Chair of Computer Vision Group
TUM School of Computation, Information and Technology
Technical University of Munich

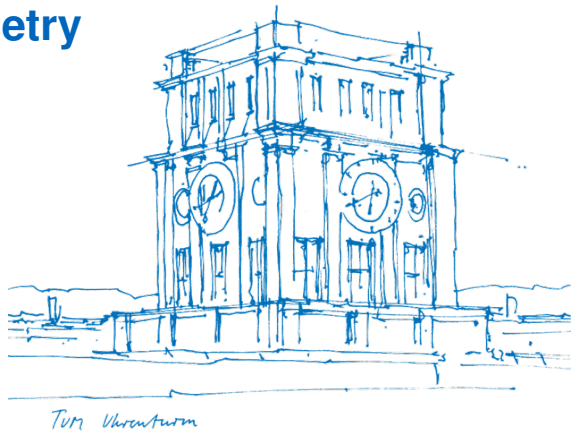
Codec-Aware Visual Odometry

Understanding and Exploiting Video Compression for Visual Odometry

Christoph Otten genannt Hermes

Chair of Computer Vision Group
TUM School of Computation, Information and Technology
Technical University of Munich

Oktober 30th, 2025



Outline

Leveraging Video Codecs for Tracking

- Online:
  example over the air transmitting for fire fighters
- Offline:
  understanding the vast amount of web videos (downstream learning)

Contributions

- **Findings**
  - comparison of video encoders for egocentric visual tracking
  - show speed improvements thanks to reduced data size
  - show a novel way of leveraging motion vector for frontend initialization that improves tracking accuracy
- **Implementations**
  - Script for batch compressing datasets
  - adding video support to Basalt with Monado.
  - updating mv-extractor to retrieve frames from H.264 encoded videos

Outline

Frame Types

- I-Frames
  → Intra encode frames, also known as keyframes
- P-Frames
  → Predictive frames made from earlier pictures, so that less coding data is required
- B-Frames
  → bidirectionally predicted pictures

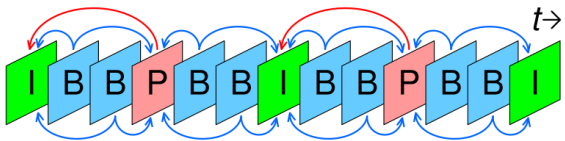


Figure 1 inter frame group of pictures

Block Structure

- Block-based coding is used to exploit spatial and temporal redundancy.
- Both **VP9** and **H.265 (HEVC)** use dynamic block sizes while **H.264** uses static sizes.
- Large blocks are efficient for static or smooth regions.
- Small blocks capture fine details and fast motion.

VP9 Block Structure

- Frames are divided into **64×64 pixel superblocks**.
- Each superblock can be recursively split into smaller blocks:
  - 64×64 → 32×32 → 16×16 → 8×8.
  - 8×8 blocks may further divide into 8×4, 4×8, or 4×4 **subblocks**.
- Each block has its own header (*mode info*); subblocks share one header.
- Decoding proceeds in **raster order**.

H.265 (HEVC) Block Structure

- Uses **Coding Tree Units (CTUs)** of size 16×16, 32×32, or 64×64 pixels.
- CTUs are divided via a **quad-tree** into smaller **Coding Units (CUs)**.
- Each CU contains one or more **Prediction Units (PUs)**:
  - Intra-coded CUs: only square partitions.
  - Inter-coded CUs: square and non-square partitions (minimum 4 pixels per side).
- Up to eight partition types for inter-coded blocks for better motion adaptation.

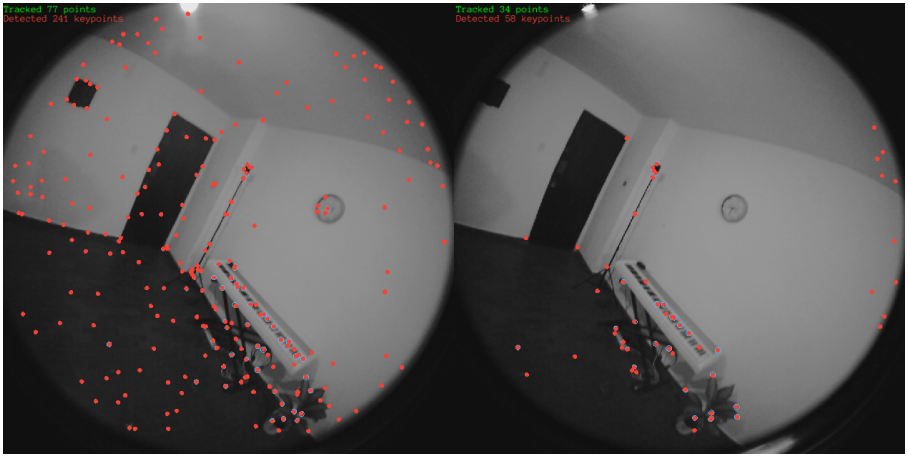
Motion Vectors (MVs)

- Used in **inter-frame prediction** (temporal compression).
- Represent offsets to similar regions in previous frames.
- Only the **motion vector** and a small **residual** are stored.

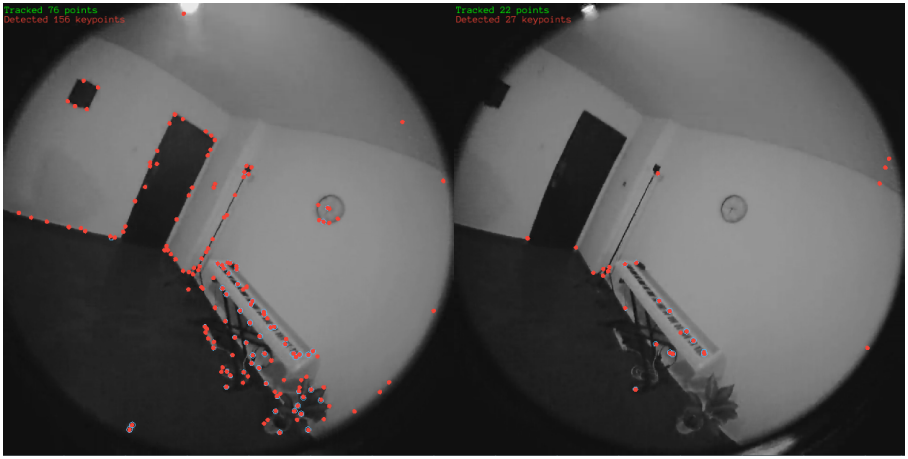
How Motion Vectors Work

- Encoder searches reference frame for the best-matching block.
- The displacement between current and reference block = **motion vector**.
- Example: static regions $\rightarrow$ zero motion vector.
- Decoder reconstructs the block by:
  1. Copying the reference block at the given offset.
  2. Adding the decoded residual.

Artefacts

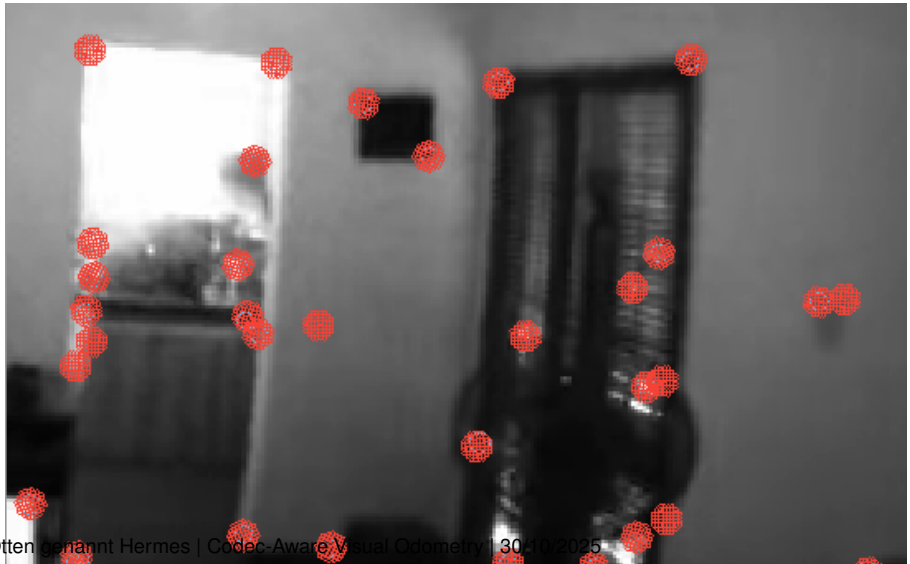

Tracked 77 points
Detected 241 keypoints
Tracked 34 points
Detected 58 keypoints

Artefacts


Tracked 76 points
Detected 156 keypoints
Tracked 22 points
Detected 27 keypoints

Artefacts

Artefacts



Dataset Selection

- Final pool: **13 sequences** across **6 environments**.
- Includes all **easy** sequences from **EuRoC** and **MSD**.
- Plus the **top-3** sequences (raw PNGs, default optical flow) by tracking accuracy.
- *Exclusion:* **Room1** (strong outlier; diverged under compression).
- Rationale: showcase improvements where default OF already tracks well; if both runs diverge, ATE/RTE differences are uninformative.

Design Goals for Encoding Settings

- **Real-world applicability** while preserving **comparability** across runs.
- Primary control knob: **average bitrate (ABR)**.
- Let the encoder **adapt per frame**: low motion $\rightarrow$ use less bitrate; complex motion/detail $\rightarrow$ allocate more.
- Choose bitrate from desired **file size** and **sequence duration**.

Two-Pass Encoding

- Use **two-pass** to hit target bitrate accurately.
- **Pass 1:** statistics collection (no audio, discard output).
- **Pass 2:** constrained optimization given Pass 1 stats.
- Applied consistently across **H.264** and **VP9**.

Codecs & Motion-Vector Controls

- Codecs: **H.264 (libx264)** and **VP9 (libvpx-vp9)**.
- H.264 configured with **MV/partition** options (per mov-slam guidance):
  - Partitions: `p8x8,p4x4,i8x8`
  - GOP/keyframe: `keyint`=1000
  - Motion estimation: `me`=`umh`, `merange`=64, `subme`=6
  - `bframes`=0, `ref`=1 for comparability
- VP9 run at matched ABR levels under two-pass.

Example: H.264 (Two-Pass, 100 kbit/s)

```
# Pass 1: stats only
ffmpeg -y -framerate 20 -pattern_type glob \
  -i "MH_01_easy/mav0/cam0/data/*.png" \
  -c:v libx264 -b:v 100k \
  -x264opts partitions=p8x8,p4x4,i8x8:keyint=1000:me=umh:merange=64:subme=6:
  -pass 1 -an -f null /dev/null

# Pass 2: actual encode
ffmpeg -y -framerate 20 -pattern_type glob \
  -i "MH_01_easy/mav0/cam0/data/*.png" \
  -c:v libx264 -b:v 100k \
  -x264opts partitions=p8x8,p4x4,i8x8:keyint=1000:me=umh:merange=64:subme=6:
  -pass 2 -an -f mp4 "MH_01_easy/mav0/cam0/data.mp4"
```

Example: VP9 (Two-Pass, 100 kbit/s)

```
# Pass 1: stats only
ffmpeg -y -framerate 20 -pattern_type glob \
  -i "MH_01_easy/mav0/cam0/data/*.png" \
  -c:v libvpx-vp9 -b:v 100k -pass 1 -an \
  -f webm /dev/null

# Pass 2: actual encode
ffmpeg -y -framerate 20 -pattern_type glob \
  -i "MH_01_easy/mav0/cam0/data/*.png" \
  -c:v libvpx-vp9 -b:v 100k -pass 2 -an \
  "MH_01_easy/mav0/cam0/data.webm"
```

What to Expect

- Low bitrate (100k): strong compression; good for static scenes, limited detail.
- Mid bitrate (500k–1M): balanced quality vs size; robust across typical motion.
- High bitrate (5M): best preservation of fine texture and fast motion.

Outline

1 Motivation

2 Video Compression

3 Results
- Size Reduction
- Compression Effects on Tracking Accuracy
- Compression effects on Tracking Speed
- Accuracy Improvements with Motion Vector Prior in Frontend

File size per sequence

Size of single camera in MB

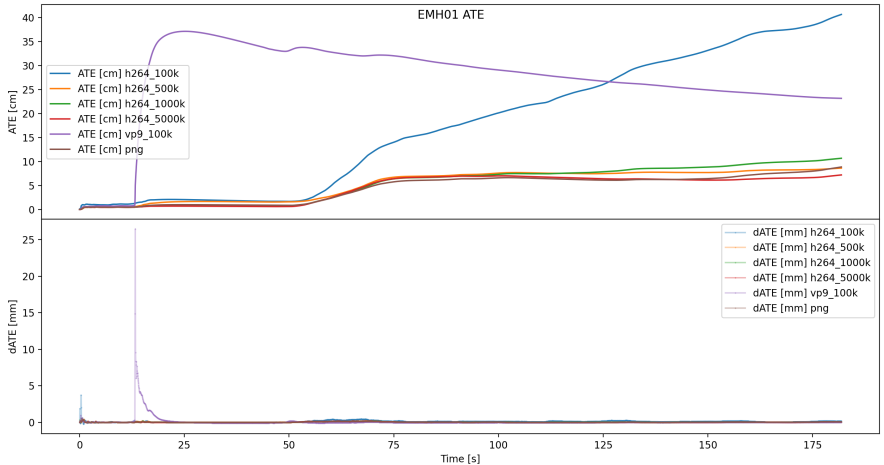
sequence	size as PNGs	size as 100k	size as 500k	size as 1000k	size as 5000k
EMH01	1333.00	2.22	11.01	21.99	109.75
EMH02	1100.87	1.84	9.07	18.14	90.64
EV101	1054.54	1.76	8.70	17.38	86.91
EV201	825.67	1.38	6.81	13.62	68.10
MGO05	730.86	1.65	8.14	16.30	81.43
MGO07	292.70	0.66	3.19	6.36	31.66
MIO05	1789.11	2.56	7.38	14.68	73.02
MIO07	1084.19	1.55	4.61	9.07	45.22
MOO05	532.80	1.22	6.00	11.99	59.84
MOO07	220.87	0.51	2.48	4.93	24.51
TR2	873.83	1.74	8.61	17.20	85.88
TR4	673.74	1.35	6.66	13.31	66.43
TR6	794.50	1.59	7.86	15.69	78.54

Compression Effects on vSLAM Tracking Accuracy

- **Motivation:** Video compression enables vSLAM on resource-limited hardware and allows use of large-scale, already-compressed online video data.
- **Goal:** Evaluate how different **encoders (H.264, VP9)** and **bitrates** affect tracking accuracy (ATE/RTE) using Basalt's original optical flow.
- **Findings:**
  - High bitrates (**5000k**) yield errors close to or below the original PNG version.
  - VP9 achieves comparable or better accuracy at lower bitrates than H.264.
  - Using compressed video reduced ATE in 8–9 sequences and RTE in 7.
- **Trends:**
  - Lower bitrates $\Rightarrow$ increased errors and occasional divergence (e.g., MIO sequences).
  - Very low bitrate (**100k**) often fails or produces unusable results.
  - **500k bitrate** offers best trade-off between file size and accuracy.
- **Conclusion:** Compression—especially with VP9—can greatly reduce storage needs while maintaining usable tracking accuracy, making compressed video viable for vSLAM.

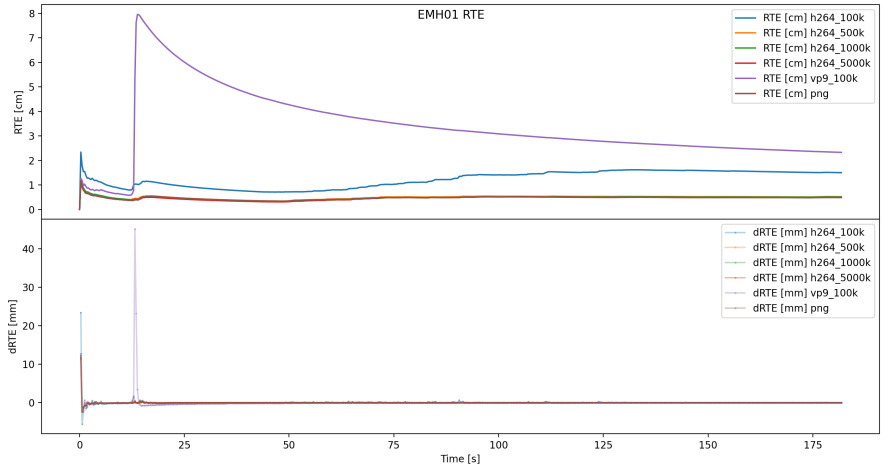
Bitrate Comparisson

EMH01 ATE



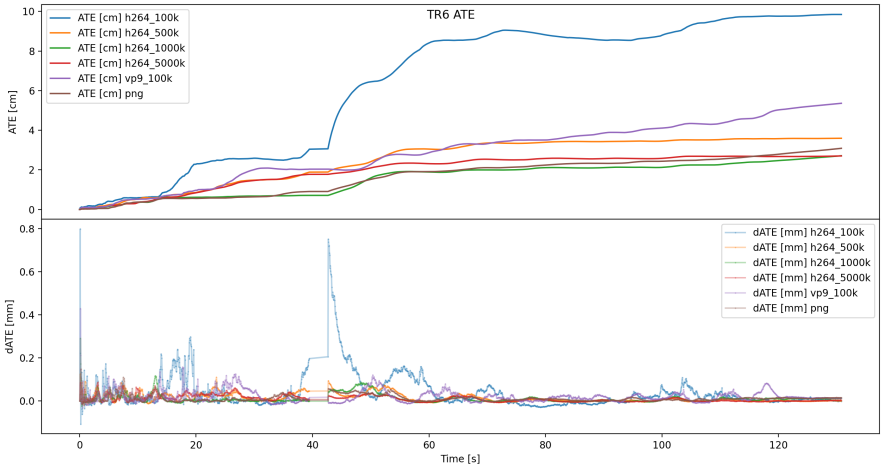
Bitrate Comparisson

EMH01 RTE

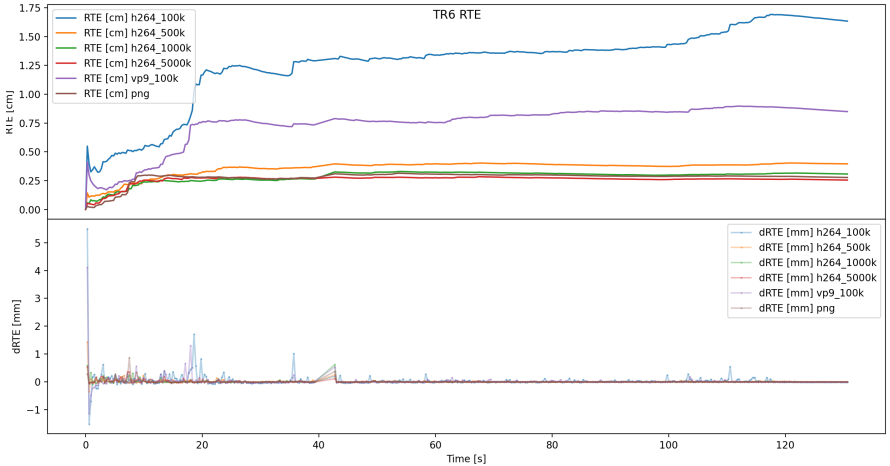


Bitrate Comparisson

TR6 ATE



Bitrate Comparisson
TR6 RTE



Impact of Video Compression on Tracking Speed

- **Objective:** Compare full end-to-end **Basalt runtime** when using raw PNGs vs. compressed video input.
- **Setup:**
  - Input: H.264-encoded videos at **500k bitrate** (recommended accuracy setting).
  - Both runs use **optical flow only** (no motion vectors).
  - Tests performed on Linux Mint 22.1, i7-12700KF, 31.2 GiB RAM, SATA SSD.
- **Findings:**
  - **Average runtime reduction: 38.3%** compared to raw PNGs.
  - **Main cause:** smaller file size $\rightarrow$ faster loading and memory access.
  - Motion vectors have minor influence on speed in this comparison.
- **Interpretation:** Using compressed video improves runtime efficiency without harming accuracy, making it highly suitable for **real-time or resource-limited vSLAM**.

Leveraging Motion Vectors for Improved Tracking

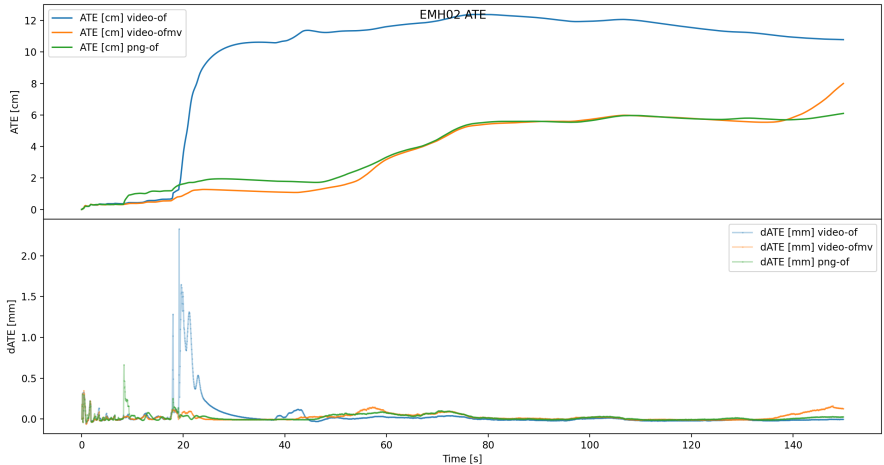
- **Concept:** Use the **motion vectors (MVs)** generated during video compression to support the **feature tracking stage** in Basalt's frontend.
- MVs complement traditional optical flow by providing **encoder-derived motion information**.
- **Evaluation:** Comparison of **optical flow only (OF)** vs. **motion-vector–assisted optical flow (MVOF)**.
- **Results:**
  - MVOF improves performance in **10 of 13 datasets**.
  - **Average ATE reduction:** $\approx$ **60%**
  - **Average RTE reduction:** $\approx$ **28%**
- **Interpretation:** Leveraging MVs helps recover lost features and **stabilizes tracking**, especially under strong compression where optical flow struggles.
- Example:

OF vs MVOF - Example

MVOF prevents large trajectory spikes (e.g., at 20s), preserving consistent tracking even with compressed input.

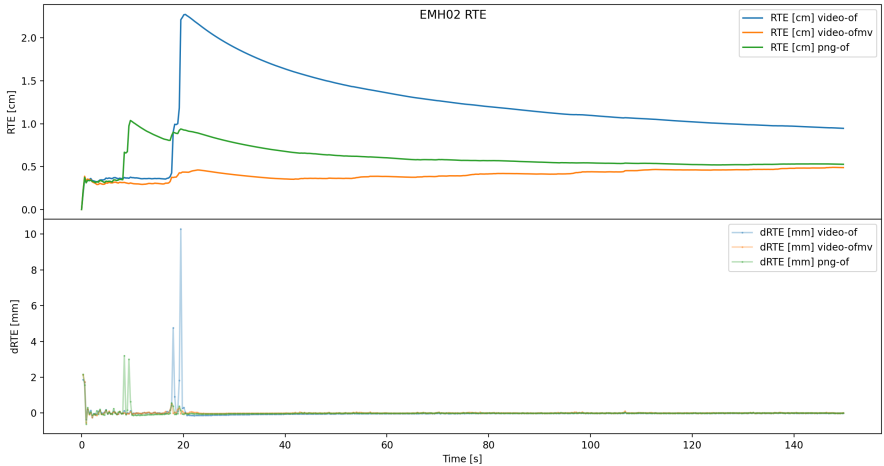
OF vs MVOF - Example

EMH02 ate

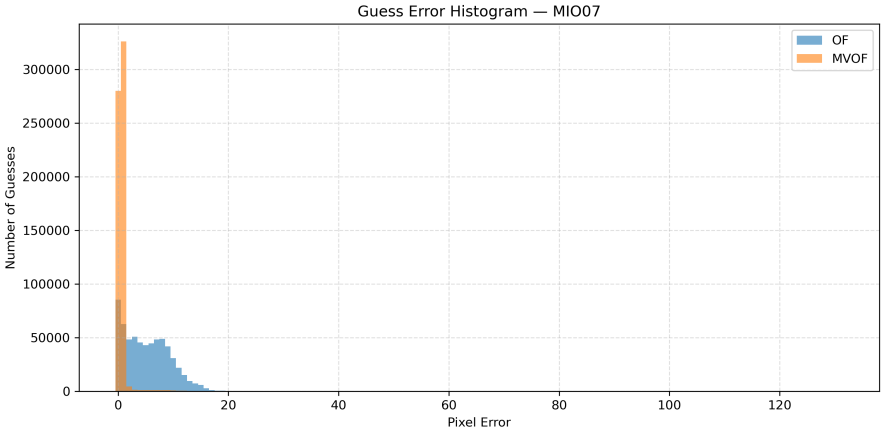


OF vs MVOF - Example

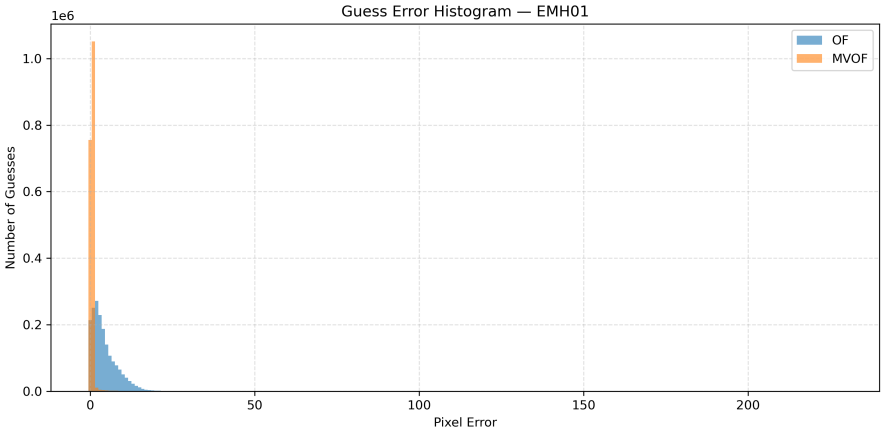
EMH02 rte



Guess Error Histogram
MIO07



Guess Error Histogram
EMH01



Guess Error Histogram
MSD

sequence	fallback	total_count	mean_error	median_error	max_error_bin
MGO05	of	1388040	5.413876	4	82
MGO05	mvof	1417892	0.832551	1	223
MGO07	of	449830	8.067670	7	64
MGO07	mvof	474180	1.034786	1	262
MIO05	of	1174348	4.964357	4	76
MIO05	mvof	1186337	0.740718	1	242
MIO07	of	618506	5.286853	5	70
MIO07	mvof	623569	0.706534	1	131
MOO05	of	671907	5.089441	4	46
MOO05	mvof	680761	0.795388	1	152
MOO07	of	225769	6.759117	6	38
MOO07	mvof	230635	0.934307	1	163

Guess Error Histogram

EuRoC & Tumvi

sequence	fallback	total_count	mean_error	median_error	max_error_bin
EMH01	of	1836115	4.353661	3	85
EMH01	mvof	1840896	0.692880	1	228
EMH02	of	1507879	4.214298	3	110
EMH02	mvof	1516745	0.711271	1	221
EV101	of	1198291	5.715888	5	51
EV101	mvof	1212295	0.746001	1	211
EV201	of	770220	5.545621	4	50
EV201	mvof	775477	0.743825	1	244
TR2	of	420463	9.078968	8	48
TR2	mvof	439613	1.394690	1	269
TR4	of	379325	9.353578	9	69
TR4	mvof	397962	1.414758	1	260
TR6	of	583197	6.359688	6	48
TR6	mvof	586464	0.987493	1	249

Figures

- Fig 1: `https://en.wikipedia.org/wiki/Inter_frame`
- All other figures were created by myself using either custom python scripts or XRTMET.