Photometric Calibration applied to DSOL on UAV

Yuchen Zhang

January 30, 2023

Contents

I Geometric Calibration

Geometric calibration aims to calibrate 3 properties in the stereo system:

1. distortion of the lens (that cause lines in 3D to become curves in 2D image)
2. intrinsic Parameter that specified how 3D point is projected onto image space
3. extrinsic parameter that specifies transform between cameras

DSOL assumes a stereo calibrated image pairs that have only translation along the baseline between the cameras. Therefore we need to obtain the camera parameters and apply rectification to the raw images.

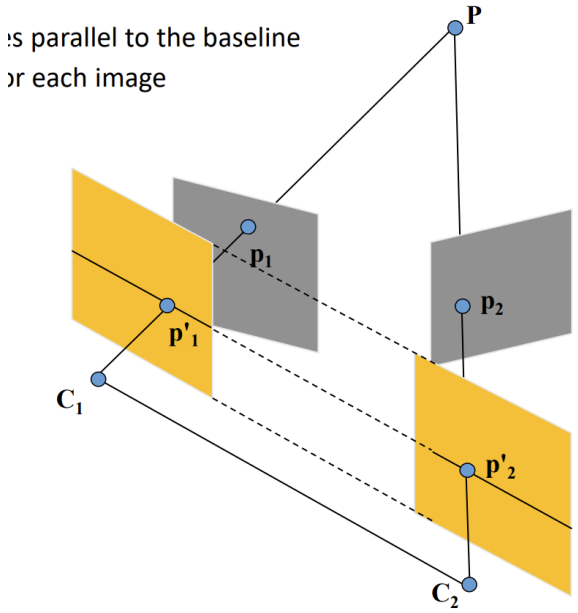

Figure 1: transform image such that only translation along baseline exist. Source: UZH Slides[1]

I Camera Parameter and Extrinsics

We used Kalibr to do extrinsic calibration. The procedure is:

1. Install Kalibr and source its setup.bash file.
2. Record bag files containing images from both cameras shooting the AprilGrid target board. (and IMU if want to calibrate transformation to IMU as well)
3. Provide the target yaml file. Be sure to validate the tagSize and tagSpacing Parameter. Below is the target file I used for the large apriltag board in lab.

apriltag_4x6.yaml

```
target_type: 'aprilgrid' #gridtype
tagCols: 4 #number of apriltags
tagRows: 6 #number of apriltags
tagSize: 0.1495 #size of apriltag, edge to edge [m]
tagSpacing: 0.177257525083612 #ratio of space between tags to tagSize
```

4. Perform camchain calibration. Here is an example:

```
source ~/kalibr_workspace/devel/setup.bash

rosrun kalibr kalibr_calibrate_cameras --bag apriltag.bag --topics /sync/cam0/image_raw
    /sync/cam1/image_raw --models pinhole-fov pinhole-fov --target aprilgrid_4x6.yaml
```

   I used distortion model pinhole-fov because I am calibrating a camera with large Field of View(thus large distortion). Also can use pinhole-equi.

5. If calibrate with IMU, provide the IMU yaml file. Here is what I used for the VN-100 IMU

```
rostopic: /sync/imu/imu

update_rate: 200.0 #Hz

# Borrowed from https://github.com/KumarRobotics/msckf_vio/issues/22#issuecomment-445096680

# an alternative is from github.com/ethz-asl/kalibr/issues/111

# Accelerometers

accelerometer_noise_density: 0.02 # Noise density (continuous-time)

accelerometer_random_walk: 0.0004 # Bias random walk

# Gyroscopes

gyroscope_noise_density: 0.010 # Noise density (continuous-time)

gyroscope_random_walk: 8.0e-06 # Bias random walk
```

6. Run calibrate imu cam similarly in Kalibr.

II Rectification

I rectified the images with the image undistort ROS package from ETH. Refer to attached code for details.

To use this package, you have to interpret their "first camera" parameter as the right camera, and "second camera" as left camera.

A quick check for good rectification is a object should same Y height in the rectified images.

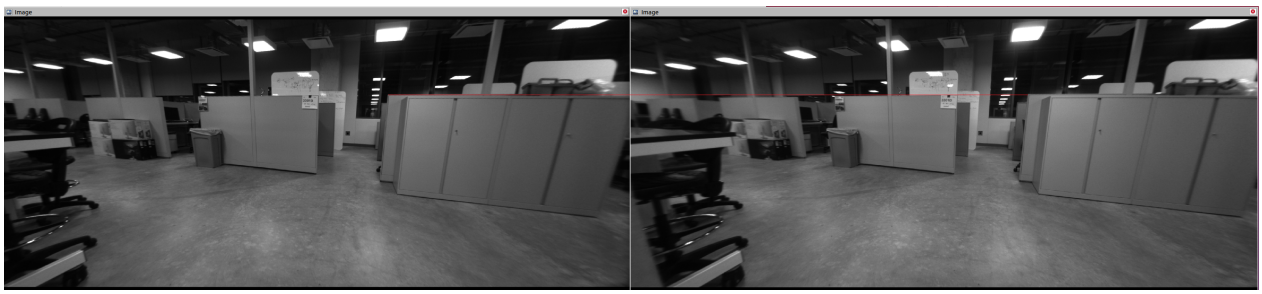

Figure 2: Red line showing same point in both image have same Y height

II Photometric Calibration

We perform photometric calibration according to TUM's paper.[2]

Their camera model considers two main source of distortion to pixel intensity:

1. Lens Vignetting

   When light ray from the same intensity but different orientation hits the lens, they are deflected differently thus suffering different levels of attenuation by the lens. Typically for wide FOV lens, the edges will be darker then the center.

2. Camera response Function

   This describes the property of the photo sensor, that linear increase in light intensity (after the lens) may not give linear increase in received pixel intensity.

Below is citation of its photometric definition:

$$I(x) = G(tV(x)B(x))$$

In which $I(x)$ represents pixel intensity at pixel x, $B(x)$ represents the (true)irradiance, i.e. the true amount of light hitting the camera which will hit pixel x, $V(x)$ is a scalar image in $[0, 1]$ that models the Vignette effect of the lens, t is the exposure time. Together, $tV(x)G(x)$ models the light intensity before hitting the photo sensor. $G : \mathbb{R} \to \mathbb{R}$ defines how a given amount of this light will be converted to pixel intensity.

I Camera Response Function Calibration

Camera response function calibration is done by putting the camera at a static scene to make $B'(x) = V(x)B(x)$ constant, and estimate G by modify the exposure time t and observe the change in $I(x)$.

Specifically, we collect dataset $\{t_i, I_i\}_{i=1}^N$ and optimize $U \in \mathbb{R}^{255}, B \in \mathbb{R}^{W \times H}$ that minimizes[2]

$$E(U, B') = \sum_{i=1}^{N} \sum_{x \in \Omega} \|U(I(x)) - t_i B'\|$$

in which Ω represents all not over-exposed pixels in image i. We can optimize U, B' separately and iterate the optimization process[2]:

$$U^*(k) = \arg\min_{U(k)} E(U, B') = \frac{\sum_{\Omega_k} t_i B'(x)}{|\Omega_k|}$$
$$B'(x)^* = \arg\min_{B'} E(U, B') = \frac{\sum_i t_i U(I_i(x))}{\sum_i t_i^2}$$

In which $\Omega_k = \{i, x | I_i(x) = k\}$, the set of pixels in all images that have intensity k. [2]

Flea3 cameras have a image metadata function that allows one to attach the exposure time used to produce a image with that image. The attached metadata can be read from FlyCapture2 API.

See "CRF_calibration.zip" in the archived files. They are placed under "catkin_ws/src". "flea3" contains my modified package to publish the attached image metadata and provide set shutter, exposure, brightness, gain service calls. "flir_calibration" package contains a node that calls these services and set shutter time from 0.5 ms to 20 ms in 1.05 multiplicative interval.

In the archive "photo_calib", first place the bag file under "static_scene/name/", and run "extract_data.py", "data_preprocess.py", and "camera_response_calibration.py". I have also archived my calibration result for the flea3 cameras in ours_17370485_result folder.

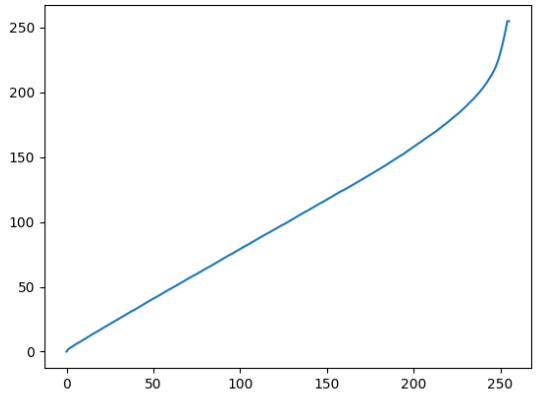

Figure 3: Inverse Camera Response Function of flea3 camera

II Vignette Map Calibration

Vignette calibration is completed by sticking a ARUCO marker on the wall and take pictures of it from different angles. The irradiance of a square area on the wall and the vignette map is jointly estimated. Consult the paper [2] for details.

III Using the TUM Code

For Vignette map calibration, I used TUM's code [2]. This code is old and I recommend to use zovelsanj's version that is OpenCV4 and C++17 compatible.

We need to provide two files:

1. raw images from both cameras in png format Obtain from a bag extraction tool such as "extract_data.py" in "photo_calib.zip".
2. camera.txt file Read the description here. The last parameter in the first line is the ω in FOV distortion. Obtain and convert all coefficients from Kalibr calibration file.

   Here is an example:

```
0.440130112 0.703562493 0.488370814 0.517833025 0.9563762435965946
1280 800
crop
1280 800
```

IV Special Note on Image Collection

This calibration made a assumption that the plane is Lambertian - same amount of light in all directions. Therefore, your surface cannot be reflective, or have certain coating that can form a vague reflected image. Just observe the area you want to use from different angles and see if you can see reflected ceiling lights.

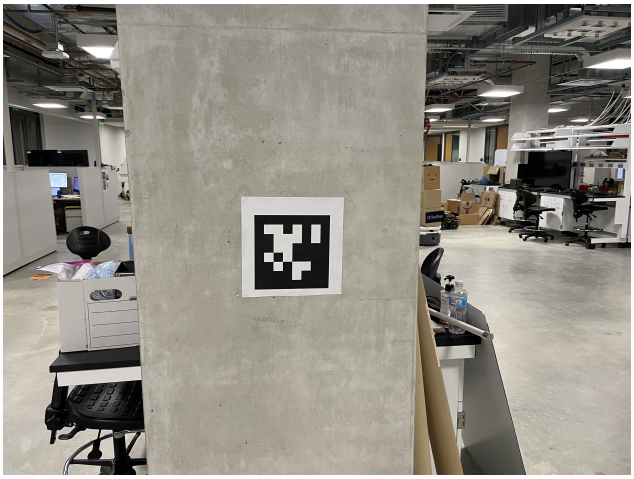

Figure 4: concrete pillars are a good non-reflective surface

It is also important to not change the estimation area's irradiance by your own shadow when collecting images! It will ruin the calibration. I recommend tilting your camera upward, holding the drone lower, therefore your shadow is always below the area of estimation. Also, make sure that all light source behind you cannot cast a shadow on the area of estimation.

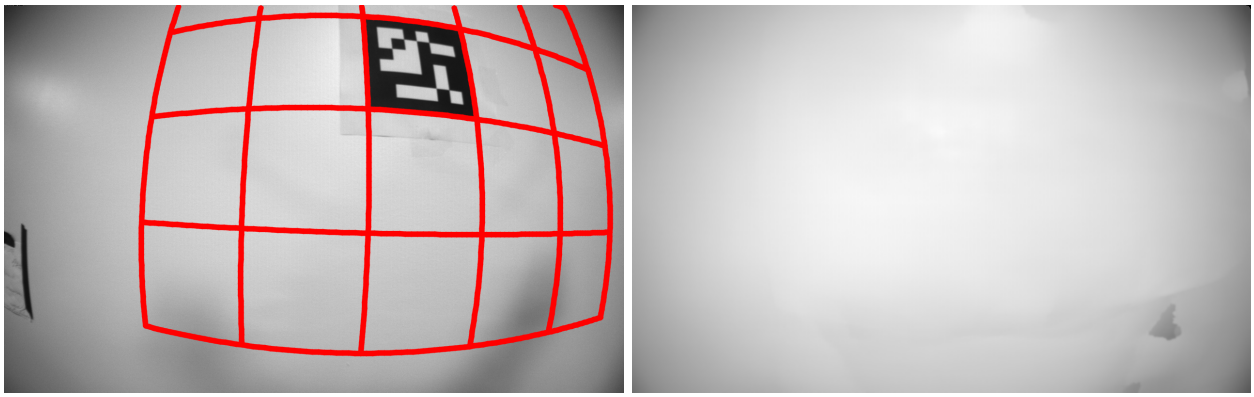

Figure 5: Left: drone's shadow changed the irradiance of area of estimation during image collection. Right: Incorrect vignette map estimated.

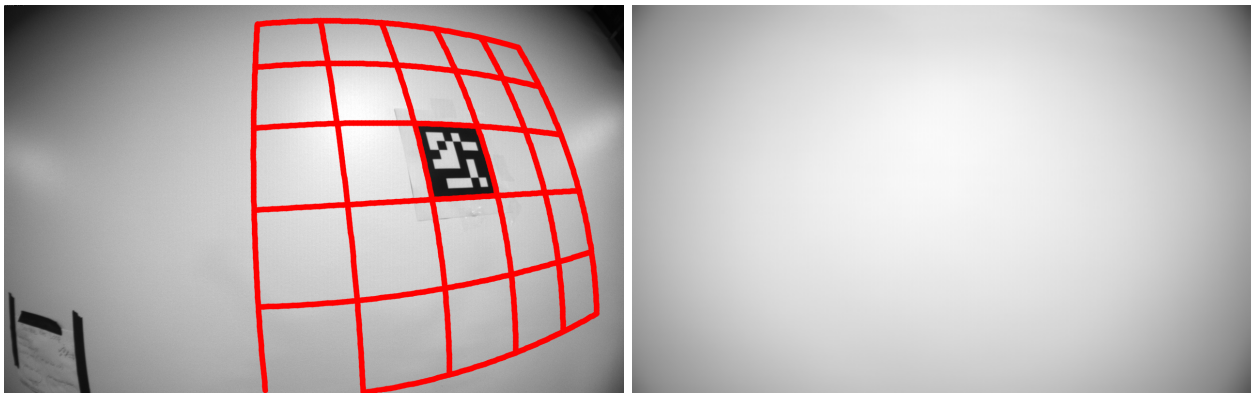

Figure 6: Left:Recommended way of holding your drone lower and point up. Right: Correct vignette map estimated.

V Photometric Correction Implementation

I modified the image_undistort package to include a vignette calibration section. I did not include the CRF calibration because it is close enough to identity and I want to reduce computation cost.

see "src/image_undistort" in "dsol_ws_20230126.zip" in the archive for implementation details.

III System Integration

I Overall Map

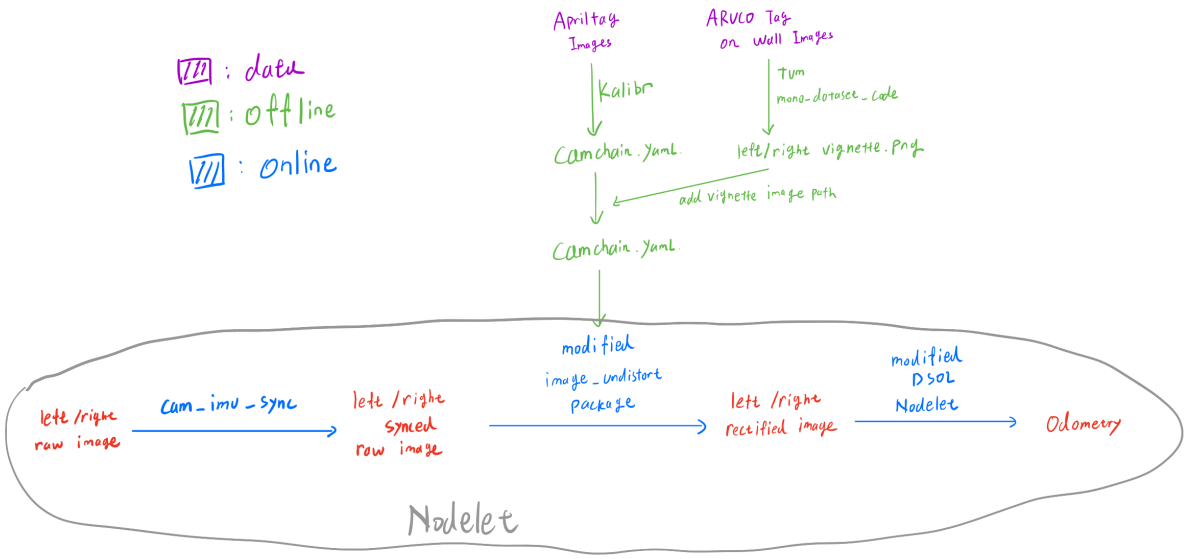

Figure 7: Workflow of entire system

I modified the image_undistort to perform both the geometric and photometric rectification of the images, to reduce topic messaging between ROS system.

II ROS Components Related to Sending Images (TLDR: Use Nodelets if possible!)

Images are big chunks of data. If we do not handle its transport correctly, we will waste computation resource and cause delays in the system.

There are several components / packages in ROS that is related to transporting images. They are:

1. Default Publisher / Subscriber of sensor_msgs::Images

   We transport images through the raw ROS system. Our image in type sensor_msgs::Images is serialized into a byte stream, sent over the ROS network, and deserialized back into type sensor_msgs::Images. This serialization / deserialization is a waste of computation resource.

2. ImageTransport

   A package that converts image publishing into interfaces. In addition to sending them in raw, you can send images in compressed form.

3. Nodelet

   A package that allows you to combine several node **writen in C++** into one node. Message sent internally are not copied, instead, is passed by pointers. This saves us from serializing / deserializing images.

4. cv_bridge

   A package that converts sensor_msgs::Images and cv::Mat used by OpenCV.

To test their effects on image delays, I wrote a testing package that includes a publisher and a subscriber. The publisher will send a image at a given rate, timestamp the image just before publishing and ros::spinOnce. The subscriber will record the receiving time stamp, compare with the image's time stamp to compute the delay between publishing and receiving.

I archived the package in "stereo_undistort.zip".

Here are the results:

Publish Rate	ROS Default	image_transport/raw	image_transport/compressed	nodelet
25	15.8 ± 1.33	23.7 ± 4.82	35.64 ± 5.76	$\mathbf{11.09 \pm 1.16}$
50	12.1 ± 1.52	11.7 ± 1.02	35.7 ± 7.4	$\mathbf{9.5 \pm 0.87}$
100	4.79 ± 0.16	8.73 ± 0.13	$119.7 \pm 2.09^{*}*$	$\mathbf{3.3 \pm 0.23}$
200	5.24 ± 1.19	5.18 ± 1.07	N/A	$\mathbf{3.4 \pm 0.41}$
400	$\mathbf{11.88 \pm 3.50}$	$9.91 \pm 2.45^{*}$	N/A	$14.76 \pm 0.09^{*}$

Table 1: Image transport delay / standard deviation in ms computed by 100 samples. * = occasional image lost($< 1\%$), ** = serious image lost

Publish Rate	ROS Default	image_transport/raw	image_transport/compressed	nodelet
25	330	330	7	330
50	661	664	14	663
100	1328	1329	28	1328
200	2558	2459	N/A	2670
400	2197	2175	N/A	2394

Table 2: Topic Bandwidth (MB/s)

My test image size is 2420×1816. All delays does not include transition from cv::Mat to sensor_msgs::Images, this is just for topic sending. As we can see, nodelet is the best option for the least delay.

This observation is also confirmed at real-drone system integration. By porting everything into nodelets thus preventing serialization of images, I am able to run DSOL much faster and can achieve better localization accuracy.

IV Odometry Evaluation

Finally we are ready to evaluate the odometry accuracy. I used the Vicon motion capture system as a ground truth of odometry.

I Body Frame Misalignment issue

I encountered the problem of body frame misalignment from Vicon and DSOL. When running Odometry, the SLAM algorithm will define its transpose as Odometry body frame to Odometry global frame. When using Vicon, it will define its own body and global frames. **The two body frames are not equal!**

I surveyed materials on this topic. [4] used hand-eye calibration to localize the reflective balls used by vicon with onboard cameras therefore establishing the relation between two frames, which we do not have. I cannot find other sources on this issue.

I implemented the following method:

II Frame of Reference

$\{V\}$	Vicon Global Frame
$\{V_B\}$	Vicon Body Frame
$\{O\}$	Odometry Global Frame
$\{O_B\}$	Odometry Body Frame

In which $\{V\}$ and $\{O\}$ are fixed by ground, thus a static transform exists between them. Let $T^{\{V\}}_{\{O\}}$ denote the static transform from $\{O\}$ to $\{V\}$.

$\{V_B\}$ and $\{O_B\}$ are fixed by the drone, thus a static transform exists between them. Let $T^{\{V_B\}}_{\{O_B\}}$ denote the static transform from $\{O_B\}$ to $\{V_B\}$.

III Trajectory Data

Vicon:	Pose array from $\{V_B\}$ to $\{V\}$	Symbol: $T^{\{V\}}_{\{V_B\}}(n)$
Odometry:	Pose array from $\{O_B\}$ to $\{O\}$	Symbol: $T^{\{O\}}_{\{O_B\}}(n)$

IV Alignment

To compare the two trajectories, we should apply static transform to bring Odometry trajectory (representing $\{O_B\}$ to $\{O\}$) to represent the same transformation as the Vicon trajectory, i.e.

$$\hat{T}^{\{V\}}_{\{V_B\}}(n) = T^{\{V\}}_{\{O\}} \cdot T^{\{O\}}_{\{O_B\}}(n) \cdot (T^{\{V_B\}}_{\{O_B\}})^{-1}$$

With such transform we can form error metric on position and orientation. My issue now is how to obtain the static transformations $T^{\{V\}}_{\{O\}}$ and $T^{\{V_B\}}_{\{O_B\}}$.

Here is my method:

1. $T^{\{V\}}_{\{O\}}$:

   Solve transformation between static frames using Umeyama's algorithm which optimizes $\hat{T}^{\{V\}}_{\{O\}}$ such that

   $$\hat{T}^{\{V\}}_{\{O\}} = \arg\min_{T} \|p(T^{\{V\}}_{\{V_B\}}(n)) - p(T \cdot T^{\{O\}}_{\{O_B\}}(n) \cdot (T^{\{V_B\}}_{\{O_B\}})^{-1})\|$$

   In which I provide the path $T^{\{V\}}_{\{V_B\}}(n)$ and $T^{\{O\}}_{\{O_B\}}(n) \cdot (T^{\{V_B\}}_{\{O_B\}})^{-1}$ to the algorithm as reference and estimation(to be aligned).

   in which $p(\cdot) : \mathrm{SE3} \to \mathbb{R}^3$ extracts the translation part of the SE(3) transform.

2. $T^{\{V_B\}}_{\{O_B\}}$

$$\hat{T}^{\{V_B\}}_{\{O_B\}} = \text{average}\left[\left(T^{\{V\}}_{\{V_B\}}(n)\right)^{-1} \cdot T^{\{V\}}_{\{O\}} \cdot T^{\{O\}}_{\{O_B\}}(n)\right]$$

Since they depend on eachother, I iterated them until convergence. Finally, I converted the trajectory with

$$\hat{T}^{\{V\}}_{\{V_B\}}(n) = \hat{T}^{\{V\}}_{\{O\}} \cdot T^{\{O\}}_{\{O_B\}}(n) \cdot (\hat{T}^{\{V_B\}}_{\{O_B\}})^{-1}$$

V Error Metrics

We use EVO to calculate APE and RPE of position and rotation. They are defined as[3]:

$$\begin{aligned}
APE_{pos}(n) &= \|\hat{p}(n) - p(n)\|_2 \\
APE_{rot}(n) &= \log\left([R(n)]^T \hat{R}(n)\right) \\
RPE_{pos}(n,m) &= \|[\hat{p}(m) - \hat{p}(n)] - [p(m) - p(n)]\|_2 \\
RPE_{rot}(n,m) &= \log\left([R(n)^T R(m)][\hat{R}(n)^T \hat{R}(m)]\right)
\end{aligned}$$

and I compute their RMS for comparison.

VI Result Analysis

I used EVO to compute error metrics of the trajectories. My data processing notebook is archived at "bag_data_processing.ipynb"

VI.1 Summary

Name	Position APE (m)	Rotation APE (deg)	Position RPE (m/m)	Rotation RPE (deg/deg)
dsol calib 25 Hz	**0.378**	**3.793**	0.076	0.277
dsol raw 25 Hz	0.532	3.911	**0.069**	**0.232**
dsol calib 60 Hz	0.388	7.659	0.128	0.461
dsol raw 60 Hz	0.712	7.649	0.151	0.480

Table 3: Statistics for test on 20230123 night. Obtained by hand-held the drone walking the similar path.

Name	Position APE (m)	Rotation APE (deg)	Position RPE (m/m)	Rotation RPE (deg/deg)
dsol calib 25 Hz	0.378	3.124	0.057	0.448
dsol raw 25 Hz	0.316	3.166	0.221	1.162
dsol calib 60 Hz	0.328	2.676	**0.049**	**0.305**
dsol raw 60 Hz	0.295	2.813	**0.049**	0.369
msckf 25 hz	0.388	3.014	0.171	0.372
msckf 60 hz	**0.181**	**2.163**	0.80	0.375

Table 4: Statistics for test on 20230126 noon. Obtained by hand-held the drone walking the similar path.

In all tests I walked in a similar path and time, therefore the distribution of the path can be considered identical and the results are comparable.

Contrasting DSOl to MSCKF performance we can see MSCKF is better in absolute accuracy, while DSOL is better in relative accuracy. **Therefore, for our purpose of training in the wild using short sequence of poses, I recommend DSOL**.

Contrasting 25Hz and 60Hz image rate, we see 60 hz version almost always better than 25 hz in position and rotation accuracy. Especially, we see great performance improvement for MSCKF in 60Hz. Later results also shown no localization failures for the 60hz case as for the 25 hz case. Therefore I suggest using a higher framerate when possible.

The 20230123 test is obtained near subset, which shows larger error compared to the test on 20230126. This requires us to maintain proper exposure for DSOL to function properly.

Photometric calibration lead to better RPE but worse APE statistics. For our purpose of training we should use photometric calibration.

VI.2 Alignment Consistency

I analyzed the consistency between transformations estimated by my alignment algorithm:

Name	Translation (m)	Rotation (quat)
DSOL $T_{\{O\}}^{\{V\}}$	$[0.355, 0.601, 0.37] \pm [0.051, 0.082, 0.033]$	$[0.424, -0.524, 0.584, -0.453] \pm [0.012, 0.02, 0.016]$
DSOL $T_{\{O_B\}}^{\{V_B\}}$	$[0.149, 0.112, 0.004] \pm [0.057, 0.019, 0.017]$	$[0.44, -0.545, 0.565, -0.436] \pm [0.006, 0.01, 0.009]$
MSCKF $T_{\{O\}}^{\{V\}}$	$[0.254, 0.249, 0.374] \pm [0.007, 0.182, 0.034]$	$[0.999, -0.007, 0.021, -0.048] \pm [0.006, 0.015, 0.006]$
MSCKF $T_{\{O_B\}}^{\{V_B\}}$	$[0.091, -0.053, -0.01] \pm [0.014, 0.04, 0.013]$	$[-0.731, 0.014, 0.682, 0.014] \pm [0.009, 0.006, 0.007]$

Table 5: Statistics for test on 20230126 noon. Estimated static transform with their standard deviation

This shows both alignment are consistent between runs.

VI.3 Position

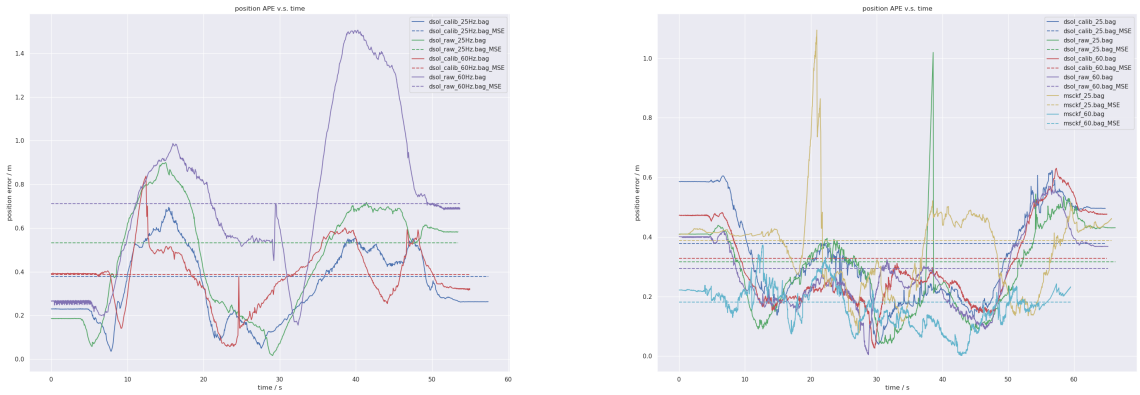

Figure 8: Position APE Left: test on 20230123 night Right: test on 20230126 noon

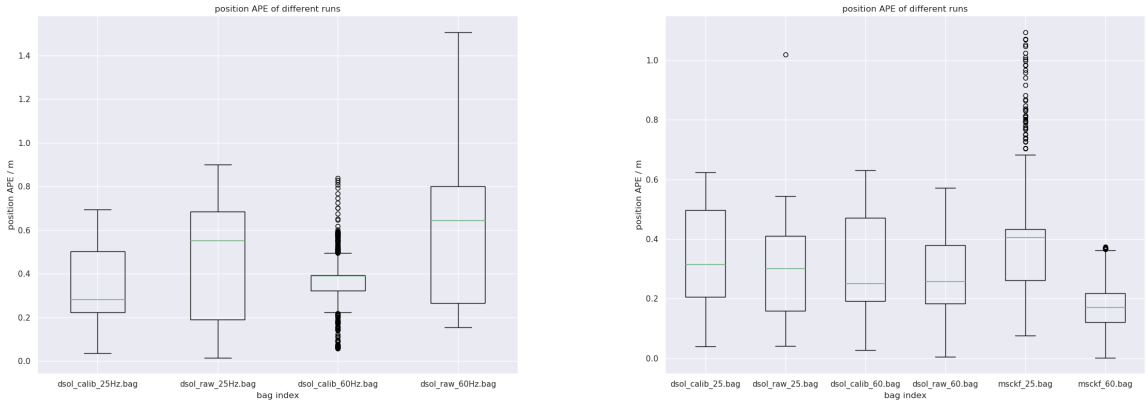

Figure 9: Position APE Left: test on 20230123 night Right: test on 20230126 noon

APE

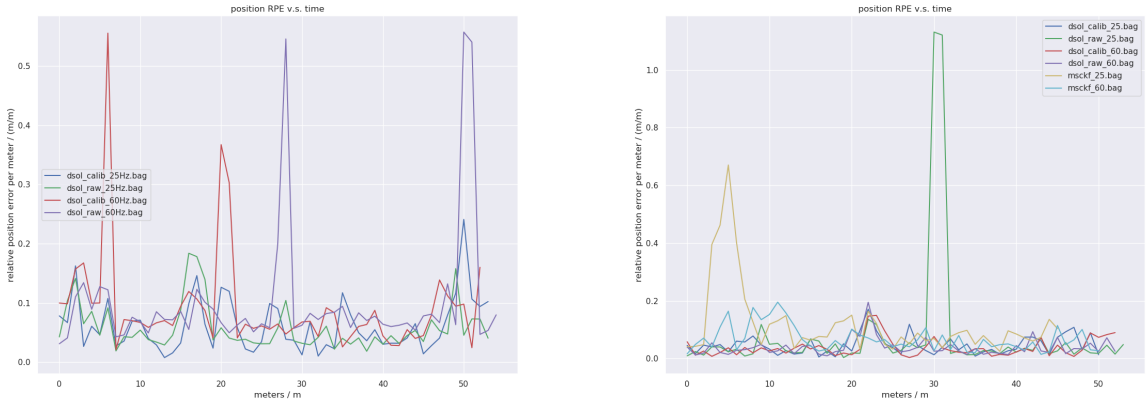

Figure 10: Position RPE Left: test on 20230123 night Right: test on 20230126 noon

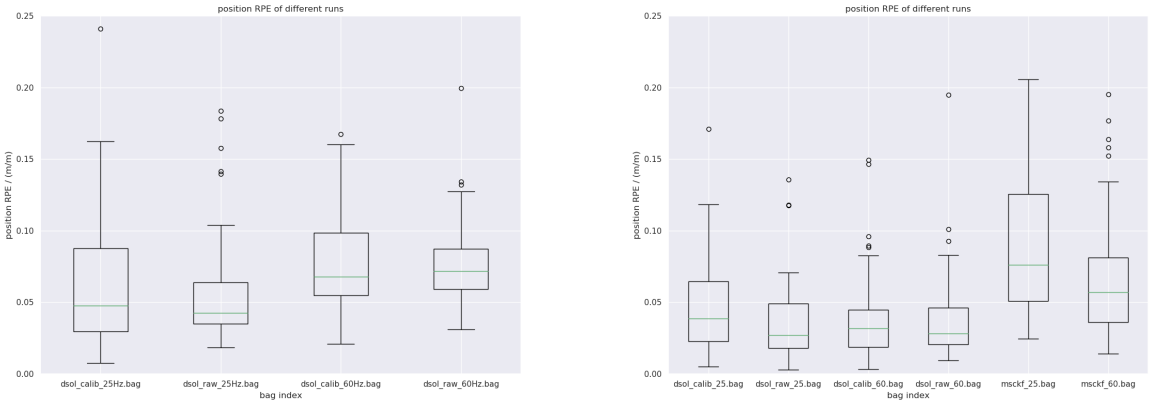

Figure 11: Position RPE Left: test on 20230123 night Right: test on 20230126 noon

RPE

VI.4 Orientation

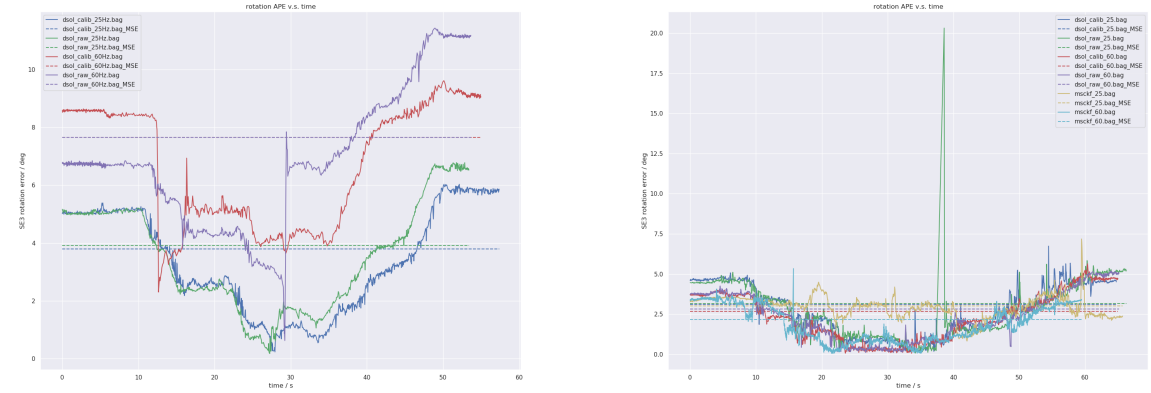

Figure 12: Rotation APE Left: test on 20230123 night Right: test on 20230126 noon

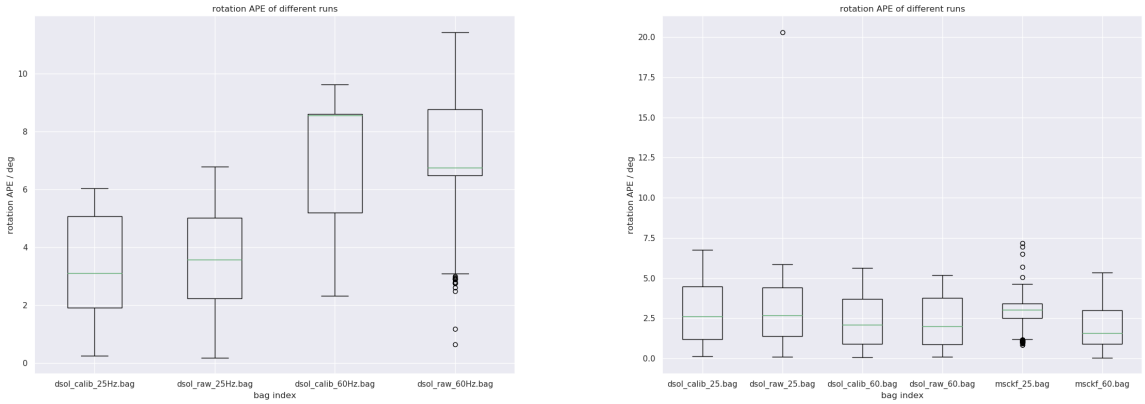

Figure 13: Rotation APE Left: test on 20230123 night Right: test on 20230126 noon

APE

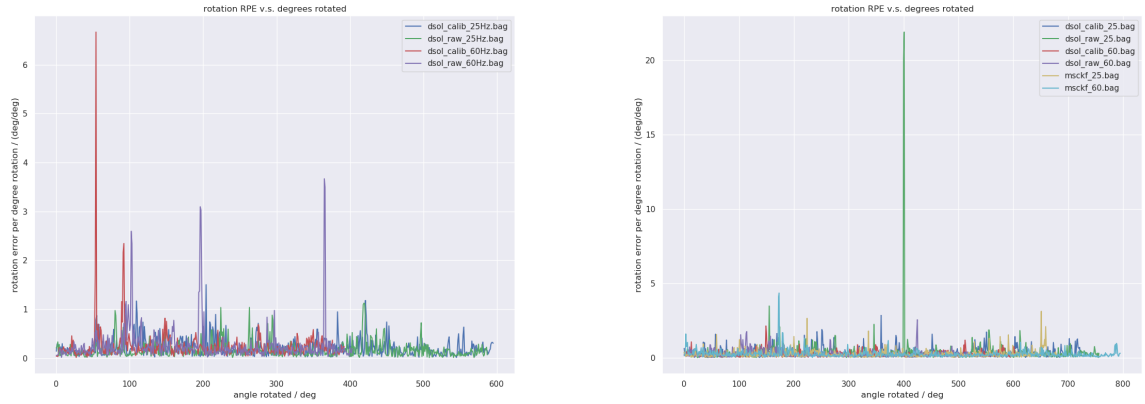

Figure 14: Rotation RPE Left: test on 20230123 night Right: test on 20230126 noon

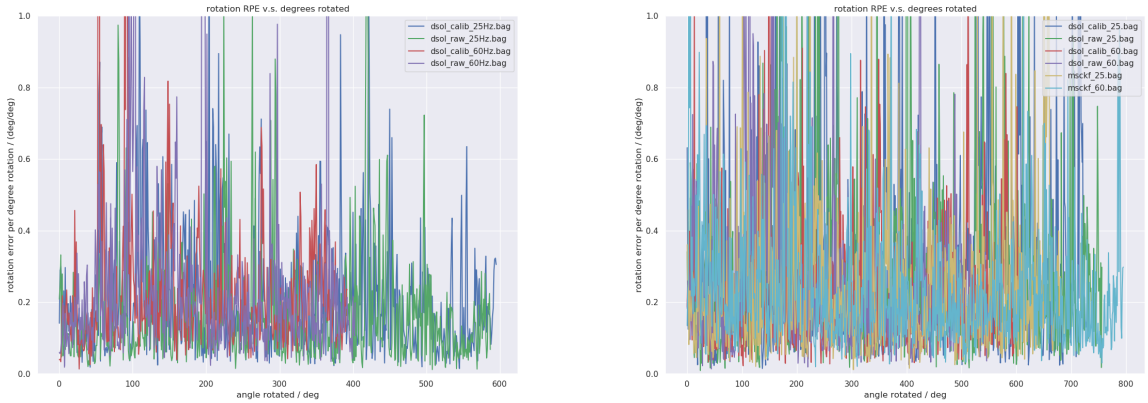

Figure 15: Zoomed in Rotation RPE Left: test on 20230123 night Right: test on 20230126 noon

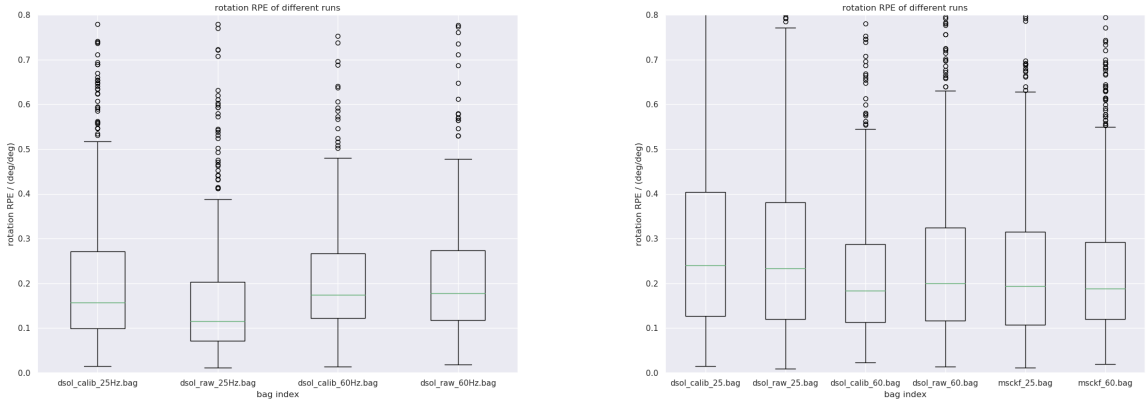

Figure 16: Rotation APE Left: test on 20230123 night Right: test on 20230126 noon

RPE

References

[1] *Department of Informatics*. URL: https://rpg.ifi.uzh.ch/teaching.html.

[2] J. Engel, V. Usenko, and D. Cremers. "A Photometrically Calibrated Benchmark For Monocular Visual Odometry". In: *arXiv:1607.02555*. July 2016.

[3] MichaelGrupp. *Evo/metrics.py_api_documentation.ipynb at master — Michaelgrupp/Evo*. Jan. 2022. URL: https://github.com/MichaelGrupp/evo/blob/master/notebooks/metrics.py_API_Documentation.ipynb.

[4] Arnaud Tanguy et al. "Closed-loop MPC with Dense Visual SLAM - Stability through Reactive Stepping". In: *2019 International Conference on Robotics and Automation (ICRA)*. 2019, pp. 1397–1403. DOI: 10.1109/ICRA.2019.8794006.