/quality possible. Note as well that the printer may have an option that to fit the image on a page which would alter the aspect ratio of the page. Make sure that this option is disabled or in general that the configuration of the printer is set in such a way that the square are square once printed.

Once printed, the calibration pattern can for example be glued to a solid and flat surface such as a table or a panel.

1.3 Acquisition of the calibration images

Move either the pattern or the camera in order to acquire a set of images from different viewpoints. The variation should be in both rotation and translation of the camera or pattern. If the camera is in video mode, move it slowly to avoid blur. Indicatively, a 60-80 seconds videos at 30 fps should be enough. The video will then be subsampled to remove similar viewpoints to get typically 60-200 images. Ideally, the pattern should be completely visible in the image but is is not a strict requirement as the algorithm discard the images that are not usable/fully detected. The pattern should also be at an angle w.r.t the image sensor and take roughly half of the size of the image. The goal is to have variations in the poses and to cover the whole image.

It is important that in the set of images we acquired, the calibration points are evenly distributed on the whole image. An example distribution is shown in Fig. 2 b). Since we use a high order polynomial to approximate the distortion, at the border of the observed area the function may show large and erroneous variations causing the function to be non monotonous, which is an unwanted characteristic. This means that the corners and the sides of the image should also be well covered. Fig. 1 shows two images of the same pattern. On the left the pattern is as close as possible to the border while on the right there is still some margin that can be trimmed.

1.4 Running the calibration tool

At this point we have a video depicting the pattern in different poses. The first step consist in extracting the video frame which can be done with:

```
ffmpeg -i vid_pattern.mp4 -r 1 frames/frame_%04d.png
```

with the option -r we subsample the video in order to remove frames that are too similar if not redundant. Moreover, we extract the frames at the highest quality, that is without compression (lossless).

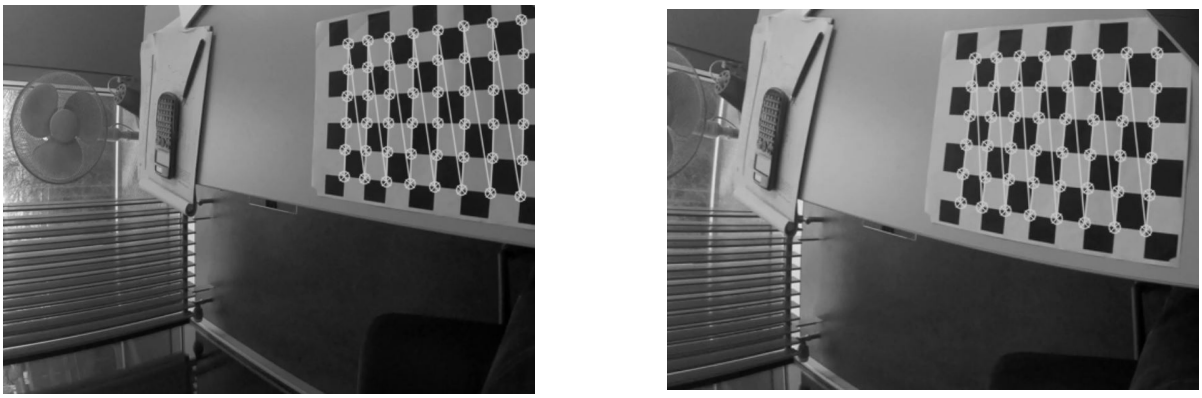
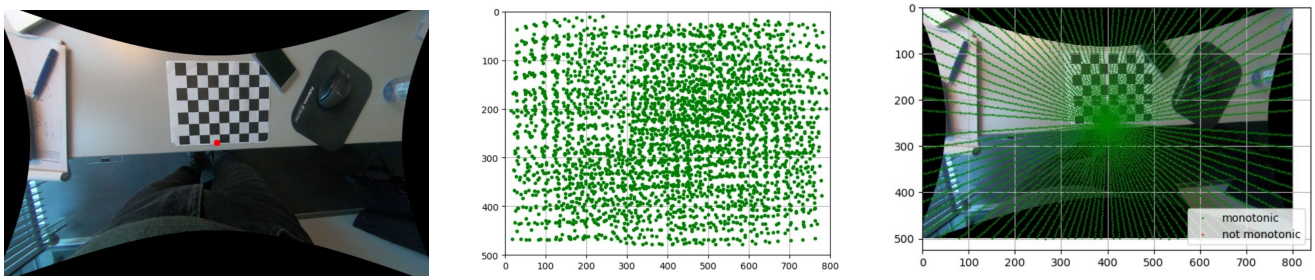


Figure 1: Example showing how the pattern should look like at the border of the image. The calibration points of this pattern are the inner corners of the squares. (left) the pattern is as close as possible to the border of the image. (right) the sheet is close but not the calibration points.



(a) Undistorted image (b) Points distribution (c) Monotonicity

Figure 2: Visual output for a good case

After the extraction we can perform the calibration with:

```
python scripts/compute_intrinsics.py --folder_images ./frames -ich 6 -icw 8 -s 30 -t 24 -rm --debug
```

-ich and -icw define the size of the pattern, -s the size of the squares, -t is the number of threads to be used, -rm enables the use of the rational model which is a more suitable for wider lenses. For camera with a low distortion the rational model is not required. –debug is a flag that enables more data to be dumped for debugging purposes.

The calibration tool should produce the following output: no error messages and a low re-projection error. Indicatively, a reprojection error less than 1 pixels is a good sign of a good calibration. Fig. 2 shows a undistorted image computed from distortion parameters that behaves correctly also at the corners, a proper distribution of calibration points and a plot showing that the distortion function is monotonous in the range of the image.

```
RMS Reprojection Error: 0.13010359523495804, Total Reprojection Error: 0.05847883003072351
```

1.5 Case studies

1.5.1 Case 1

The calibration tool produced the following message:

```
RMS Reprojection Error: 0.2807004705638825, Total Reprojection Error: 0.09515656286369172
-------------------------------------------------
 The distortion function is not monotonous for alpha=0.95!
 To fix this we suggest sampling more precise points on the corner of the image first.
```

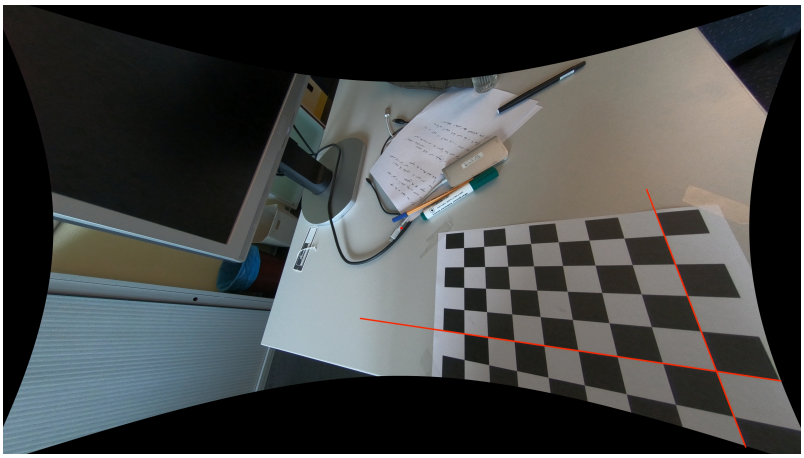
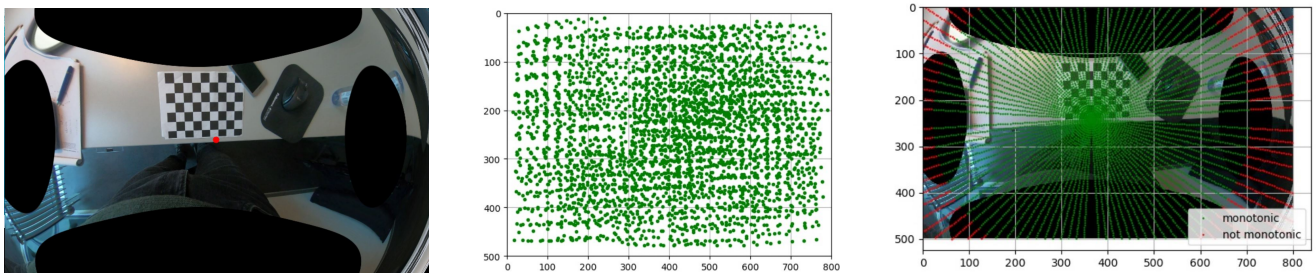


Figure 3: Example undistorted image using well estimated distortion parameters. The key to understand if the distortion function is precise – other than checking the reprojection error – is to verify that lines are straight, especially on the corner of the image.

```
 If this is not enough, use the option Rational Camera Model.
-------------------------------------------------
```



(a) Undistorted image (b) Points distribution (c) Monotonicity

Figure 4: Visual output for a not so good calibration case

It can be noted from Fig. 4 that the distortion function does not behave correctly at the corner of the image. We can clearly see that there is a mirror effect as we approach the corners from the center of the image. This happens because the distortion function is not monotonous in the range of the image. As a result, points that are at the borders can be re-projected/warped back into the image creating these artefact. The reason why the function is not monotonous is because the function is approximated by a high order polynomial and therefore it can produce high variations at the border of the observer region. Moreover, OpenCV does not enforce it to be monotonic since the framework does not support the required integer programming and polynomial inequalities. One way to overcome this is to sample more precise points and to better cover the corners of the image. If this last option is not enough, it is possible to switch to the rational model with the option -rm. This is a distortion model that is more suited for wider angle lenses.

2 Extrinsics estimation

We follow the example **multiview_calib/examples/drone_rolex**.

2.1 Setting up the cameras

The positions of the cameras during the calibration have to be the same as in the final system. A new calibration of the extrinsics is required every time a camera move. Moreover, the cameras have to have the same mode/configuration as in the final system. That is, same zoom, focus, aperture, resolution, etc.. Auto-focus, auto-stabilization or other such features that dynamically alter the geometry of the image must be disabled.

2.2 Data acquisition

The procedure requires a set of synchronized images/videos depicting one or multiple objects such as a drone or plane. The images have to be annotated with the positions of the objects. The more precise the annotations the better the calibration. The positions

2.3 Running the calibration tool

At this stage we have a set of image points for each calibration object in the images that will be used to calibrate the cameras.

The calibration tool requires the following files:

- **intrinsics.json**: Intrinsics parameters for each camera
- **filenames.json**: One image filename for each view for visualization
- **landmarks.json**: image keypoints 2D
- **landmarks_global.json** corresponding landmarks in global coordinate 3D
- **setup.json**: The name of the views and a spanning tree defining the connection between cameras while computing the initial solution
- **ba_config.json** Bundle adjustment config file

To understand the format of the first four files we invite you to follow the guidelines reported in the repository **multiview_calib**. For **ba_config.json** we give an explanation in Section 2.3.2.

2.3.1 Initial solution

We describe here the steps required to compute the initial solution which is used to initialize the Bundle Adjustment parameters.

Relative poses. To compute the relative poses between the camera pairs defined in the file **setup.json** we execute the following command:

```
python scripts/compute_relative_poses.py -s setup.json -i intrinsics.json -l landmarks.json -f
    ↪ filenames.json --dump_images
```

The Residual error and the Sampson distance give you an idea whether the estimated fundamental matrix and hence the relative pose fits the annotated image points. A Residual error of 5 pixels is low if we consider that the images are 4K resolution.

```
2021-07-08 15:51:12,186 [root] Computing relative pose of pair ['6-4', '6-2']:
2021-07-08 15:51:12,222 [root]  0 out of 7174 points considered outliers.
2021-07-08 15:51:12,223 [root]  Fundamental matrix:
             [ 2.10110511e-10  2.22439218e-07 -2.95668490e-04]
```

```
             [-6.12194200e-09  9.91837873e-10 -6.06330738e-04]
             [ 9.75536405e-06 -2.53456977e-05  1.00000000e+00]
2021-07-08 15:51:12,224 [root]  Right camera position:
             [ 0.99984121 -0.00340984 -0.01749051]
2021-07-08 15:51:12,350 [root]  Residual error: 5.02266716598603
2021-07-08 15:51:12,363 [root]  Sampson distance: 75.38368466310308
```

Concatenation of relative poses. To concatenate the relative poses and hence obtain the initial solution for the Bundle Adjustment we execute the following command. It takes the relative poses computed beforehand as input.

```
python scripts/concatenate_relative_poses.py -s setup.json -r output/relative_poses/
    ↪ relative_poses.json --dump_images
```

The output message here below is just a part of the information provided by the tool. It gives you information regarding the concatenation of two pairs of relative poses, the estimated relative scale between them and the new position of the last camera of the triplet. Here the first pair of cameras is ('6-4', '6-2') which is concatenated to ('6-2', '6-3'). The relative scale between pair one and pair two is estimated to be 0.990. This values is usually a good indicator to understand if a relative pose is good. A value that is very high or close to zero should be the warning sign for a problem with a relative pose.

```
2021-07-08 15:59:44,697 [root] Concatenating relative poses for pair: ('6-2', '6-3')
2021-07-08 15:59:44,697 [root]   Relative scale to ('6-4', '6-2') n_points=7148: 0.990+-0.125
2021-07-08 15:59:44,698 [root]   6-3 new position: [1.00883899 0.1722372  0.9563082 ]
```

2.3.2 Bundle Adjustment

- **each_training**: To subsample the calibration points. Integer. Default=1. A value > 1 can be used to speed the algorithm up in the case the number of training samples is large and the optimizer takes a long time to converge. In general is better to use as many points as possible.
- **each_visualisation**: Same as **each_training** but it is used mostly for reducing the amount of information plotted.
- **th_outliers_early**: Points that have a residual error, computed using the initial solution, that are higher then this threshold are discarded. The plot bundle_adjustment/initial_residuals.jpg shows you the initial residuals while bundle_adjustment/early_outlier_rejection_residuals.jpg after this thresholding.
- **th_outliers**: Points that have a residual error, computed using the solution of the first pass of bundle adjustment, that are higher then this threshold are discarded. The plot bundle_adjustment/optimized_residuals.jpg shows you the residuals after the first pass of BA while bundle_adjustment/optimized_residuals_outliers_removal.jpg shows the residuals after the second pass of BA where outliers have been removed. The second pass of BA is performed only if some points are considered outliers and hence removed otherwise is simply not executed.
- **optimize_points**: Boolean flag defining whether to refine the points in 3D. For standard BA this is set to True.
- **optimize_camera_params**: Boolean flag defining weather to refine the camera parameters. For standard BA this is set to True.
- **bounds**: Boolean flag defining whether to use bounded optimization. The bounds define the searching space for the parameters. Setting the bounds correctly allows the optimizer to converge faster and to obtain a more precise solution. Setting the bounds incorrectly impede the optimizer to find the good solution. Be careful when setting the bounds! Make sure that the optimized parameters are not clamped by the bounds.
- **bounds_cp**: List of bounds magnitudes for the camera parameters. If b is the bound's magnitude of parameter x where the initial values x_0 is given at initialization, the bound is $x_0 - b < x < x_0 + b$. The magnitude of the bounds are the same for all cameras and are provided as a list in the following order: 3 values for the rotation

in vector form, 3 for translation, 4 intrinsics [f_x, f_y, c_x, c_y], 5 or 8 distortion parameter depending on the model. Note that the magnitude of the bounds should be defined with the same scale as the initial solution. If for example the translation vector is defined in millimeters, a bound of $x_0 - 2 < x < x_0 + 2$ is most likely to narrow. The translation vectors of the initial solution compute using our tool is in general normalized and in the range $0 - 1$, which is not in world coordinate.

- **bounds_pt**: Magnitude of the bounds for the 3D points. Same explanation as **bounds_cp**.
- **max_nfev**: Maximum number of iterations to perform for the first pass of BA.
- **max_nfev2**: Maximum number of iterations to perform for the second pass of BA.
- **ftol**: Tolerance stopping criterion.
- **xtol**: Tolerance stopping criterion.
- **loss**: Loss function. Default="linear". Other losses such as "soft_l1" and "huber" might be more beneficial in presence of outliers.
- **f_scale**: Value of soft margin between inlier and outlier residuals, Default=1.0. No effect for loss="linear".
- **output_path**: Output folder.

To run the Bundle Adjustment we execute the following command. It takes as input the poses computed beforehand.

```
python scripts/bundle_adjustment.py -s setup.json -i intrinsics.json -e output/relative_poses/
    ↪ poses.json -l landmarks.json -f filenames.json --dump_images -c ba_config.json
```

this script performs the following operations in this order:

1. Estimation of the initial 3D points using triangulation from the intial solution.
2. Outliers rejection (named Early Outlier Rejection)
3. Optimization of the 3D points and camera parameters. (first pass of BA)
4. Outliers rejection
5. (If at least one point has been considered as an outlier.) Optimization of the 3D points and camera parameters. (second pass of BA)

An example of the last part of the outputs message is shown below.

```
2021-07-12 10:52:02,254 [root] Optimization took 103 seconds
2021-07-12 10:52:02,264 [root] Average absolute residual: 0.77 over 6112.0 points.
2021-07-12 10:52:02,784 [root] Reprojection errors (mean+-std pixels):
2021-07-12 10:52:02,786 [root]   6-4 n_points=1361: 1.029+-0.791
2021-07-12 10:52:02,788 [root]   6-2 n_points=1455: 1.066+-0.762
2021-07-12 10:52:02,790 [root]   6-3 n_points=1383: 1.557+-1.149
2021-07-12 10:52:02,791 [root]   6-1 n_points=507: 0.976+-0.607
2021-07-12 10:52:02,792 [root]   6-0 n_points=1406: 1.321+-0.957
2021-07-12 10:52:02,793 [root] Reprojection errors (median pixels):
2021-07-12 10:52:02,794 [root]   6-4 n_points=1361: 0.818
2021-07-12 10:52:02,796 [root]   6-2 n_points=1455: 0.880
2021-07-12 10:52:02,798 [root]   6-3 n_points=1383: 1.237
2021-07-12 10:52:02,799 [root]   6-1 n_points=507: 0.897
2021-07-12 10:52:02,801 [root]   6-0 n_points=1406: 1.075
2021-07-12 10:52:02,802 [root] Visualise all the annotations.
```

Fig. 5 is one of the visual outputs used for debugging purposes. It shows the manual annotations (green), the estimated 3D locations of the object(s) projected into the view (red), the camera locations (squares) and in black the 3D object locations computed using triangulation for all pairs of views. If the calibration is correct, the distance between the black curves is small. Fig. 7 shows an example where these curves do not coincide.

Figure 5: (Good case) This undistorted image looks correct and the black (triangulated form pairs of views), red (3D estimates) and green (annotations) do coincide well.

2.3.3 Global registration

The camera poses results of the linear solution (initial solution) and hence of the bundle adjustment are defined w.r.t. to the first camera and at a certain scale. In order to map these poses to the global reference system we need to estimate the rigid transformation between the set of 3D points defining the position of the object in the scene and their correspondig location in the global/world reference system. The rigid transformation is defined in term of a rotation, trnaslation and scale. All the camera poses will undertake the same transformation. In other words it means that the reprojection error obtained beforhand with the bundle adjustment will be the same. To do so we run the following command:

```
python scripts/global_registration.py -s setup.json -ps output/bundle_adjustment/ba_poses.json
    ↪ -po output/bundle_adjustment/ba_points.json -l landmarks.json -lg landmarks_global.json
    ↪ -f filenames.json --dump_images
```

2.4 Case studies

2.4.1 Case 1

The low reprojection error suggests us that the optimization produced a set of parameters that minimizes well the loss, however, the undistorted image for one camera (or more) does not look right. There are two reasons for this happen. The first one is that the bounds for the distortion parameters were too generous and hence the optimizer took advatage of them to minimize the loss. The second more subtle one is that the bounds for the intrinsic parameters such that the two focal lengths and the principal points were to narrow. As a consequence, the optimizer used the distortion parameters to compensate.

It is also important to note here that the points do not cover the entire surface of the image. As a result, it is more likely that the distortion function behaves incorrectly on the borders.

```
2021-07-12 09:54:47,638 [root] Optimization took 81 seconds
2021-07-12 09:54:47,651 [root] Average absolute residual: 0.86 over 6316.0 points.
2021-07-12 09:54:48,042 [root] Reprojection errors (mean+-std pixels):
2021-07-12 09:54:48,044 [root]   6-4 n_points=1423: 1.160+-1.168
2021-07-12 09:54:48,046 [root]   6-2 n_points=1486: 1.174+-1.327
2021-07-12 09:54:48,047 [root]   6-3 n_points=1448: 1.733+-1.464
2021-07-12 09:54:48,048 [root]   6-1 n_points=510: 1.051+-1.093
2021-07-12 09:54:48,050 [root]   6-0 n_points=1449: 1.477+-1.371
2021-07-12 09:54:48,050 [root] Reprojection errors (median pixels):
```

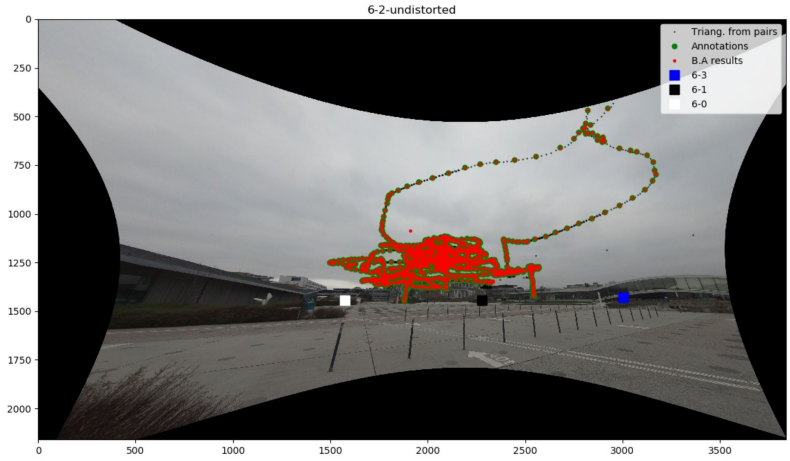


Figure 6: (Case 1) This undistorted image does not look like a proper one. The degree of asymmetry of the distortion function is high which suggests that the bounds for the distortion parameters were too generous or those of the intrinsics too narrow.

```
2021-07-12 09:54:48,052 [root]   6-4 n_points=1423: 0.846
2021-07-12 09:54:48,054 [root]   6-2 n_points=1486: 0.918
2021-07-12 09:54:48,056 [root]   6-3 n_points=1448: 1.317
2021-07-12 09:54:48,057 [root]   6-1 n_points=510: 0.915
2021-07-12 09:54:48,058 [root]   6-0 n_points=1449: 1.114
2021-07-12 09:54:48,059 [root] Visualise all the annotations.
```

2.4.2 Case 2

The reprojection error is relatively high and the black and red curves in Fig. 7 do not coincide well. This usually happens when the optimizer is either stuck on a poor local minimum or it is constrained by the bounds. The bounds were set to:

```
"bounds_cp": [
    0.3, 0.3, 0.3,
    100, 100, 100,
    5, 5, 5, 5,
    0.01, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
  ]
```

```
2021-07-10 09:47:36,960 [root] Optimization took 186 seconds
2021-07-10 09:47:36,970 [root] Average absolute residual: 3.11 over 6315.0 points.
2021-07-10 09:47:37,456 [root] Reprojection errors (mean+-std pixels):
2021-07-10 09:47:37,458 [root]   6-4 n_points=1422: 5.627+-5.252
2021-07-10 09:47:37,460 [root]   6-2 n_points=1487: 5.273+-4.428
2021-07-10 09:47:37,461 [root]   6-3 n_points=1447: 4.695+-4.213
2021-07-10 09:47:37,462 [root]   6-1 n_points=510: 2.784+-2.245
2021-07-10 09:47:37,463 [root]   6-0 n_points=1449: 5.028+-3.893
2021-07-10 09:47:37,463 [root] Reprojection errors (median pixels):
2021-07-10 09:47:37,465 [root]   6-4 n_points=1422: 3.512
2021-07-10 09:47:37,466 [root]   6-2 n_points=1487: 3.908
2021-07-10 09:47:37,468 [root]   6-3 n_points=1447: 3.438
2021-07-10 09:47:37,469 [root]   6-1 n_points=510: 2.107
2021-07-10 09:47:37,470 [root]   6-0 n_points=1449: 3.965
2021-07-10 09:47:37,471 [root] Visualise all the annotations..
```

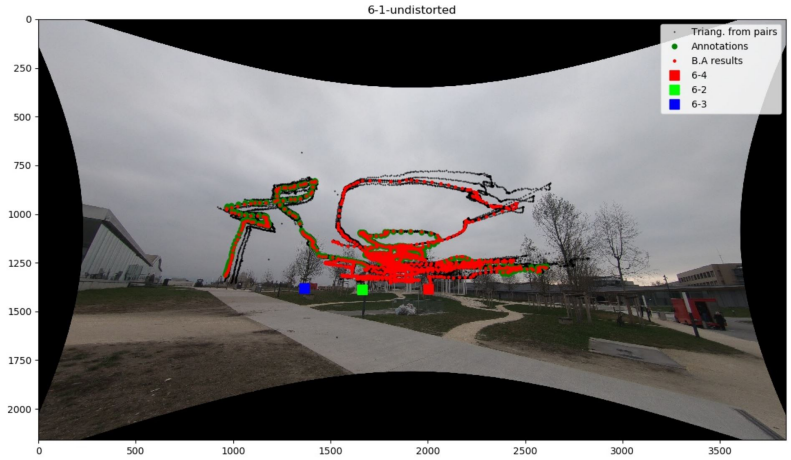


Figure 7: (Case 2) This undistorted image looks correct, however, the black and red points do not coincide well. It's a sign that either the annotations are not correct or the optimization could not converge to a good solution.

The bounds for the rotation and translation parameters of the cameras are most likely large enough to comprise the optimal solution. The bounds for the intrinsic parameters such as the two focal lengths and the optical centers are instead quite narrow so as the one of the distortion parametwers. Since the estimated intrinsic parameters may have errors larger than this, we suggest using larger bounds such as:

```
"bounds_cp": [
  0.3, 0.3, 0.3,
  100, 100, 100,
  400, 400, 400, 400,
  0.5, 0.5, 0.1, 0.1, 0.1, 0, 0, 0, 0, 0, 0, 0, 0, 0
]
```

2.4.3 Case 3

The reprojection error is very low and the undistorted image of Fig. 8 is not correct. It can be noted that there are only few green points (annotations) in the image which suggests that most of them have been considered outliers. From the output messages we can also see that 700 points have been considered outliers. The problem here is that the threshold for the outliers rejections meachnisms were set too low which results in lack of annotations. Clearly, this leads to a lower loss and reprojection errors but also to non plausible parameters.

```
...
2021-07-12 10:22:39,933 [root]   Number of points considered outliers: 697
...
2021-07-12 10:30:00,245 [root] Optimization took 47 seconds
2021-07-12 10:30:00,250 [root] Average absolute residual: 0.41 over 3314.0 points.
2021-07-12 10:30:00,579 [root] Reprojection errors (mean+-std pixels):
2021-07-12 10:30:00,580 [root]   6-4 n_points=243: 1.107+-0.894
2021-07-12 10:30:00,581 [root]   6-2 n_points=1028: 0.584+-0.548
2021-07-12 10:30:00,583 [root]   6-3 n_points=1301: 0.529+-0.654
2021-07-12 10:30:00,584 [root]   6-1 n_points=368: 0.817+-0.543
2021-07-12 10:30:00,584 [root]   6-0 n_points=374: 0.834+-0.835
2021-07-12 10:30:00,585 [root] Reprojection errors (median pixels):
2021-07-12 10:30:00,585 [root]   6-4 n_points=243: 0.826
2021-07-12 10:30:00,587 [root]   6-2 n_points=1028: 0.456
2021-07-12 10:30:00,588 [root]   6-3 n_points=1301: 0.352
2021-07-12 10:30:00,589 [root]   6-1 n_points=368: 0.740
```

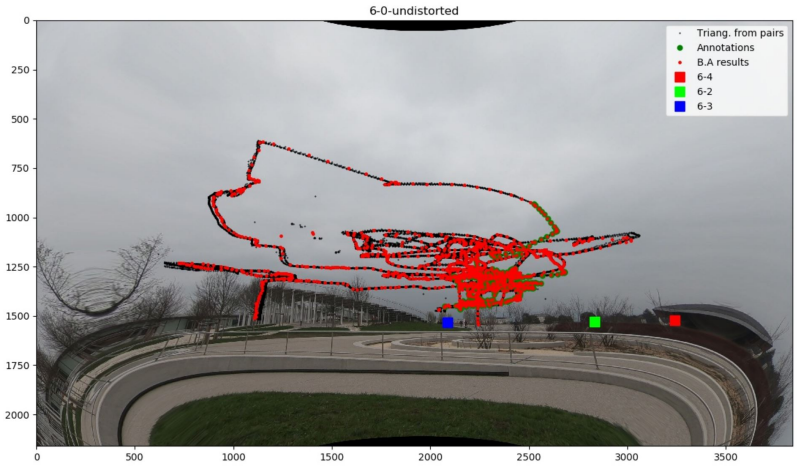


Figure 8: (Case 3) This undistorted image is not correct on the borders and the number of annotations (green points) is very small compared to the the initial set. The problem here are the thresholds for the outlier rejection mechanisms which were set too low.

```
2021-07-12 10:30:00,590 [root]   6-0 n_points=374: 0.641
2021-07-12 10:30:00,591 [root] Visualise all the annotations.
```