Calibration Tutorial for ORB-SLAM3 v1.0

Juan J. Gómez Rodríguez, Carlos Campos, Juan D. Tardós

December 22, 2021

1 Introduction

This document contains a brief explanation of visual and visual-inertial calibration for ORB-SLAM3 v1.0. In this version we introduce a new calibration file format whose main novelties are:

- Improved readability with consistent naming.
- Option for internal rectification of stereo images.
- Option for internal resizing of the input images.

Some examples of use of the new calibration fromat can be found at directory `Examples`. Although we recommend using the new file format, for the user convenience, we maintain compatibility with the file format used in previous versions, whose examples are in directory `Examples_old`.

2 Reference systems and extrinsic parameters

Extrinsic calibration consists of the set of parameters which define how different sensors are geometrically related. The most complex case is the stereo-inertial configuration, shown in figure 1, where we define the following reference frames:

- **World** (`W`): Defines a fixed reference system, whose $z_{\mathtt{W}}$ axis points in opposite direction of the gravity vector $\mathbf{g}$. Translation and yaw are freely set by the SLAM system and remain fixed once initialized. In the pure visual case, the world reference is set to the first camera pose.
- **Body** (`B`): This is the optimizable reference and is attached to the IMU. We assume the gyroscope and the accelerometer share the same reference system. Body pose $\mathbf{T}_{\mathtt{WB}}$ and velocity $\mathbf{v}_{\mathtt{B}}$ expressed in `W` are the optimizable variables.
- **Cameras** ($\mathtt{C}_1$ and $\mathtt{C}_2$): These are coincident with the optical center of the cameras, with $z_{\mathtt{C}}$ pointing forward along the optical axis, $y_{\mathtt{C}}$ pointing down and $x_{\mathtt{C}}$ pointing to the right, both aligned with image directions u and v.

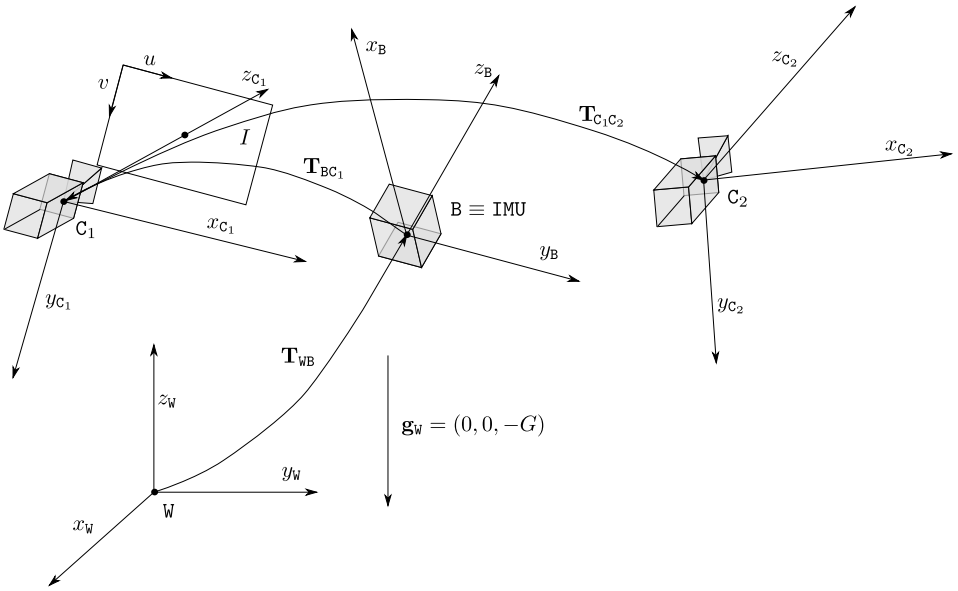

Figure 1: Reference systems defined for ORB-SLAM3 stereo-inertial.

Cameras and body poses relate as:

$$\mathbf{T}_{\mathtt{WC}_1} = \mathbf{T}_{\mathtt{WB}}\mathbf{T}_{\mathtt{BC}_1} \tag{1}$$

$$\mathbf{T}_{\mathtt{WC}_2} = \mathbf{T}_{\mathtt{WB}}\mathbf{T}_{\mathtt{BC}_1}\mathbf{T}_{\mathtt{C}_1\mathtt{C}_2} \tag{2}$$

where, for example, $\mathbf{T}_{\mathtt{WC}_1} \in \mathrm{SE}(3)$ is the homogeneous transformation that passes points expressed in camera one reference to the world reference:

$$\mathbf{x}_{\mathtt{W}} = \mathbf{T}_{\mathtt{WC}_1}\mathbf{x}_{\mathtt{C}_1} \tag{3}$$

The extrinsic parameters that the calibration file needs to provide are:

- Stereo-inertial: $\mathbf{T}_{\mathtt{BC}_1}$ and $\mathbf{T}_{\mathtt{C}_1\mathtt{C}_2}$
- Monocular-inertial: $\mathbf{T}_{\mathtt{BC}_1}$
- Stereo: only $\mathbf{T}_{\mathtt{C}_1\mathtt{C}_2}$, ORB-SLAM3 estimates the pose of the left camera ($\mathbf{B} = \mathbf{C}_1$)
- Monocular: No parameters needed, ORB-SLAM3 estimates the camera pose.

If your camera or dataset provides rectified stereo images, the calibration file only needs to specify the baseline, instead of the full $\mathbf{T}_{\mathtt{C}_1\mathtt{C}_2}$ transformation.

All the extrinsic parameters can be obtained using calibration software such as *Kalibr* [3], as explained in section 4.

3 Intrinsic parameters

Those are calibration parameters which only depend on each sensor itself. Here, we distinguish inertial and visual sensors.

3.1 Camera intrinsic parameters

Depending on camera set-up we will need to provide different calibration parameters. Those can be calibrated using *OpenCV* or *Kalibr* [3]. At ORB-SLAM3 we distinguish three camera types:

- *Pinhole*. The intrinsic parameters to provide are the camera focal length and central point in pixel (f_x, f_y, c_x, c_y), together with the parameters for a radial-tangential distortion model [4] with two or three radial distortion parameters (k_1, k_2, k_3) and two tangential distortion coefficients (p_1, p_2).

- *KannalaBrandt8*. The Kannala-Brandt model [2] is appropriate for cameras with wide-angle and fisheye lenses. The camera parameters to provide are the focal length and central point in pixels $(f_x,\ f_y,\ c_x,\ c_y)$ and four coefficients for the equidistant distortion model (k_1, k_2, k_3, k_4).

- *Rectified*. This camera type is to be used for camera or dataset that provide rectified stereo images. In this case the calibration file only needs to provide $(f_x,\ f_y,\ c_x,\ c_y)$ for the rectified images and the stereo baseline b in meters.

In the case of stereo *pinhole* cameras, ORB-SLAM3 internally rectifies the left and right images using OpenCV's `stereorectify` function. To avoid losing resolution and field-of-view, cameras of type *KannalaBrandt8* are not rectified.

Examples of calibration files for the three camera types can be found at directory `Examples/Stereo-Inertial` in files `Euroc.yaml` (Pinhole), `TUM_VI_512.yaml` (KannalaBrandt8) and `Realsense_D435i.yaml` (Rectified).

3.2 IMU intrinsic parameters

IMU readings (linear acceleration $\widetilde{\mathbf{a}}$ and angular velocity $\widetilde{\boldsymbol{\omega}}$) are affected by measurement noise $(\boldsymbol{\eta}^a, \boldsymbol{\eta}^g)$ and bias $(\mathbf{b}^a, \mathbf{b}^g)$, such as:

$$\widetilde{\mathbf{a}} = \mathbf{a} + \boldsymbol{\eta}^a + \mathbf{b}^a \tag{4}$$

$$\widetilde{\boldsymbol{\omega}} = \boldsymbol{\omega} + \boldsymbol{\eta}^g + \mathbf{b}^g \tag{5}$$

where $\mathbf{a}$ and $\boldsymbol{\omega}$ are the true acceleration (gravity not subtracted) and angular velocity at the body reference `B`. Measurement noises are assumed to follow centered normal distributions, such that:

$$\boldsymbol{\eta}^a \sim \mathcal{N}(\mathbf{0}, \sigma_a^2 \mathbf{I}_3) \tag{6}$$

$$\boldsymbol{\eta}^g \sim \mathcal{N}(\mathbf{0}, \sigma_g^2 \mathbf{I}_3) \tag{7}$$

where σ_a and σ_g are both noise densities, which are characterized in the IMU data-sheet. They need to be provided at the calibration file, with $m/s^2/\sqrt{\text{Hz}}$ and $\text{rad}/s/\sqrt{\text{Hz}}$ units, as shown in listing 1.

```
1  IMU.NoiseAcc:  0.0028  # m/s^1.5
2  IMU.NoiseGyro: 0.00016 # rad/s^0.5
3  IMU.Frequency:     200 # s^-1
```

Listing 1: Noise densities for IMU. Values are from TUM-VI dataset

When integrating the IMU measurements and estimating their covariances, the used noise densities $\sigma_{a,f}$ will depend on the IMU sampling frequency f, which must be provided in the calibration file. This is internally managed by ORB-SLAM3, which computes $\sigma_{a,f} = \sigma_a/\sqrt{f}$

Regarding biases, they are assumed to evolve according to a Brownian motion. Given two consecutive instants i and $i+1$, this is characterized by:

$$\mathbf{b}^a_{i+1} = \mathbf{b}^a_i + \boldsymbol{\eta}^a_{\text{rw}} \quad \text{with } \boldsymbol{\eta}^a_{\text{rw}} \sim \mathcal{N}(\mathbf{0}, \sigma^2_{a,\text{rw}}\mathbf{I}_3) \tag{8}$$

$$\mathbf{b}^g_{i+1} = \mathbf{b}^g_i + \boldsymbol{\eta}^g_{\text{rw}} \quad \text{with } \boldsymbol{\eta}^g_{\text{rw}} \sim \mathcal{N}(\mathbf{0}, \sigma^2_{g,\text{rw}}\mathbf{I}_3) \tag{9}$$

where $\sigma_{a,\text{rw}}$ and $\sigma_{g,\text{rw}}$ need to be supplied in the calibration file, as shown in listing 2.

```
IMU.AccWalk:  0.00086  # m/s^2.5
IMU.GyroWalk: 0.000022 # rad/s^1.5
```

Listing 2: Random walk standard deviations for IMU biases. Values are from TUM-VI dataset.

It is common practice to increase the random walk standard deviations provided by the IMU manufacturer (say multiplying them by 10) to account for unmodelled effects and improving the IMU initialization convergence.

4 Calibration example of Realsense D435i with *Kalibr*

Here we show an example of visual-inertial calibration for a Realsense D435i device in monocular-inertial configuration. We will assume the realsense library (`https://github.com/IntelRealSense/librealsense`) has been previously installed. For the calibration, we will use the well-established *Kalibr* open source software [1, 3], which can be found at `https://github.com/ethz-asl/kalibr`. We will follow the next steps:

- First, for simplicity, we will use *Kalibr* CDE. Just download it from `https://github.com/ethz-asl/kalibr/wiki/downloads`

- As calibration pattern, we will adopt the the proposed April tag. You will find templates for this pattern in the previous download page. You need to print it and make sure pattern is just scaled and proportions are not modified. Update the configuration .yaml file for this pattern. You can find a template corresponding to the A0 size in the same download page.

- For recording the calibration sequences, you can use the SDK from realsense or our provided recorder, simply running:

```
./Examples/Calibration/recorder_realsense_D435i ./Examples/
Calibration/recorder
```

Listing 3: Run visual-inertial recorder

  Make sure recorder directory exists and contains empty folders cam0 and IMU.

- Use python script to process IMU data and interpolate accelerometer measurements to synchronize it with the gyroscope.

```
python3 ./Examples/Calibration/python_scripts/process_imu.py ./
Examples/Calibration/recorder/
```

Listing 4: Run visual-inertial recorder

- Nest, you will need to convert this dataset to a rosbag with rosbag creater from *Kalibr*. Run the next command from *Kalibr* CDE repository:

```
./kalibr_bagcreater --folder /path_to_ORB_SLAM3/Examples/
Calibration/recorder/. --output-bag /path_to_ORB_SLAM3/Examples/
Calibration/recorder.bag
```

Listing 5: Run visual-inertial recorder

4.1 Visual calibration

This step solves for the intrinsic camera parameters as well as relative camera transformation for stereo sensors. We will go through the next steps:

- Following previous section steps, record a dataset with slow motion to avoid image blur, as well as good lighting to have a low exposure time. It could be even possible to move the pattern instead of the camera, but make sure it is not deformed. An example calibration sequence can be found at `https://youtu.be/R_K9-O4ool8`.

- Use a pattern as big as possible, so you can fill the whole image with the pattern as far as possible, easing the camera focus. There not exists a minimum size for this dataset, just try to have different points of view and make sure all pixels see the pattern multiple times.

- You could decrease the frame rate for this calibration, to make *Kalibr* solution faster, but this is not a requirement.

- Finally run the calibration from the *Kalibr* CDE repository:

```
./kalibr_calibrate_cameras --bag /path_to_ORB_SLAM3/Examples/
Calibration/recorder.bag --topics /cam0/image_raw --models
pinhole-radtan --target /path_to_ORB_SLAM3/Examples/Calibration/
april_to_be_updated.yaml
```

Listing 6: Run visual calibration

For our monocular Realsense D435i, we compare *Kalibr* and factory calibration at table 1. Typical values of reprojection error for a successful visual calibration are those whose mean is close to zero pixels ($|\mu| < 10^{-4}$) and its standard deviation $\sigma < 0.3$.

4.2 Inertial calibration

Once calibrated visual parameters, we will continue with the inertial calibration:

- First, record a new dataset with fast motion, trying to excite accelerometer and gyroscope along all their axes. Try to keep always most of the pattern visible in the image, so *Kalibr* can accurately estimate its relative pose. Pattern should remain

Table 1: Comparative factory/*Kalibr* calibration for Realsense D435i.

	Factory	*Kalibr*
f_x	382.613	381.69830045 ± 0.47867208
f_y	382.613	381.6587096 ± 0.48696699
c_x	320.183	321.58237544 ± 0.40096812
c_y	236.455	236.20193592 ± 0.38600065
k_1	0	-0.00469988 ± 0.00171615
k_2	0	0.00110469 ± 0.00196003
r_1	0	-0.00029279 ± 0.00028587
r_2	0	0.00066225 ± 0.00034486

fixed during this calibration. A suitable sequence for visual-inertial calibration can be found at `hhttps://youtu.be/4XkivVLw5k4`

- You will need to provide the noise density and bias random walk for the IMU sensor. You can find these values from IMU datasheet (BMI055 for Realsense D435i) or estimate them using data acquired while keeping the IMU steady for a long period of time.

- Finally run the calibration from the *Kalibr* CDE repository:

```
./kalibr_calibrate_imu_camera --bag /path_to_ORB_SLAM3/Examples/
Calibration/recorder.bag --cam /path_to_ORB_SLAM3/Examples/
Calibration/camera_calibration.yaml --imu /path_to_ORB_SLAM3/
Examples/Calibration/imu_intrinsics.yaml --target /
path_to_ORB_SLAM3/Examples/Calibration/april_to_be_updated.yaml
```

Listing 7: Run visual calibration

For our monocular Realsense D435i, we compare *Kalibr* and factory calibration at table 2.

Table 2: Comparative factory/*Kalibr* calibration for Realsense D435i.

	Factory	*Kalibr*
$\mathbf{T}_{\mathtt{BC}}$	$\begin{pmatrix} 1 & 0 & 0 & -0.005 \\ 0 & 1 & 0 & -0.005 \\ 0 & 0 & 1 & 0.0117 \\ 0 & 0 & 0 & 1 \end{pmatrix}$	$\begin{pmatrix} 0.9999 & 0.0003 & -0.0135 & -0.0015 \\ -0.0002 & 0.9999 & 0.0054 & 0.0004 \\ 0.0135 & -0.0054 & 0.9999 & 0.0201 \\ 0 & 0 & 0 & 1 \end{pmatrix}$

4.3 Launch ORB-SLAM3

With all these calibration parameters, you can finally create your .yaml file to be used along with ORB-SLAM3. Finally, run ORB-SLAM launcher as:

```
./Examples/Monocular-Inertial/mono_inertial_realsense_D435i
Vocabulary/ORBvoc.txt Examples/Monocular-Inertial/RealSense_D435i.
yaml
```

Listing 8: Run real-time monocular-inertial SLAM

5 Reference for the new calibration files

This section summarizes all parameters that are required by ORB-SLAM3 in any of its stages, including intrinsic and extrinsic calibration parameters, ORB extraction parameters and visualization settings. All this configuration parameters must be passed to ORB-SLAM3 in a `yaml` file. In this type of files, data is stored as a `<Key, value>` pair, encoding the parameter name and its value respectively. We now define all parameters that ORB-SLAM3 accepts, their types and whether they are required or optional.

5.1 General calibration parameters

These are general calibration parameters:

- *File.version = "1.0"* **[REQUIRED]**: specifies that the new calibration file format is being used.
- *Camera.type* (string) **[REQUIRED]**: specifies the camera type beeing used. Must take one of the following values:
  - *PinHole* when using pinhole cameras.
  - *KannalaBrandt8* when using fish-eye cameras with a Kannala-Bradt calibration model.
  - *Rectified* when using a stereo rectified pinhole camera.
- *Camera.height, Camera.width* (int) **[REQUIRED]**: input image height and width.
- *Camera.newHeight, Camera.newWidth* (int) [OPTIONAL]: If defined, ORB-SLAM3 resizes the input images to the new resolution specified, recomputing the calibration parameters as needed.
- *Camera.fps* (int) **[REQUIRED]**: frames per second of the video sequence.
- *Camera.RGB* (int) **[REQUIRED]**: specifies if the images are: BGR (0) or RGB (1). It is ignored if the images are grayscale.
- *System.thFarPoints* (float) [OPTIONAL]: if defined, specifies the maximum depth in meters allowed for a point. Points farther than this are ignored.

5.2 Camera intrinsic parameters

We define the *Camera1* as the monocular camera (in monocular SLAM) or the left camera (in stereo SLAM). Its intrinsic parameters correspond to:

- *Camera1.fx, Camera1.fy, Camera1.cx, Camera1.cy* (float) **[REQUIRED]**: corresponds with the intrinsic calibration parameters of the camera 1.

If *Camera.type* is set to *PinHole*, you should specify:

- *Camera1.k1, Camera1.k2* (float) **[REQUIRED]**: corresponds to the radial distortion coefficients.

- *Camera1.p1, Camera1.p2* (float) **[REQUIRED]**: corresponds to the tangential distortion coefficients.
- *Camera1.k3* (float) [OPTIONAL]: sometimes a third radial distortion parameter is used. You can specify it with this parameter.

If *Camera.type* is set to *KannalaBrandt8*, you should specify:

- *Camera1.k1, Camera1.k2, Camera1.k3, Camera1.k4* (float) **[REQUIRED]**: Kannala-Brandt distortion coefficients.

If using a stereo camera, specify also the parameters for *Camera2* (right camera), unless *Camera.type* is set to *Rectified* in which case both cameras are assumed to have the same parameters.

5.3 Stereo parameters

The following parameter is required for stereo configurations:

- *Stereo.ThDepth* (float) **[REQUIRED]**: it is the number of the stereo baselines we use to classify a point as close or far. Close and far points are treated differently in several parts of the stereo SLAM algorithm.

If *Camera.type* is set to *Rectified*, you need to add the following parameter:

- *Stereo.b* (float) **[REQUIRED]**: stereo baseline in meters.

If you are using a non-rectified stereo, you need to provide:

- *Stereo.T_c1_c2* (cv::Mat) **[REQUIRED]**: relative pose between stereo cameras.

If you are using stereo fish-eye cameras (i.e. *Camera.type* is set to *KannalaBrandt8*), you also need to specify the overlapping area between both images:

- *Camera1.overlappingBegin, Camera2.overlappingBegin* (int) **[REQUIRED]**: start column of the overlapping area.
- *Camera1.overlappingEnd, Camera2.overlappingEnd* (int) **[REQUIRED]**: end column of the overlapping area.

5.4 Inertial parameters

When using a inertial sensor, the user must define the following parameters:

- *IMU.NoiseGyro* (float) **[REQUIRED]**: measurement noise density of the gyroscope.
- *IMU.NoiseAcc* (float) **[REQUIRED]**: measurement noise density of the accelerometer.
- *IMU.GyroWalk* (float) **[REQUIRED]**: Random walk variance of the gyroscope.

- *IMU.AccWalk* (float) **[REQUIRED]**: Random walk variance of the accelerometer.

- *IMU.Frequency* (float) **[REQUIRED]**: IMU frequency.

- *IMU.T_b_c1* (cv::Mat) **[REQUIRED]**: Relative pose between *IMU* and *Camera1* (i.e. the transformation that takes a point from *Camera1* to *IMU*).

- *IMU.InsertKFsWhenLost* (int) [OPTIONAL]: Specifies if the system inserts KeyFrames when the visual tracking is lost but the inertial tracking is alive.

5.5 RGB-D parameters

The following parameter is required for RGB-D cameras:

- *RGBD.DepthMapFactor* (float) **[REQUIRED]**: factor to transform the depth map to real units.

5.6 ORB parameters

The following parameters are related with the ORB feature extraction:

- *ORBextractor.nFeatures* (int) **[REQUIRED]**: number of features to extract per image.

- *ORBextractor.scaleFactor* (float) **[REQUIRED]**: scale factor between levels in the image pyramid.

- *ORBextractor.nLevels* (int) **[REQUIRED]**: number of levels in the image pyramid.

- *ORBextractor.iniThFAST* (int) **[REQUIRED]**: initial threshold to detect FAST corners.

- *ORBextractor.minThFAST* (int) **[REQUIRED]**: if no features are found with the initial threshold, the system tries a second time with this threshold value.

5.7 Atlas parameters

These parameters define if we load/save a map from/to a file:

- *System.LoadAtlasFromFile* (string) [OPTIONAL]: file path where the map to load is located.

- *System.SaveAtlasToFile* (string) [OPTIONAL]: destination file to save the map generated.

5.8 Viewer parameters

These are some parameters related to the ORB-SLAM3 user interface:

- *Viewer.KeyFrameSize* (float) **[REQUIRED]**: size in which the KeyFrames are drawn in the map viewer.
- *Viewer.KeyFrameLineWidth* (float) **[REQUIRED]**: line width of the KeyFrame drawing.
- *Viewer.GraphLineWidth* (float) **[REQUIRED]**: line width of the covisibility graph.
- *Viewer.PointSize* (float) **[REQUIRED]**: size of the MapPoint drawing.
- *Viewer.CameraSize* (float) **[REQUIRED]**: size in which the current camera is drawn in the map viewer.
- *Viewer.CameraLineWidth* (float) **[REQUIRED]**: line width of the current camera drawing.
- *Viewer.ViewpointX, Viewer.ViewpointY, Viewer.ViewpointZ, Viewer.ViewpointF* (float) **[REQUIRED]**: starting view point of the map viewer.
- *Viewer.imageViewScale* (float) OPTIONAL: resize factor for the image visualization (only for visualization, not used in the SLAM pipeline).

References

[1] Paul Furgale, Joern Rehder, and Roland Siegwart. Unified temporal and spatial calibration for multi-sensor systems. In *IROS*, pages 1280–1286. IEEE, 2013.

[2] Juho Kannala and Sami S Brandt. A generic camera model and calibration method for conventional, wide-angle, and fish-eye lenses. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 28(8):1335–1340, 2006.

[3] Joern Rehder, Janosch Nikolic, Thomas Schneider, Timo Hinzmann, and Roland Siegwart. Extending kalibr: Calibrating the extrinsics of multiple IMUs and of individual axes. In *Proc. IEEE Int. Conf. Robotics and Automation (ICRA)*, pages 4304–4311, 2016.

[4] Richard Szeliski. *Computer vision: algorithms and applications*. Springer Verlag, London, 2011.