Spectacular AI SDK

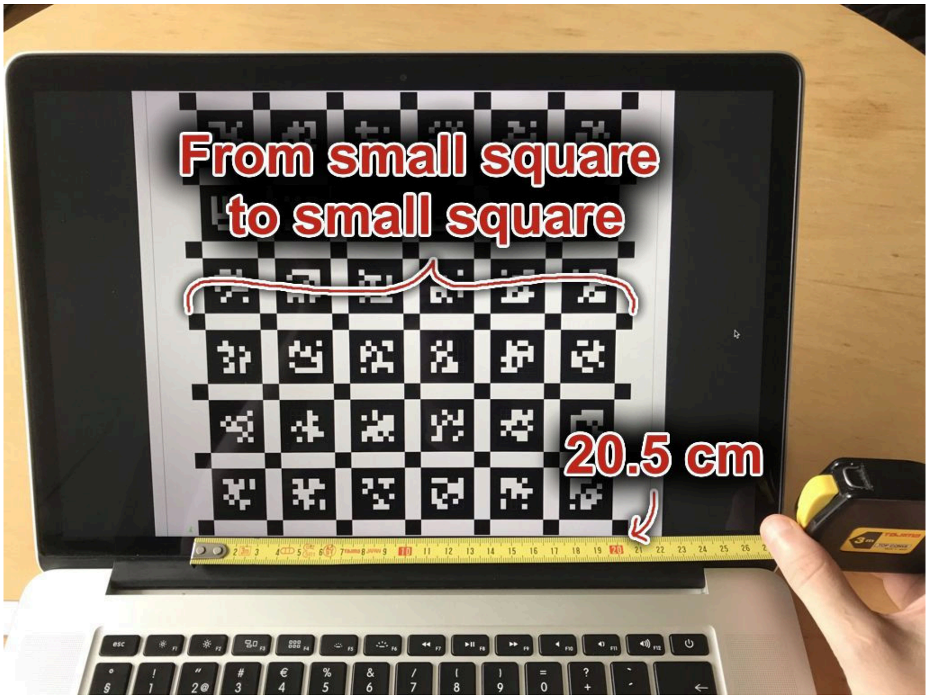
Calibration manual

This document describes the calibration and hardware aspects that are relevant to the Spectacular AI SDK. The target audience are users of the *core SDK* who wish to perform the camera calibration and sensor integration manually. For the *wrapper SDKs* (e.g., OAK-D or RealSense wrappers), this is not required as the factory calibration parameters are provided by the devices and automatically used by the SDK.

To start using manual calibration with Spectacular AI SDK, it is recommended to start with a *calibration test* in collaboration with Spectacular AI engineers. This provides a reference that helps the integration with the customers' camera calibration pipeline. The calibration test is described in the first section of this document.

This document also includes a self-contained description of the mathematical calibration models and coordinate systems used by the SDK, intended as a reference for integration.

Calibration test



Example: simple AprilGrid calibration using a computer screen: measure and divide by 8.1 to get `tagSize`.

In the calibration test, a single device individual is calibrated in collaboration with Spectacular AI engineers.

The recommended way to perform the test is using the built-in recording function in Spectacular AI SDK to record a calibration sequence. If the SDK has not yet been integrated to enable this, it is also possible to record camera (and preferably also IMU) data using other software and convert the data to a suitable format in collaboration with Spectacular AI.

A calibration test sequence should view a calibration pattern from multiple angles. The calibration pattern may be displayed on a computer screen or printed on a flat surface. Larger screens may improve accuracy of the calibration. In the calibration test, the IMU is also calibrated from the data and, consequently, the **calibration target must be stationary** and the device should be moved around it (see “Recommended calibration movements” below for more detailed instructions)

The following patterns are supported:

- AprilGrid (link to PDF)
- Checkerboard

The size of the calibration pattern should be defined in the YAML format supported by the *Kalibr* software. For example

```
target_type: 'aprilgrid'
tagCols: 6               # number of apriltags
tagRows: 6               # number of apriltags
tagSize: 0.037654321     # size of apriltag, edge to edge [m]
tagSpacing: 0.3          # ratio of space between tags to tagSize
```

The recommended resolution for calibration image data is the maximum sensor resolution. However, if this cannot be achieved, the resolution should be at least 400x400. If IMU data is included, it should be recorded at 50Hz or more (500Hz recommended). If IMU data cannot be recorded, then approximate IMU-to-camera extrinsics (see the sections below for more information) should be provided. If the calibrated system includes stereo camera pairs, the frames recorded from stereo camera pairs must be synchronized.

The recorded sequence should be sent to Spectacular AI, who will calibrate the device based on the data and send a calibration file, which can be used as a reference and example for the subsequent steps in SDK integration.

Recommended calibration trajectories

For a good but simple calibration sequence, we recommend the following systematic approach, which can be divided into the following phases

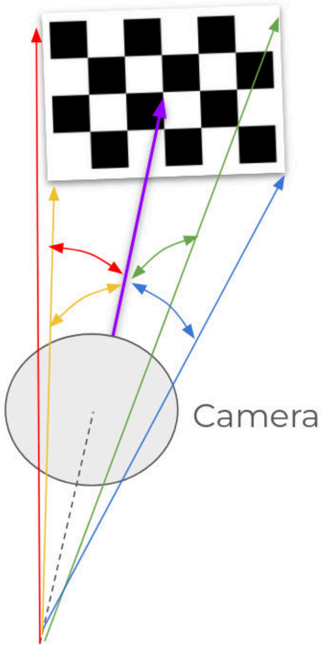
1. Intrinsic coverage (for each camera in the system)
2. IMU-to-camera extrinsic calibration
3. Stereo extrinsic calibration (between each overlapping camera pair - in stereo and multi-camera systems only)

Each of the above phase(s) can be further divided into steps, which are detailed below. The following points are also good guidelines for the entire calibration:

- Not too slow, not too fast: the entire multi-phase sequence requires between 1 and 3 minutes to complete
- Minimize motion blur: slow movements
- After recording, view the video and check if it matches these instructions. If not, try again. It does not take many rounds of practice to get it right, but it's not common to get it right on the first

- Adjust placement of the calibration target and lighting or exposure settings in the camera if needed
- In most frames, the calibration target should span ~50% of the height of the image

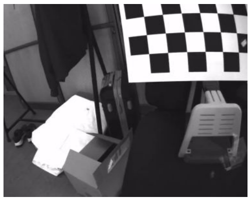
Phase 1: *Intrinsic coverage*



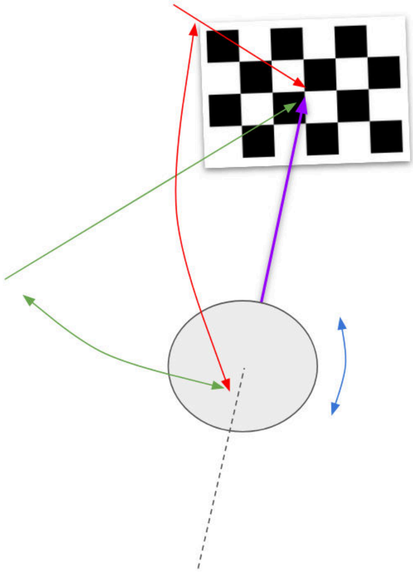
Tilt the camera to point the principal axis (image center) towards each corner. Return to target center (purple) after each step:

- Top-left corner
- Top-right corner
- Bottom-right corner
- Bottom-left corner

Order does not matter. Ideally, the pattern should move over the edges of the image (example below for bottom-left state)



Phase 2: *IMU calibration*



Rotate camera along each "Euler" axis, keeping the principal axis pointed at the center of the calibration target

1. Pitch
2. Yaw
3. Roll

In each step rotate

- -45°
- +90° (opposite 45° extreme)
- -45° (return to center)

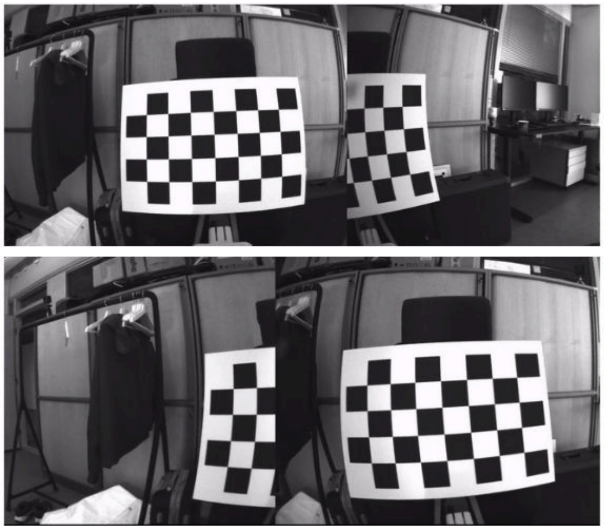
Note: keep the calibration target in focus and "orbit" the camera around it for pitch & yaw.

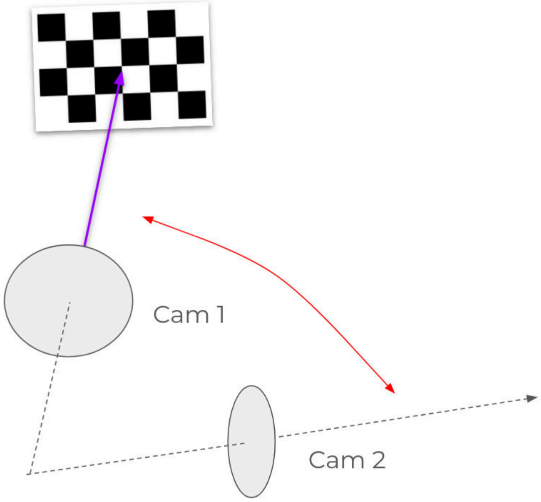
Phase 3: *Stereo extrinsic calibration*

Starting with camera 1 pointing at center, tilt to point camera 2 at center. Then tilt back to point camera 1 towards the center again.

Repeat 2x (we need back and forth transitions between the views below).

This should be done for each overlapping camera pair in a multi-camera system.





Note: the figure above illustrates a high-vergence camera system (the principal axes of camera 1 and camera 2 are not parallel). In a more typical stereo camera system, you should be able to see the entire pattern in both camera views simultaneously.

Calibration specification

File format

The calibration is provided to the SDK in JSON format. Spectacular AI will provide a reference as the results of a successful *calibration test*. An example calibration file for a stereo-camera-IMU-system is given below.

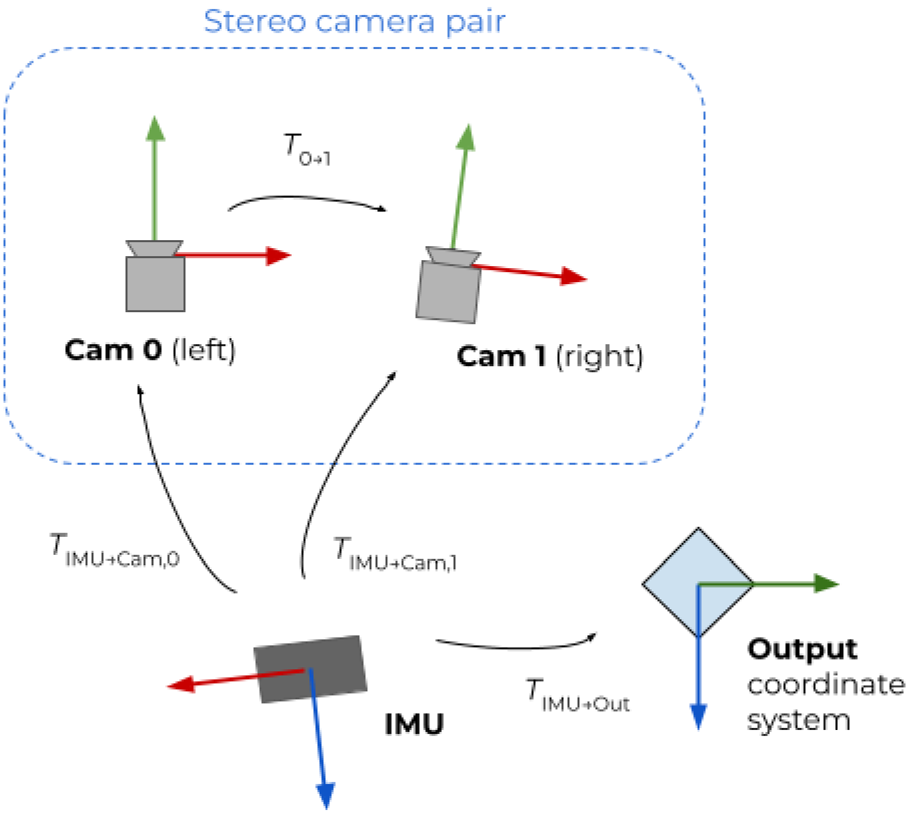
```
{
  "cameras": [
    {
      "imageWidth": 1280,
      "imageHeight": 800,
      "focalLengthX": 689.9600212721717,
      "focalLengthY": 689.7791814512566,
      "principalPointX": 625.7728119663589,
      "principalPointY": 406.30847173743695,
      "model": "kannala-brandt4",
      "distortionCoefficients": [-0.042199872,-0.0024873,-0.0156296,0.008040966],
      "imuToCamera": [

[-0.007597321889990516,-0.9999685028233531,-0.0022965324560711986,0.003925088167884679],

[-0.028027852548307086,-0.0020827542464345594,0.9996049727848895,-0.002080025490845079],
        [-0.9995782711632094,0.007658687614133075,-0.028011146395656494,-0.06311860979590438],
        [0.0,0.0,0.0,1.0]
      ]
    },
    {
      "imageWidth": 1280,
      "imageHeight": 800,
      "focalLengthX": 689.6159071698686,
      "focalLengthY": 689.3776100206506,
      "principalPointX": 637.155260132079,
      "principalPointY": 410.031637138216,
      "model": "kannala-brandt4",
      "distortionCoefficients": [-0.0381701,-0.015025785,0.0042020,-0.0005575143],
      "imuToCamera": [

[-0.008900660156368811,-0.9999585078949196,-0.0019392620624486545,-0.12881566945954037],

[-0.014472758680460995,-0.0018103137813196835,0.9998936253523123,-0.0004115181854138228],
        [-0.9998556483337772,0.008927779823628468,-0.014456045187493105,-0.06294064508330674],
        [0.0,0.0,0.0,1.0]
      ]
    }
  ],
  "imuToOutput": [
    [0.06859197197751811,-0.9973466692339874,-0.024387758160742287,-0.0],
    [0.995903321632435,0.06700802688203558,0.06071654053764488,0.0],
    [-0.05892126391820337,-0.028452516606581855,0.9978570734113344,0.04],
    [0.0,0.0,0.0,1.0]
  ]
}
```

The colors in the above markup listing highlight the role of the different parameters as follows:

- Intrinsic camera calibration
  - left camera
  - right camera
- Extrinsic calibration parameters
  - Left IMU-to-camera matrix ($T_{\text{IMU}\to\text{Cam},0}$)
  - Right IMU-to-camera matrix ($T_{\text{IMU}\to\text{Cam},1}$)
  - IMU to output reference matrix ($T_{\text{IMU}\to\text{Out}}$)

The meaning of each parameter is explained in more detail in the following sections.

Extrinsic coordinate systems


Stereo camera pair
$T_{0\to1}$
Cam 0 (left)
Cam 1 (right)
$T_{\text{IMU}\to\text{Cam},0}$
$T_{\text{IMU}\to\text{Cam},1}$
IMU
$T_{\text{IMU}\to\text{Out}}$
Output coordinate system

The extrinsic calibration models the geometry of the device, from the point of view of the IMU-camera system. In Spectacular AI SDK, the system is assumed to consist of an IMU sensor and one or more cameras. The first two cameras in the system may form a stereo camera pair.

The extrinsic calibration is specified as 4x4 homogeneous coordinate transformation matrices. For a system with one stereo camera pair, one must specify:

- $T_{\text{IMU→Cam,0}}$: IMU-to-camera matrix for the left camera
- $T_{\text{IMU→Cam,1}}$: IMU-to-camera matrix for the right camera

Optionally, a matrix $T_{\text{IMU→Out}}$ (IMU-to-output matrix) can be given to specify the reference point and local coordinate axis directions that the SDK will use for its output (see the coordinate *Coordinate conversion example* at the end of this document). The IMU-to-output matrix also specifies the reference point for *external pose inputs* in the core SDK.

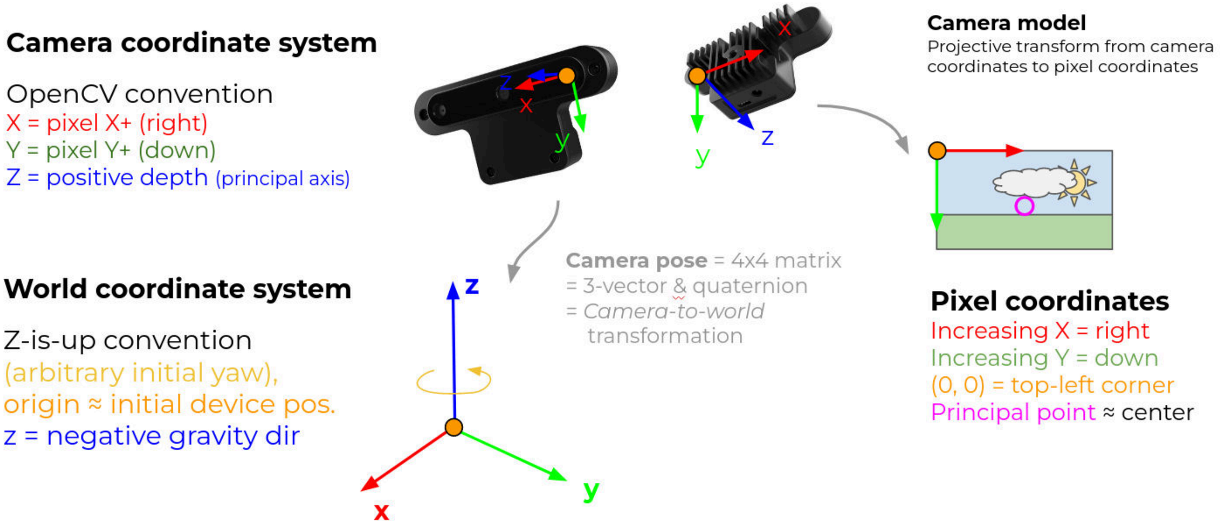
The IMU-to-output can also be convenient in a case where the SDK output is compared to an external reference system, such as VIVE trackers. Specifying the VIVE tracker reference point and orientation using $T_{\text{IMU→Out}}$ allows direct comparison of the poses returned by the two systems with SE(3) or SE(2)-aligned metrics.

Another way to control the IMU-to-output matrix is using the optional parameter `outputCameraPose` flag in the `vio_config.yaml` file. If set to true (which is also the default in SDK wrappers, but not the core SDK), the local output coordinate system is the primary camera coordinate system (in OpenCV convention, see below for more details).

Coordinate system conventions

Spectacular AI SDK uses the following coordinate conventions, which are also illustrated in the image below

- **World coordinate system**: Right-handed Z-is-up
- **Camera coordinate system**: *OpenCV* convention:
  X =right, Y = down, Z = forward.
- **IMU coordinate system**: assumed to be right-handed (and use SI units)
- **Local output coordinate system** (“reference point”). Can be specified in the calibration data. Default: IMU coordinates



Camera coordinate system

OpenCV convention
X = pixel X+ (right)
Y = pixel Y+ (down)
Z = positive depth (principal axis)

Camera model
Projective transform from camera coordinates to pixel coordinates

Camera pose = 4x4 matrix
= 3-vector & quaternion
= *Camera-to-world* transformation

World coordinate system

Z-is-up convention
(arbitrary initial yaw),
origin ≈ initial device pos.
z = negative gravity dir

Pixel coordinates
Increasing X = right
Increasing Y = down
(0, 0) = top-left corner
Principal point ≈ center

Intrinsic calibration

Intrinsic calibration models the optics of the cameras and possible digital deformations of the image in the camera pipeline. Spectacular AI SDK uses a small-aperture approach that is typical for computer vision. Depth of field or chromatic aberration effects are not modeled..

All models have four common parameters: (f_x, f_y, c_x ,c_y), which describe the "focal length" (exact interpretation depends on the camera model) and principal point. They should be given as separate JSON fields (see JSON format description above), e.g., `focalLengthX`. The rest of the model parameters are given in a `distortionCoefficients` array and the type of the camera model in the `model` field.

The SDK currently supports the following types of calibration models (see *Mathematical details* for more info):

1. Brown-Conrady radial-tangential models
   a. Undistorted pinhole: no distortion coefficients (only (f_x, f_y, c_x ,c_y))
   `"model": "pinhole"`
   b. 3-coefficient radial variant:
   `"model": "pinhole",`
   `"distortionCoefficients": [`k_1, k_2, k_3`]`

c. 5-coefficient variant, a.k.a. “pinhole-radtan”
   `"model": "brown-conrady",`
   `"distortionCoefficients":` [k_1, k_2, p_1, p_2, k_3, 0, 0, 0]
d. 8-coefficient variant (Brown-Conrady / Inverse Brown-Conrady)
   `"model": "brown-conrady",`
   `"distortionCoefficients":` [$k_1, k_2, p_1, p_2, k_3, k_4, k_5, k_6$]
e. 14-coefficient variant (Brown-Conrady / Inverse Brown-Conrady)
   `"model": "brown-conrady",`
   `"distortionCoefficients":` [$k_1, k_2, p_1, p_2, k_3, k_4, k_5, k_6, s_1, s_2, s_3, s_4, \tau_x, \tau_y$]

2. Kannala-Brandt
   a. Kannala-Brandt-4, radially symmetric
      `"model": "kannala-brandt4",`
      `"distortionCoefficients":` [k_0, k_1, k_2, k_3]
   b. Full Kannala-Brandt-18 (ask us for a reference implementation)

3. OpenCV “omnidir” camera model (Mei & Rivers, 2007):
   `"model": "omnidir",`
   `"distortionCoefficients":` [$k_1, k_2, s, \xi, p_1, p_2$]

The values of the calibration parameters depend on the manner the image is rescaled, cropped or undistorted by the camera pipeline before being input to the SDK. If these aspects are changed, the calibration needs to be modified accordingly.

One possible non-trivial modification to the images is undistortion, which may be applied to the images before they are given as input to the SDK. After undistortion, the images are typically assumed to obey the simple undistorted pinhole camera model. This may also be performed simultaneously with *stereo rectification*, which can also effectively rotate the camera coordinate systems (see *Stereo rectification* the *Mathematical details* section for more information).

Accuracy requirements

Extrinsics and intrinsics

The transformation between the two stereo cameras, $T_{0\to1}$, must be accurate and consistent with the intrinsic camera calibration parameters. It is recommended to compute this quantity with a calibration system that jointly optimizes these parameters (see the section below for more information). The average residual reprojection error should be less than 0.3 pixels (RMSE).

The transformation between IMU and the stereo camera system is not required to be highly accurate in indoor use cases. The error in IMU-to-camera matrix should be

- less than 1 degree in orientation (less than 3 degrees for indoor use cases)
- less than 5% in translation (or less than 3 millimeters, whichever is greater)

The as-designed IMU-to-camera extrinsics from a CAD model can typically be assumed to be sufficiently accurate for indoor and ground vehicle use cases. The camera calibration, including intrinsic calibration and the extrinsic calibration within stereo camera pairs must be performed per device individual.

Less accurate IMU-to-camera extrinsic information can be combined with more accurate stereo extrinsics by using the mathematical identities described in *Mathematical details*.

Synchronization

The IMU and camera timestamps should be synchronized to a precision of 1 millisecond. Delays up to 10 ms can be tolerated by enabling an online time-shift calibration feature in the SDK, but may reduce accuracy and robustness. The timestamps of the camera frames given to the SDK should represent the midpoint of exposure (taking rolling shutter readout time into account, if applicable).

Hardware recommendations

Camera specifications

Item	Minimum	High-quality
Sensor type	Rolling shutter	Global shutter (e.g. OV9282)
Field-of-View (HFoV)	70°	160°
Resolution	240p	800 x 800
Aspect ratio	16:9	1:1
Video encoding (for data recording purposes)	h264/h265 @ 400kBps	h264/h265 @ 7MBps

The recommended FPS depends on the use case but is generally between 10 FPS and 50 FPS. For most indoor use cases, 24 - 30 FPS is a good starting point.

The use of auto-focus or variable-focus lenses is not recommended as changing the focus affects the calibration parameters in a hard-to-predict manner. Optical or electronic image stabilization (OIS/EIS) should not be used.

Auto-exposure and auto-white-balance (in case of color data) may be used. The pixel data that is fed to the SDK can be linear or gamma-corrected. If possible, the exposure settings should be tuned to minimize exposure time (and thus motion blur) even when this results in high time-varying noise. However, static noise should be minimized.

IMU settings

Category	Minimum	High-quality
Frequency	50 Hz	500 Hz
Calibration	Uncalibrated or HW temperature calibrated	

Especially in high-vibration environments, the IMU range should be configured to a relatively high value (e.g., [-8g, 8g] for acc.) to avoid saturation. The built-in digital low-pass filters should be configured to values that are reasonable for the selected sampling rates (ODR).

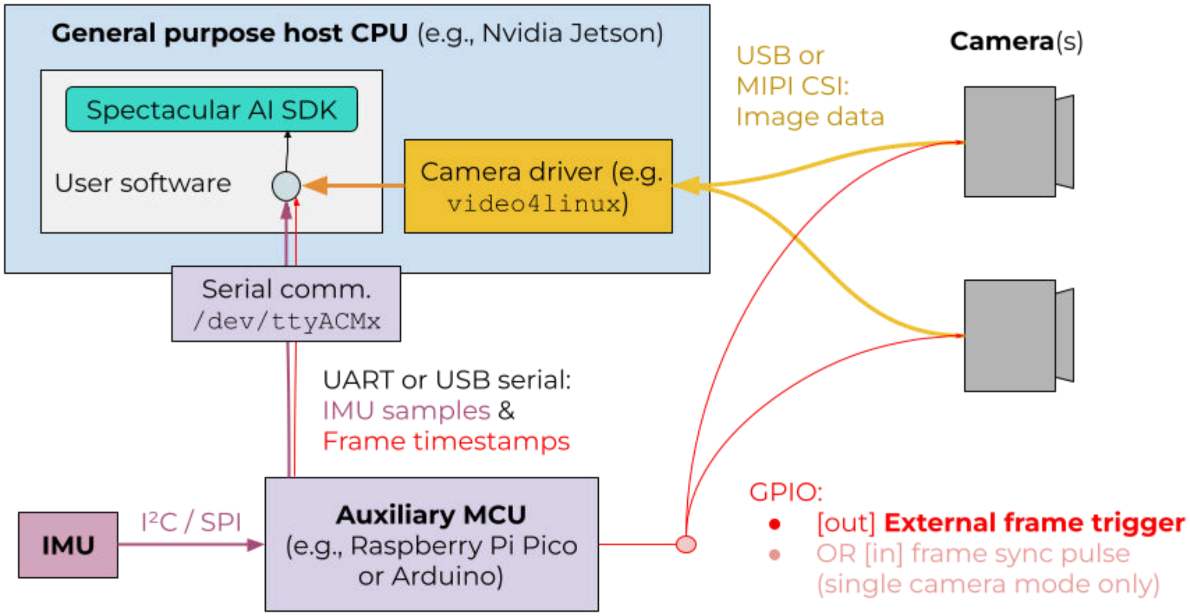
Synchronization mechanisms

The following high-level hardware designs can be used to achieve submillisecond synchronization accuracy. In all of the options below, a camera module that supports either external trigger pulses (FSIN, typically found in global shutter modules) or frame synchronization output pulses (frame start or frame end, FSYNC) is required. For multi-camera systems, an external trigger (or vendor-specific synchronization mechanism together with FSYNC) is needed.

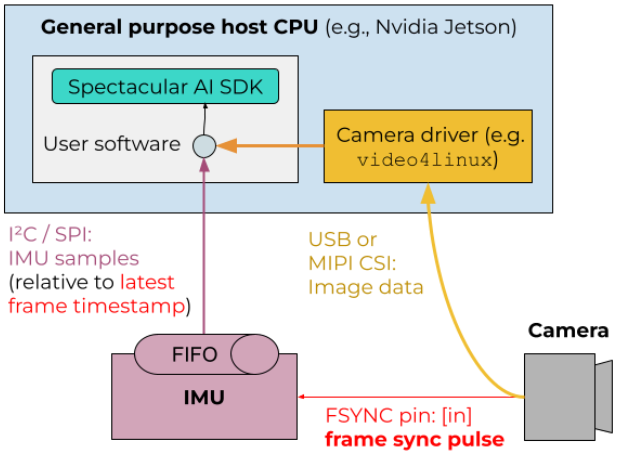
It is relatively rare for camera modules sold for Raspberry Pi or Nvidia Jetson, let alone less popular development kits like Texas Instruments, to support these features, especially if you have other requirements for the camera such as a wide FoV lens or global shutter sensor.

In all of the options below, the exposure time should be read from the camera driver to compute the exposure midpoint timestamp that should be fed to the Spectacular AI SDK.

Auxiliary MCU


General purpose host CPU (e.g., Nvidia Jetson)
Spectacular AI SDK
User software
Camera driver (e.g. video4linux)
USB or MIPI CSI: Image data
Camera(s)
Serial comm. /dev/ttyACMx
UART or USB serial: IMU samples & Frame timestamps
IMU
I²C / SPI
Auxiliary MCU (e.g., Raspberry Pi Pico or Arduino)
GPIO:
● [out] External frame trigger
● OR [in] frame sync pulse (single camera mode only)

One mechanism that enables high synchronization accuracy is using a separate microcontroller or other real-time processor for reading the IMU and generating the trigger pulse for the cameras (or reading the FSYNC pulse generated by the camera). Examples of suitable microcontrollers include Raspberry Pi Pico (RP2040) and Arduino devices. These are also relatively straightforward to include directly into custom PCB designs that can also include the IMU and other sensors. An example of such hardware is shown in this video.



IMU FSYNC pin

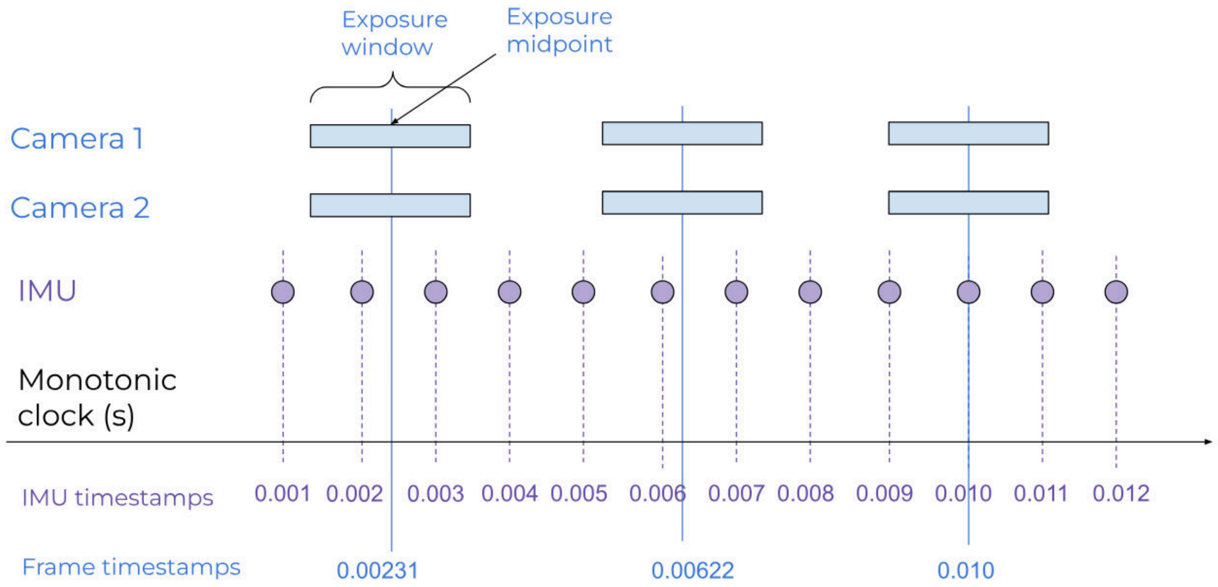
Some IMU sensors also feature a separate FSYNC pin for hardware synchronization with cameras (typically originally intended for Electronic Image Stabilization). This method does not require a separate MCU, but an IMU that supports the FSYNC feature is required.

Synchronization explainer

In the context of the Spectacular AI SDK, *synchronization* means that events have the same time base, i.e., the timestamps **originate from the same monotonic clock.** In practice, achieving good IMU-camera synchronization often requires support from the hardware.

Basic: IMU and camera in the same monotonic clock

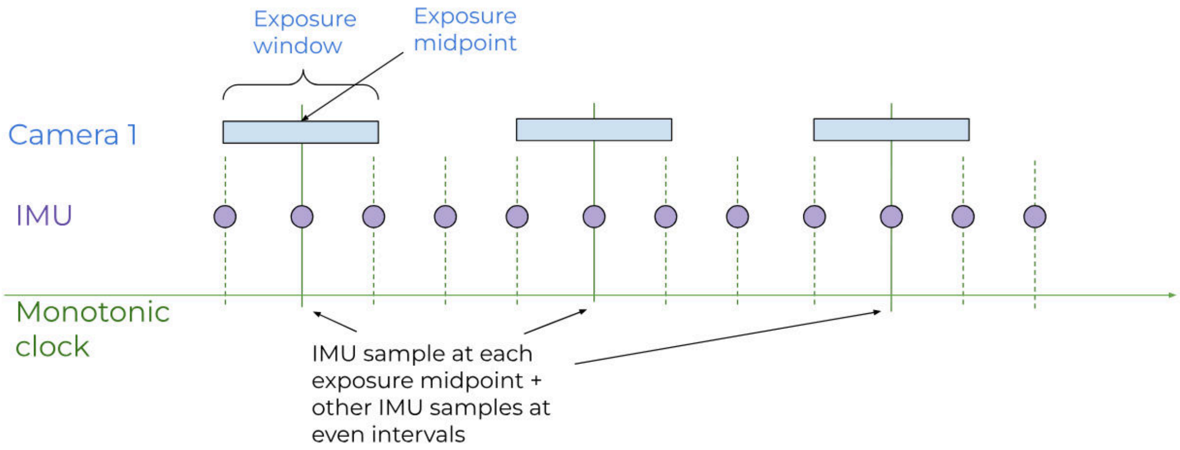
An example of this is given in the figure below: One of the camera frames and one IMU sample happen to have the same timestamp, 0.010s, which means that the IMU sample represents the same time instance as the exposure midpoint in the camera frame. However, in this basic synchronization, there is no requirement that all camera frames would match some IMU sample.



For simplicity, the above figure assumes that the accelerometer and gyroscope are always read simultaneously into one “IMU sample”. The example also shows a stereo camera setup, where the two cameras are correctly triggered at the same time and use the same exposure time.

Advanced: IMU samples at exposure midpoints

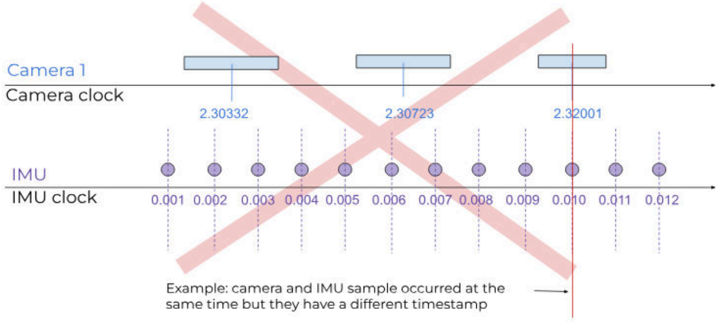
Ideally, IMU samples and frames should be timed so that, for each exposure midpoint, there also exists an IMU sample whose timestamp coincides with that of the frame. However, this is an advanced configuration, which may further improve VIO accuracy, but is not mandatory.

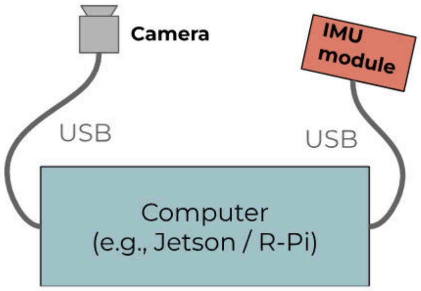

Exposure window
Exposure midpoint
Camera 1
IMU
Monotonic clock
IMU sample at each exposure midpoint + other IMU samples at even intervals

List of common misconfigurations

There are several possible hardware-software-firmware combinations that can result in correct synchronization. However, in practice, there are also a lot of systems with incorrect synchronization, which are practically impossible to fix without hardware changes. Below, we outline some common mistakes

Unsynchronized IMU

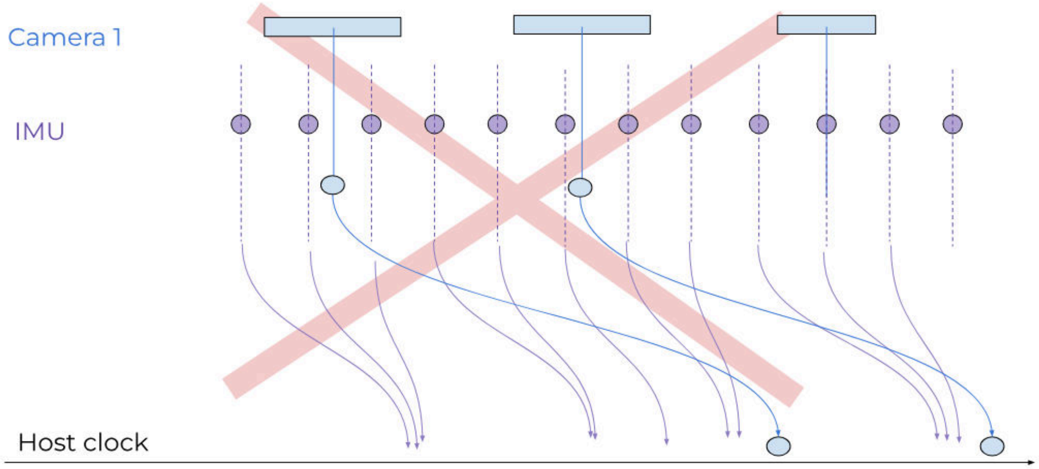




The most common misconfiguration is a system where the IMU and cameras have separate clocks, whose timestamps have no direct relationship. This adds two problems which are very hard to fix in real-time software:

1. There is an unknown time offset between the IMU and the camera timestamps (typically in the order of 100ms after simple hacks), which varies between runs
2. The offset slowly changes over time due to the drift of the monotonic clocks, at the rate of a few milliseconds per hour

Arrival timestamps

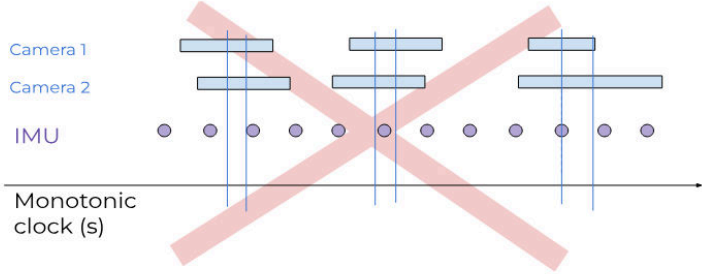


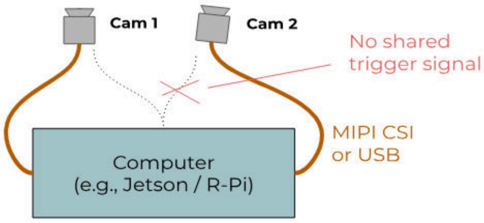
Timestamp assigned using `clock()` when the IMU sample or frame is read from a buffer on the host computer

Another common mistake is not using a timestamp that represents when an IMU sample actually occurs or when a frame was captured (exposure midpoint), but using the clock of the host machine to assign a timestamp when a sample or frame is *read* from a buffer, which introduces large and random delays in the timestamps, which also depend on the sensor (IMU or camera). This is fatal to VIO performance.

In the lucky case, this might be just an easily fixable software issue, but it is unfortunately more common that, once the issue is discovered, it appears to the programmer that the correct timestamps are actually not available at all or out of sync (see previous misconfiguration) and fixing this requires firmware or hardware updates.

Unsynchronized stereo

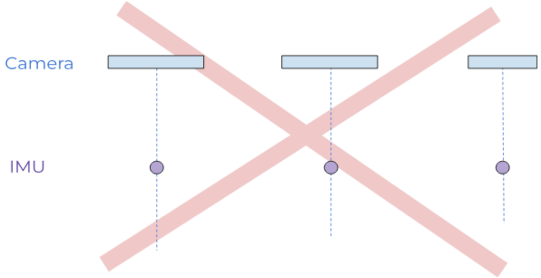




One common issue in stereo and multi-camera setups is that the different cameras are not triggered at the same time. This can be tolerated in certain use cases, but in the most typical case, where the intention is to benefit from stereo vision, this severely degrades the accuracy of the system, not just from the point of view of VIO, but also stereo depth estimation in general (except for the special case where the camera does not move, or moves very slowly).

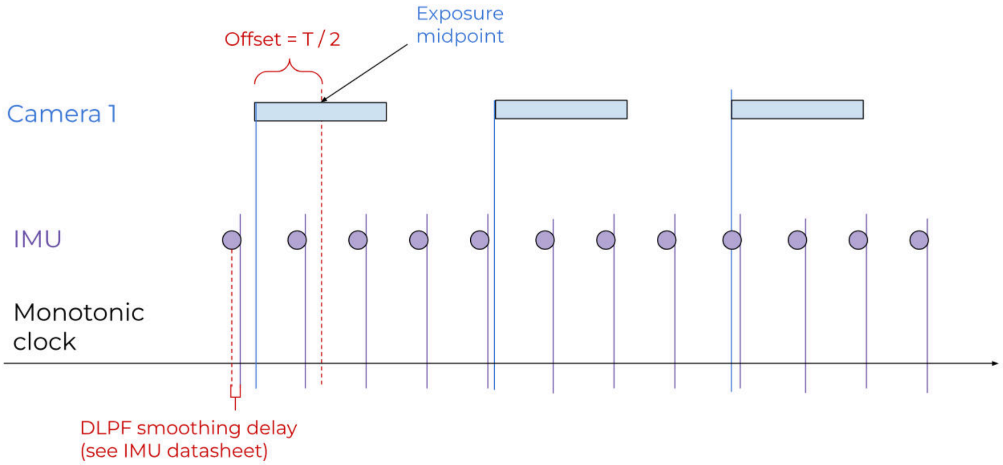
Misconception: One IMU sample per frame (low IMU rate)

Synchronization does *not* mean that there should be one IMU sample per camera frame. IMU rate is typically a lot higher than the camera FPS. However, there is some benefit for some IMU samples exactly coinciding with exposure midpoints (see the "Advanced" case below)



Missing offsets

Typically, a camera system does not directly output the exposure midpoint, and to compute it, one has to combine data from multiple sources. For example, the exposure midpoint can be computed by adding half of the *exposure time* to the *trigger timestamp*. In case of rolling shutter sensors, it is also necessary to add half of the *readout* time.

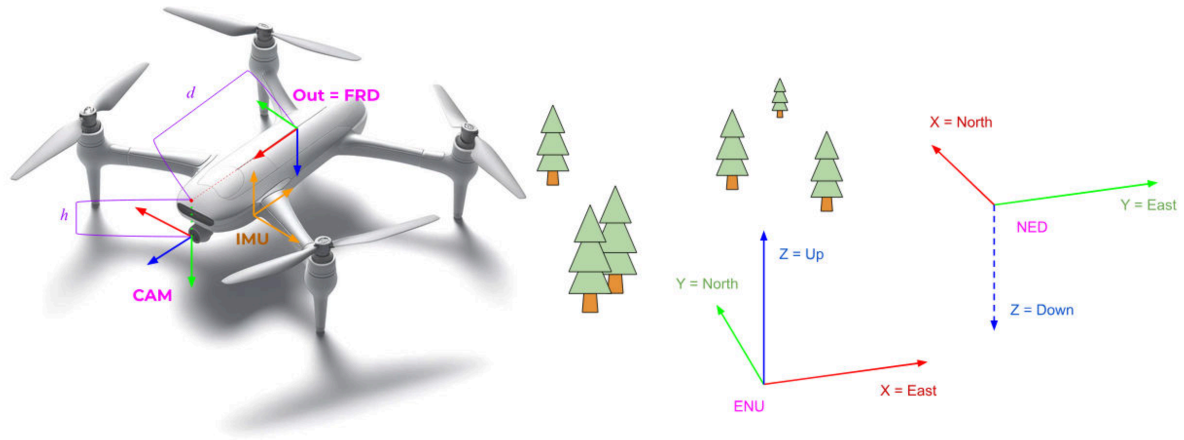


Similarly, the IMU, or the real-time processor that reads it, may not natively output the best timestamp for the IMU sample since IMU samples also represent a short time window, whose duration depends on the IMU settings, in particular the DLPF configuration. You can refer to the IMU datasheet for these values.

Luckily, this category of problems is typically fixable in the host-side software.

Mathematical details

Coordinate conversion example



Consider an example, where we have a small UAV with a front-facing camera and we would like to convert from the Spectacular AI GNSS-VIO output poses (IMU-to-ENU) to an FRD-to-NED convention used by, e.g., PX4.

Local coordinate system conversion: First, we need to define our IMU-to-output matrix. There are two options:

1. We can directly extract it from CAD file, which determines the exact position of the IMU, w.r.t., the reference point for the FRD frame (e.g., center of gravity)
2. It is not known accurately, let's use a tape measure:
    a. Measure the longitudinal distance between the camera and the reference point (*d*) and the distance in the altitude direction (*h*), in meters. Let us assume that the lateral distance is nearly zero
    b. The CAM-to-OUT matrix ($T_{\text{Cam}\rightarrow\text{Out}}$) is:

```
[
        [0, 0, 1, d],   # Row 1: OUT X (forward) = CAM Z + offset d
        [1, 0, 0, 0],  # Row 2: OUT Y (right)   = CAM X
        [0, 1, 0, h],   # Row 3: OUT Z (down)    = CAM Y + offset h
        [0, 0, 0, 1]
]
```

    c. Extract the "`imuToCamera`" matrix (for the only camera) from the `calibration.json` file (assuming it has been automatically computed by a calibration tool).

d. Compute: $T_{\text{IMU→Out}} = T_{\text{Cam→Out}} \cdot T_{\text{IMU→Cam}}$

Then, $T_{\text{IMU→Out}}$ should be written to the `calibration.json` file as the `imuToOutput` field (see the *File format* section above for reference). After this step, the local reference frame of the Spectacular AI output poses (and GNSS-VIO poses) should be the FRD frame. Also add `outputCameraPose: False` in the `vio_config.yaml` file, to ensure this is the case, independent of the parameter sets.

World coordinate system conversion: World coordinate system conversions are not a part of the Spectacular AI SDK and must always be performed manually. For example, an ENU-to-NED conversion that does not change the origin of the system is the matrix ($T_{\text{ENU→NED}}$):

```
[
        [0,  1,  0,  0],   # Row 1: NED X = North =  ENU Y
        [1,  0,  0,  0],   # Row 2: NED Y = East  =  ENU X
        [0,  0, -1,  0],   # Row 3: NED Z = Down  = -ENU Z
        [0,  0,  0,  1]
]
```

The Spectacular AI pose matrix ($T_{\text{Out→ENU}}$) can then be transformed into NED convention as:

$$T_{\text{Out→NED}} = T_{\text{ENU→NED}} \cdot T_{\text{Out→ENU}}$$

The full conversion formula is

$$T_{\text{Out→NED}} = T_{\text{ENU→NED}} \cdot T_{\text{IMU→ENU}} \cdot T_{\text{IMU→Out}}^{-1}$$

where the local conversion part ($T_{\text{IMU→ENU}}$ -> $T_{\text{Out→ENU}}$) is handled by the SDK if the `imuToOutput` is correctly configured in `calibration.json`.

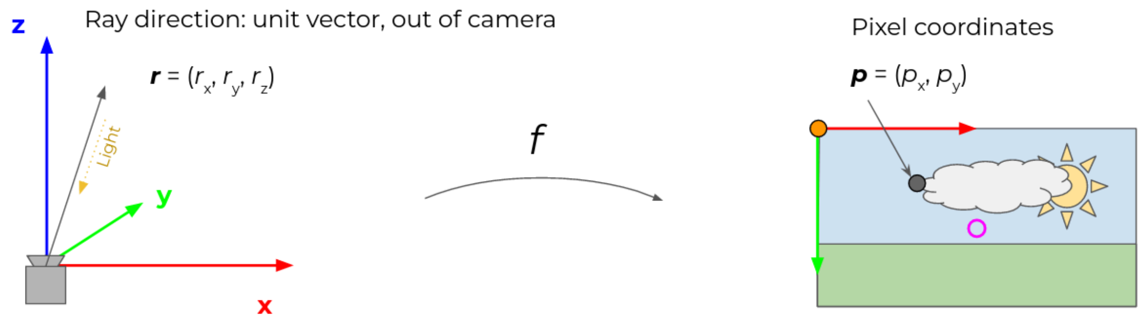
The global coordinate system conversion can, in this case, also be applied only to the orientation as:

$$R_{\text{Out→NED}} = R_{\text{ENU→NED}} \cdot R_{\text{Out→ENU}}$$

where each rotation matrix *R* in the formula represents the top-left 3x3 corner of the corresponding *T* matrix.

Camera models

Spectacular AI SDK utilizes a simplified camera model typical for computer vision: the origin of the camera coordinate system acts as a single point-like center of projection. The light captured by the camera is thought to arrive directly at this point from different directions. The incident direction of a ray of light is assumed to determine the (fractional) coordinates of the pixel it contributes to.


z
Ray direction: unit vector, out of camera
$\boldsymbol{r} = (r_x, r_y, r_z)$
Light
y
x
f
Pixel coordinates
$\boldsymbol{p} = (p_x, p_y)$

Camera coordinate system

OpenCV convention
X = pixel X+ (right)
Y = pixel Y+ (down)
Z = positive depth (principal axis)

Camera model

$f: \boldsymbol{r} \mapsto \boldsymbol{p}$

Light coming from the direction $-\boldsymbol{r}$ is assumed to be recorded at pixel $\boldsymbol{p}$.

Image coordinate system

Increasing X = right
Increasing Y = down
(0, 0) = top-left corner
Principal point ≈ center

The intrinsic calibration model is a function that maps an outbound ray direction $\boldsymbol{r} = (r_x, r_y, r_z)$ to pixel $\boldsymbol{p}$, that is, light coming from the direction opposite to $\boldsymbol{r}$ is assumed to be contributing to the pixel with coordinates $\boldsymbol{p} = (p_x, p_y)$.

The camera model function f_θ: $(r_x, r_y, r_z) \mapsto (p_x, p_y)$ depends on a vector of parameters θ, which are determined by a calibration procedure. One of the simplest widely used models is the undistorted pinhole camera model with 4 parameters $\theta = (f_x, f_y, c_y, c_x)$, defined as

$$p_x = r_x / r_z \cdot f_x + c_x$$
$$p_y = r_y / r_z \cdot f_y + c_y$$

In a stereo camera system, the calibration typically involves simultaneously optimizing, $T_{L \to R}$, the extrinsic transformation between the cameras and the calibration parameters θ_L, θ_R for each camera in the stereo camera pair.

Brown-Conrady

Applied in practice by first computing $x = p_x$ and $y = p_y$ using the pinhole model above and then applying the distortion coefficients (p_1, p_2, k_1, k_2, k_3, k_4, k_5, k_6) as follows

$$p_x' = x\,C + 2\,p_1\,x\,y + p_2(r^2 + 2\,x^2)$$
$$p_y' = y\,C + p_1(r^2 + 2\,y^2) + 2\,p_2\,x\,y$$

where $r^2 = x^2 + y^2$ and $C = (1 + k_1 r^2 + k_2 r^4 + k_3 r^6) \,/\, (1 + k_4 r^2 + k_5 r^4 + k_6 r^6)$.

Kannala-Brandt

Applied by first converting the ray direction $\boldsymbol{r}$ to polar coordinates θ, φ, then applying

$$(x', y') = (r(\theta) + \Delta_r(\theta, \varphi))\,\boldsymbol{u}_r(\varphi) + \Delta_t(\theta, \varphi)\,\boldsymbol{u}_\varphi(\varphi)$$

the radial and tangential directions are

$$\boldsymbol{u}_r(\varphi) = (\cos(\varphi), \sin(\varphi)) \qquad \text{and} \qquad \boldsymbol{u}_\varphi(\varphi) = (-\sin(\varphi), \cos(\varphi)),$$

and the distortion terms are defined from the 18 coefficients (k_0, k_1, k_2, k_3, l_1, l_2, l_3, i_1, i_2, i_3, i_4, m_1, m_2, m_3, j_1, j_2, j_3, j_4) as

$$r(\theta) = \theta\,(1 + k_0\theta^2 + k_1\theta^4 + k_2\theta^6 + k_3\theta^8)$$
$$\Delta_r(\theta, \varphi) = (l_1\theta + l_2\theta^3 + l_3\theta^5) \cdot (i_1 \cos\varphi + i_2 \sin\varphi + i_3 \cos 2\varphi + i_4 \sin 2\varphi)$$
$$\Delta_t(\theta, \varphi) = (m_1\theta + m_2\theta^3 + m_3\theta^5) \cdot (j_1 \cos\varphi + j_2 \sin\varphi + j_3 \cos 2\varphi + j_4 \sin 2\varphi).$$

In the Kannala-Brandt-4 model, the other terms vanish and the model simplifies to

$$(x', y') = r(\theta)\,\boldsymbol{u}_r(\varphi)$$

Finally, the pinhole intrinsics are applied as

$$p_x = x' \cdot f_x + c_x$$
$$p_y = y' \cdot f_y + c_y.$$

Implementation note: Trigonometric functions can mostly be avoided in the computations. The full Kannala-Brandt model can be applied as

$$\theta = \cos^{-1}(r_z / \mathrm{sqrt}(r_x^2 + r_y^2 + r_z^2))$$
$$(c, s) = \texttt{safeNormalize}(r_x, r_y) \qquad = (r_x, r_y) / \mathrm{sqrt}(r_x^2 + r_y^2) = (\cos(\varphi), \sin(\varphi))$$
$$c' = 1 - 2s^2 \qquad = \cos(2\varphi)$$
$$s' = 2\,s\,c \qquad = \sin(2\varphi)$$

$$t = \theta^2$$
$$r(\theta) = \theta\ (1 + t(k_0 + t(k_1 + t(k_2 + tk_3))))$$
$$\Delta_\mathrm{r}(\theta, \varphi) = \theta\ (l_1 + t(l_2 + tl_3)) \cdot (i_1\, c + i_2\, s + i_3\, c' + i_4\, s')$$
$$\Delta_\mathrm{t}(\theta, \varphi) = \theta\ (m_1 + t(m_2 + tm_3)) \cdot (j_1\, c + j_2\, s + j_3\, c' + j_4\, s')$$
$$\boldsymbol{u}_\mathrm{r}(\varphi) = (\mathrm{c}, \mathrm{s})$$
$$\boldsymbol{u}_\varphi(\varphi) = (\mathrm{-s}, \mathrm{c})$$

Formally, $(\cos(\varphi), \sin(\varphi)) = (r_\mathrm{x}, r_\mathrm{y}) / \mathrm{sqrt}(r_\mathrm{x}^2 + r_\mathrm{y}^2)$ but some care needs to be taken to avoid the singularity at $(r_\mathrm{x}, r_\mathrm{y}) = (0, 0)$.

Stereo rectification

If the images are stereo rectified before being given to the SDK, special care needs to be taken to ensure that the image plane rotations used in the rectification process are correctly applied to the extrinsic matrices.

Before rectification, the calibrated stereo-camera-IMU system consists of the extrinsic matrices $T_{\mathrm{IMU}\to\mathrm{Cam0}}$, $T_{\mathrm{IMU}\to\mathrm{Cam1}}$ and the camera models f_0, f_l.

The rectification operation outputs the image plane rotation matrices R_0, R_l and a pinhole camera model f_pin. The pixels in the rectified images then obey the camera model

$$f_i' : \boldsymbol{r} \mapsto \boldsymbol{p}' = f_\mathrm{pin}(R_i \cdot \boldsymbol{r}), \qquad i\mathrm{=0 \ or \ }i\mathrm{=1}$$

It is then possible to define the undistortion function as

$$h_i : \boldsymbol{p'} \mapsto \boldsymbol{p} = f_i(\boldsymbol{r}) = f_i(R_i^\mathrm{T} \cdot f_\mathrm{pin}^{-1}(\boldsymbol{p}')),$$

where the inverse projection can be defined as

$$f_\mathrm{pin}^{-1} : (p_\mathrm{x}, p_\mathrm{y}) \mapsto ((p_\mathrm{x} - c_\mathrm{x})f_\mathrm{x}, (p_\mathrm{y} - c_\mathrm{y})f_\mathrm{y}, 1).$$

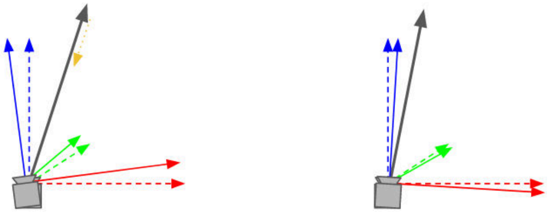
The rectified images I_i' are formed from the original image I_i as

$$I_i'(\boldsymbol{p}') = I_i(\boldsymbol{p}) = I(h_i(\boldsymbol{p}'))$$

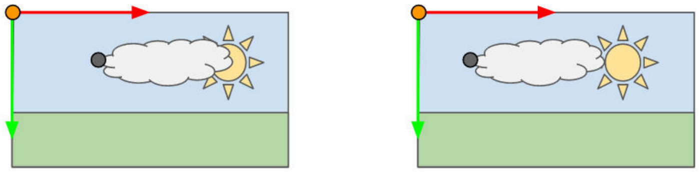
If the rectified images I' are used as input to the SDK, the calibration data should define:

- Undistorted pinhole camera model f_{pin} for both cameras
- IMU-to-camera matrices modified as $T_{\text{IMU}\rightarrow\text{Cam},i}' = g(R_i) \cdot T_{\text{IMU}\rightarrow\text{Cam},i}$

where *g* creates a 4x4 transformation matrix from a 3x3 rotation matrix as described in the figure below.



Original (dashed) vs rectified (solid) IMU-to-camera matrices



Rectification rotation

$R: \boldsymbol{r} \mapsto \boldsymbol{r}'$

Stereo rectification induces a rotation that is applied to the ray direction before the pinhole camera model f_{pin}.

The full model for the *rectified* pixel coordinates $\boldsymbol{p}'$ is

$f': \boldsymbol{r} \mapsto \boldsymbol{p}' = f_{pin}(R \cdot \boldsymbol{r})$

if using the original IMU-to-cam. matrix and

$f_{rect}: \boldsymbol{r} \mapsto \boldsymbol{p}' = f_{pin}(\boldsymbol{r})$

with the rectified matrix T' = g(R) · T, where

$$g(R) = \begin{pmatrix} R & \\ & 1 \end{pmatrix}$$

Relationships between the extrinsic matrices

To combine an accurate stereo extrinsic matrix $T_{0\rightarrow 1}$ with "as-designed" or otherwise approximate IMU-to-camera extrinsics, first determine the IMU-to-camera extrinsics for the left camera, $T_{0\rightarrow 1} \cdot T_{\text{IMU}\rightarrow\text{Cam},}$, and then compute

$$T_{\text{IMU}\rightarrow\text{Cam},1} = T_{0\rightarrow 1} \cdot T_{\text{IMU}\rightarrow\text{Cam},0}$$

where the dot symbol denotes matrix multiplication.

The following identities may also be useful in other similar scenarios (the superscript "-1" denotes matrix inversion):

- $T_{\text{IMU}\rightarrow\text{Cam},0} = T_{0\rightarrow 1}^{-1} \cdot T_{\text{IMU}\rightarrow\text{Cam},1}$
- $T_{0\rightarrow 1} = T_{\text{IMU}\rightarrow\text{Cam},1} \cdot T_{\text{IMU}\rightarrow\text{Cam},0}^{-1}$
- $T_{1\rightarrow 0} = T_{0\rightarrow 1}^{-1}$

.