Codec-Aware Visual Odometry

Understanding and Exploiting Video Compression for Visual Odometry

Scientific work to obtain the degree
B.Sc.
at the School of Computation, Information and Technology at the Technical University of Munich.

Supervisor: Prof. Dr. Daniel Cremers
Chair of Computer Vision & Artificial Intelligence

Advisor: Mateo de Mayo
Chair of Computer Vision

Author: Christoph Otten genannt Hermes
Nymphenburger Str. 150
80634 Munich
+49 89 30702062

Submission Date: Munich, the 15.10.2025

Appendix I

Declaration

I hereby certify that I have written the thesis I have submitted independently and have not used any sources or resources other than those indicated.

Munich, 15.10.2025 Christoph Otten genannt Hermes

Contents

1. Introduction

1.1. Context of Visual Odometry and SLAM

Simultaneous Localization and Mapping (SLAM) allows autonomous agents to estimate their motion and build a map of their surroundings using onboard sensors. When performed from the perspective of the moving platform itself—such as a camera mounted on a headset, drone, or robot—it is referred to as egocentric visual SLAM. In contrast to external tracking systems, egocentric SLAM operates in an inside-out manner, relying solely on the agent's own sensors to perceive and understand the environment in real time.
This approach is central to many modern technologies. In extended reality (XR), it enables headsets to track user motion and maintain the stability of virtual content. For drones, it supports flight and navigation in areas where GPS is unreliable. In autonomous driving, visual SLAM complements GPS and LiDAR to enhance localization and scene understanding. Similarly, humanoid and mobile robots use it to navigate, manipulate objects, and interact safely within dynamic environments.
Across these domains, the common goal is to allow a system to perceive and localize itself from its own viewpoint while adapting to challenges such as rapid motion, changing illumination, and dynamic scenes. Egocentric visual SLAM thus forms a key bridge between perception and intelligent action in embodied systems.

1.2. Importance of Compression

Always using raw or more accurately uncompressed data in VSLAM enables the best case or most reliable scenario for tracking. The obvious downside to this is the high amount of space needed to store these files and also the increased demand on bandwidth while handling them. The idea to compress the visual files to save space and bandwidth comes to mind. The big downsides of doing that are the introduction of visual artifacts and the necessary computation time for the compression.
To separate between two use cases at this point: Using VSLAM in an "online"/"realtime" setting or in an "offline"/"video" setting. Video compression is of great importance for both. In realtime settings the available bandwidth for transmitting all sensor data is usually the performance bottleneck. For a system to be able to keep up a really fast connection is needed, which is not really possible to do wirelessly. But even with a wired connection it is hard to push those amounts of data. To reduce the bitrate compression can be applied which enables fast and reliable transmission even over the air.
In an offline setting speed is not the greatest concern (even though it shouldn't take forever either) so why is compression still relevant? Developing a vSLAM system that is able to handle the visual degradation of its input data and still perform well enough enables the usage of not only carefully curated training data like the datasets used in this study but also

opens the door to using a massive pool of videos. Basically every video that is stored on the internet is compressed to some degree so being able to handle artifacts is a necessary step to enable the usage of this enormous pool of data and information.

1.3. Problems and Opportunities

When using video compression in SLAM systems, several challenges arise that can significantly affect performance. Compression inherently removes high-frequency visual information, such as fine textures and sharp edges, which are often essential for reliable feature detection and matching. Moreover, temporal and spatial artifacts are introduced by prediction based encoding and can distort visual cues which leads to false correspondences between frames. These effects reduce photometric consistency and hinder both feature-based and direct SLAM methods that rely on accurate brightness constancy. In section 3.1 video compression and it's effects are explained in greater detail.
In addition to these problems or challenges applying compression to the visual input of a SLAM system can help tracking as well. The motion vectors generated by the encoder can be utilized for calculating tracking guesses in the frontend of a SLAM system which can lead to an improvement in tracking accuracy as is shown later in the results chapter. Also video compression not only removes details like fine textures from the raw material but can also remove noise which in turn can improve FAST detection.

1.4. Contributions

The contributions of this thesis can be summarized in three categories. Each of which will be explained further in the methodology chapter and their respective results will be shown in the results section. Hopefully these contributions will help others in improving their visual SLAM systems or spark ideas for further research.
The main focus of this thesis is to compare video encoders for egocentric visual tracking. Both with the usually used raw image files and also between different encoders with multiple sets of settings. The goal being able to understand the effects of video compression on the visual tracking. We perform a thorough benchmark on different compression configurations, and give an analysis on the size vs quality trade offs from a tracking perspective. The results can be seen in section 4.2.
While working on the mentioned goal we found that the reduction in file size by using compression significantly reduces the runtime of the tracking algorithm. The reason why that may happen and the exact findings can be found in section 4.3. This result leads to some interesting ideas, like applying quick/light compression to leverage the speed advantage for tracking, that are explained further in the future works chapter.
Finally, we show that codec data present in compressed videos can be exploited for improved tracking quality. In particular, we leverage motion vector data as priors for a sparse optical

flow-based frontend in the tracking system. We show that using this information as priors for the optical flow optimizations significantly improves tracking accuracy. The results are shown and further explained in section 4.4.

2. Related Work

A lot of work and research has been and still continues to be put in improving the capabilities of VSLAM. This thesis stands on the shoulders of that research. In this chapter the most important research that is adjacent to the topic of the thesis will be mentioned and explained so that the reader is able to set all findings of this study into context with the current state of the field.

2.1. Video Compression Artifacts

Compressing the visual data used for VSLAM saves enormous amounts of space but also introduces visual anomalies that can harm the results of visual tracking. These anomalies are called artifacts. The challenge of identifying and quantifying the types of artifacts and their intensity is described in [1]. Therefore different types of spatial and two types of temporal artifacts are specified. This study also focuses on these artifacts.

2.1.1. Spatial Artifacts

Spatial artifacts are artifacts that get created by Block-based video coding schemes due to block partitioning and quantization. The spatial artifacts, with different visual appearances, can be identified without temporal reference.

- 1) Blurring: Aiming at a higher compression ratio, the HEVC encoder quantizes transformed residuals discrepantly. When the video signals are reconstructed, high frequency energy may be severely lost, which may lead to visual blur. Perceptually, blurring usually appears as the loss of spatial details or sharpness of edges or texture regions in an image.
- 2) Blocking: The HEVC encoder is block-based, and all compression processes are performed within non-overlapped blocks. This often results in false discontinuities across block boundaries. The visual appearance of blocking may be different subject to the region of visual discontinuities.
- 3) Ringing: Ringing is caused by the coarse quantization of high frequency components. When the high frequency component of oscillating structure has a quantization error, the pseudo structure may appear near strong edges (high contrast), which manifests artificial wave-like or ripple structures, denoted as ringing.
- 4) Color bleeding: The chromaticity information is coarsely quantized to cause color bleeding. It is related to the presence of strong chroma variations in the compressed images leading to false color edges. It may be a result of inconsistent image rendering across the luminance and chromatic channels.

2.1.2. Temporal Artifacts

Temporal artifacts are manifested as temporal information loss, and can be identified during video playback.

- 1) Flickering: Flickering is usually frequent brightness or color changes along the time dimension. There are different kinds of flickering including mosquito noise, fine-granularity flickering and coarse-granularity flickering. Mosquito noise is high frequency distortion and the embodiment of the coding effect in the time domain. It moves together with the objects like mosquitoes flying around. It may be caused by the mismatch prediction error of the ringing effect and the motion compensation. The most likely cause of coarse-granulating blinking may be luminance variations across Group-Of-Pictures (GOPs). Fine-granularity flickering may be produced by slow motion and blocking effect.
- 2) Floating: Floating refers to the appearance of illusory movements in certain areas rather than their surrounding environment. Visually these regions create a strong illusion as if they are floating on top of the surrounding background. Most often, a scene with a large textured area such as water or trees is captured with cameras moving slowly. The floating artifacts may be due to the skip mode in video coding, which simply copies a block from one frame to another without updating the image details further.

2.2. Comparison of Encoding Speeds

2.3. Datasets

To be able to provide proper findings the results of using multiple different datasets needs to be used. Using datasets that reflect different use-cases for visual odometry and SLAM enables conclusions that apply to wider range of scenarios by reducing biases. For this reason and also to be able to compare results with other research only datasets that are from respectable publishers and are used by many researchers should be considered. For this study a dataset can only be used if it contains synchronized images of at least two cameras, although this limitation could be changed in the future by adding mono camera support to basalt, inertial measurement units (IMU), which are needed to compare the motion vector tracking results and a reliable ground-truth trajectory, which is necessary for calculating the accuracy of any tracking algorithm. Of course the sensor system and their spatial relation to each other is also necessary. Considering these factors the following three datasets were chosen:

The **Euroc Dataset** [2] uses a Micro Aerial Vehicle (MAV) in two different environments, a machine hall and a test room. It contains approximately 22 minutes of visual and inertial data split up into eleven different scenes with varying difficulty. As mentioned in [1] it is one of the most widespread benchmarks used in the field and provides a reasonable degree of difficulty. Also for this thesis the choice of datasets is limited to ones that follow the same structure/data format that is defined by EuRoC [2].

The **Monado SLAM dataset (MSD)** [3] aims to fill a gap in the literature by using a divers set of consumer product to cover the head-mounted VR space. One of them even has four cameras which is great to see the effects of compression when more than two cameras are used. It's the largest dataset used for this thesis and contains "5 hours and 15 minutes of real-world footage and challenging scenarios, including long and uninterrupted datasets of up to 40 minutes and fast-paced game play".
And lastly the **TUM-Vi Dataset** [4] using a handheld device for capture rounds out the mix of datasets. It contains approximately one hour and 30 minutes of footage in five different environments. But only one of them contains full ground truth so only the six sequences of the "room" scenario are usable for this thesis to be able to compare the results.

2.4. Basalt: Visual-Inertial Mapping with Non-Linear Factor Recovery

Measuring the differences in tracking accuracy and speed is obviously necessary for the analysis in this thesis. For this purpose we decided to use basalt [5] as a starting point and adapted the system to our needs. These modifications are based on the already modified version of basalt used in [3]. As all datasets used are tested to work great with basalt as shown here [3] it was chosen. To reduce complexity we decided to focus only on one VIO-System as adding the necessary changes to multiple systems was out of scope for this work. In theory adding compressed video support to systems like OKVIS2 should work just as well.

2.5. MovSlam

MoV-SLAM [6] is an important reference for this thesis as parts of their research are applicable. Their focus however lies heavily on "Using Motion Vectors for Real-time Single-CPU Visual SLAM" [6]. They use the motion vectors provided by H.264 in interesting ways and we used the same encoder settings using FFMPEG as they suggest. Our ability to leverage the motion vectors to the same degree was limited by the fact that they applied a patch to FFMPEG providing lower level access and enabling the leverage of some encoder data like the distance of the source frame to the current frames motion vector. These ideas are elaborated in the future works section.

3. Methodology

3.1. Video Compression

3.1.1. Encoding Formats

There are multiple viable options for encoding and compressing the lossless data for space saving. Namely H.264, H.265 and VP9 [7] will be used for this study. All three have advantages and disadvantages so there is no clear best choice among them. One could argue that H.264 due to its age and the improvements found in H.265 make it a clear worse choice but using H.264 enables to extraction of motion vectors. These are used by the encoder to save space by not saving each pixels data on each frame but rather only saving that for keyframes and using the motion of the camera to encode where a pixel should be in reference to the last keyframe. Even though H.264 was used in the end in the beginning the focus was on VP9 as a more modern and open source option so more general concepts of video compression will be explained along VP9 examples.

Block organization

Each frame is split up into blocks of different sizes that are used for the en- and decoding process. Different Formats use different block sizes and organize them in specific ways. Modern standards (like h.265 and vp9) usually take advantage of dynamically adjustable block sizes to be able to adapt to varying level of detail and motion in a scene. Areas with less detail can be encoded using big blocks to save space and high detail areas use small blocks to save as much detail as possible while still saving space. Older standards like h.264 use more rigid block organization. H.264 uses macroblocks of size 16 pixels by 16 pixels which can be further split into smaller 8×8 or 4×4 blocks for more detailed encoding. H.265 uses Coding Tree Units (CTUs), which can be up to 64×64 pixels and split into asymmetrical sub-blocks (not rigidly grid-like). VP9 uses a hierarchical structure where blocks can be up to 64×64 and recursively divided.

3.1.2. VP9

The VP9 video coding standard employs a block-based approach to exploit spatial redundancy within an image. The underlying motivation for its block partitioning strategy lies in the observation that different regions of a frame exhibit varying levels of spatial complexity. Regions with little or no motion, such as static backgrounds, can often be efficiently represented using larger blocks, whereas areas containing fine details or rapid changes in texture benefit from smaller block sizes to capture these variations more accurately.
Each frame in VP9 is decomposed into superblocks of size 64×64 pixels. The encoding pro-

cess for each superblock is governed by a partitioning structure that determines how it is subdivided. A superblock can either be encoded as a single 64×64 block, divided into two 64×32 or two 32×64 blocks, or split into four 32×32 blocks. The decoding of these blocks proceeds in raster scan order. Furthermore, each 32×32 block can itself undergo recursive partitioning down to an 8×8 block size. At the 8×8 level, several sub-partitioning options are available, including a single 8×8 block, two 8×4 or two 4×8 subblocks, or four 4×4 subblocks. An important distinction in VP9 is made between blocks and subblocks. A block is defined as an independent unit that carries its own header, referred to as mode info, whereas subblocks share a common block header within an 8×8 region. While subblocks may have different intra prediction modes or motion vectors, they share higher-level information, such as the reference frame used for prediction. This hierarchical partitioning mechanism allows VP9 to adaptively allocate coding resources to different regions of the image, improving both compression efficiency and visual quality.

3.1.3. H.265 (HEVC)

In contrast, the H.265/HEVC standard adopts a more flexible and hierarchical partitioning scheme based on Coding Tree Units (CTUs), which can be 16×16, 32×32, or 64×64 pixels in size. Each CTU can contain a combination of inter- and intra-coded regions, offering greater adaptability to spatial and temporal variations in the video content.
A CTU is recursively divided into Coding Units (CUs) using a quad-tree structure. Each CU represents a region that can be further subdivided into Prediction Units (PUs), which carry distinct motion information or intra-prediction modes. The partitioning of a CU depends on whether it is intra- or inter-coded. For intra-coded CUs, only square-shaped partition units are allowed; the encoder decides whether to maintain a single PU or divide the CU into four smaller PUs (for 8×8 CUs only). In contrast, inter-coded CUs may be partitioned into both square and non-square PUs, provided that no side is smaller than four pixels (excluding 4×4 PUs). This design results in eight possible partitioning configurations for inter-coded CUs, enabling efficient representation of diverse motion patterns and object boundaries.

Decoding Blocks

Both VP9 and H.265 employ predictive coding techniques that reduce redundancy by transmitting only the information necessary to reconstruct a block, rather than the block's full pixel data. Two principal modes of prediction are used: intra-frame prediction (spatial compression) and inter-frame prediction (temporal compression). The difference between the predicted and the actual block is encoded as a residual.
In VP9, each block is represented by parameters describing how to predict its contents, along with transform coefficients corresponding to its residual signal. During decoding, the prediction is first reconstructed, and then the inverse transform is applied to the residual coefficients. The final block is obtained by adding the residual to the prediction. Even for the initial blocks

of a sequence—where no previously decoded pixels exist—the decoder assumes fixed-value pixels in the off-screen regions to maintain consistency.

Intra-Block Prediction (Spatial Compression)

Intra-frame prediction exploits spatial correlations within a single frame. The contents of a block are predicted from its neighboring, already-decoded blocks, and only the residuals (differences) between the actual and predicted values are stored. This technique performs particularly well in homogeneous image regions, although it may struggle in areas containing complex textures or high-frequency details.

An intra mode specifies the direction or structure of the prediction. During decoding, the intra mode is first read, and the decoder uses it to filter reference samples from previously decoded pixels in the same frame to form a prediction. Typical intra modes include vertical, horizontal, and various diagonal orientations (e.g., 45°). H.264 defines ten intra prediction modes—eight angular, one DC, and one planar—while H.265 extends this set to 35 modes, consisting of 33 angular, one DC, and one planar mode.

Angular prediction interpolates pixel values from reference samples along a specific direction; DC prediction assigns a constant value equal to the average of neighboring reference pixels; and planar prediction computes a smooth gradient by averaging horizontal and vertical predictions.

Removing Intra Artifacts:

Intra prediction can introduce visual artifacts, particularly at block boundaries or near edges in the reference sample arrays. To mitigate these, H.265 employs both pre-processing and post-processing techniques.

Pre-processing aims to smooth reference samples before prediction. This reference sample smoothing reduces contouring artifacts and can be implemented using either a three-tap filter or a stronger intra-smoothing method that incorporates corner reference pixels. The application of smoothing depends on both the PU size and the prediction mode.

Post-processing focuses on boundary smoothing to address discontinuities along block borders introduced by intra prediction. For DC prediction, a three-tap filter is applied to the first row and column of the prediction, while a two-tap filter is used for horizontal and vertical predictions.

Inter-Block Prediction (Temporal Compression)

Inter-frame prediction, also known as temporal compression, exploits temporal redundancy between consecutive frames to achieve higher compression efficiency. Instead of storing the entire content of a block anew, the encoder searches for a visually similar region in one or more previously encoded frames. The location of this region is described by a motion vec-

tor, which specifies the displacement between the current block and its reference block in the previous frame. Only this motion information and the corresponding residual signal (the pixel-wise difference between the prediction and the actual block) are transmitted.
The basic principle is that a block can often be well approximated by shifting a block from a previously decoded frame. For example, static regions in a scene—such as a stationary background—are typically represented by zero motion vectors, indicating that the block can be directly reused without modification. This approach drastically reduces the amount of data required to represent temporally correlated image regions.
The decoding process for inter blocks follows the same general procedure as for intra blocks. First, the prediction is generated based on the reference frame and motion vector. The residual transform coefficients are then inverse-transformed, and the resulting residual is added to the predicted block to obtain the reconstructed output. This consistent processing structure simplifies both encoder and decoder implementations.
Motion vectors in VP9 and H.265 can specify displacements with either integer- or fractional-pixel precision. When fractional-pixel motion is used, the reference frame must be interpolated to compute the predicted block with sub-pixel accuracy. To this end, an interpolation filter is applied to the reference frame. The characteristics of this filter—specifically its bandwidth—affect the balance between detail preservation and noise suppression. For noisy source material, a narrow-bandwidth filter is preferred, as it attenuates high-frequency noise that would otherwise degrade prediction quality. Conversely, for clean or high-quality sources, a high-bandwidth filter can be employed to preserve fine textures and details. The interpolation filter can be globally defined for an entire frame or adaptively selected on a per-block basis, providing additional flexibility in optimizing compression performance.

3.1.4. Deciding on compression standard

Choosing between the different compression standards was complicated as they all have their merits. In the beginning of this study VP9 was used because it is a modern standard that is developed and used by Google in their products [7]. Also it is open source. But the speed advantage of using H.264 and H.265 can't be overlooked. They are so much faster because most computing devices have some sort of hardware accelerated support for H.264, H.265 and AV1 but not for VP9.
After shifting focus to use the motion vectors to help the tracking algorithm it was necessary to use H.264 as it is the only one that makes the motion vectors used easily accessible by ffmpeg.

3.1.5. Using Motion Vectors as prior for feature tracking

As explained in the previous section Motion vectors are created by video encoders to save space by saving the movement of sections of the frame instead of all the pixels. They are stored by saving the direction of the source frame (in our case it's only previous frames), the dimensions of the macroblock, the source and destination position. There are also additional

information saved but they are not necessary for this use case. How the motion vectors are calculated and what they mean exactly is explained in the previous section. In this chapter only the usage for this thesis is explained.

3.1.6. Optical Flow

In a Visual SLAM (vSLAM) system, the frontend is responsible for processing raw camera images to extract motion and geometric information in real time. Its main goal is to estimate how the camera moves between consecutive frames and to track the visual features that will later be refined and optimized by the backend. Within this frontend pipeline, optical flow plays a central role as it provides the apparent motion of image points across frames, allowing the system to infer how the camera has moved in 3D space.

Optical flow refers to the pattern of apparent motion of brightness structures in an image sequence. It is based on the assumption that the brightness of a point in the image remains constant between consecutive frames while its position changes due to motion. Mathematically, this is often expressed as $I(x, y, t) \approx I(x+u, y+v, t+1)$, where u and v represent the horizontal and vertical displacements of a pixel between two frames. By estimating this displacement field, one can infer how features in the scene have moved relative to the camera.

Since basalt [5] is used as the vSLAM system the frontend begins with FAST feature detection, where distinctive and repeatable points are extracted from the image. These detected key points serve as the initial set of visual anchors that the system will attempt to track over time.

Once the features are detected, optical flow tracking is used to determine where each feature has moved in the next frame. The most widely used approach for this purpose is the Lucas–Kanade [8] optical flow method, which assumes that motion within a small image patch is constant and seeks the displacement vector (u,v) that minimizes the photometric error between the two patches. This minimization problem can be solved efficiently as a linear least-squares system, and to handle larger motions, the algorithm is often applied in a multi-scale (pyramidal) fashion. This coarse to fine approach is used because the Lucas-Kanade method relies on the assumption that the motion between two frames is really small. Otherwise the linear constraint equation $I_x + I_y + I_t \neq 0$ is not valid anymore. By using lower resolution versions of the frames the amount of motion in pixels is reduced. So if the frame is of resolution $N \times N$ and a pixel moved 100 pixel from the current frame to the target frame by lowering the resolution to $N/2 \times N/2$ to amount is also divided by two so the pixel only moved 50 pixels. This is repeat for multiple levels of lowered resolution to guarantee a small enough motion on the lowest level. The resulting estimate is then propagated as initialization to the next finer level, where it is refined further.

The result is a set of displacement vectors that describe how each tracked feature has moved from one frame to the next. For outlier rejection basalt tracks patches from the current frame to the target frame and back to check consistency. [5]

Since this is an iterative nonlinear optimizer the system requires a good initialization to find a solution and to find it in a small number of iterations. If the starting guess for (u,v) is too far from the true displacement, the linearization is no longer accurate — and the iterative updates will either converge to a wrong local minimum or diverge completely.

3.1.7. Why do Motion Vectors Help?

By default the frontend only uses the previous pixel positions in order to calculated optical flow. Since the data is encoded and compressed some visual details get lost irrecoverably. This loss in detail and the introduction of visual artifacts can lead to great difficulty in the aforementioned optical flow calculations. As shown in the chapter about compression and artifacts (3.1) this visual degradation expresses itself firstly by reducing fine details and leading to bigger section of the same brightness. These regions are especially difficult to handle by traditional optical flow strategies if they are big enough. The big advantage of using the video encoders motion vectors is that the encoding process takes place over the raw data, meaning that these motion estimations can be calculated before this degradation happens. The motion vectors may have information that is not recoverable from just the pixels.
As mentioned before these better guesses for the motion of elements in the frame leads to a higher likeliness to converge or in other words to find the feature. Furthermore a better guess means less iterations so a reduction in computation time could be achieved. The improvements in tracking accuracy and tacking speed will be shown in the Results chapter 4

3.2. Implementation

In this chapter all implementation work that was done for this thesis is mentioned. Most of the contributions should be usable for further work and integrate nicely into their respective original projects. Thanks to all authors on who's work these additions were based. More detailed information on the original works can be found in the related works chapter. It makes sense to explain the EuRoC dataset format here since it plays a big role for the implementations explained in this chapter. As mentioned in section 2.3 all datasets used in this thesis need to follow the EuRoC format. This decision was made to be able to assume the structure and contents of the datasets which made the implementations manageable for the scope of this thesis. Adding support for different dataset formats could be added (as long as they contain everything need as explain in section 2.3) but was decided against to reduce complexity. Also since the EuRoC format is widespread this limitation was not an issue as there are many datasets available that follow this format.
As explained in [2] each dataset contains one or more "sensor systems" and each sensor system can contain several "sensors". Every sensor comes with a sensor.yaml file that specifies its' calibration parameters and a data.csv file that either contains the sensor measurements or points to data files in the optional data folder. Example:

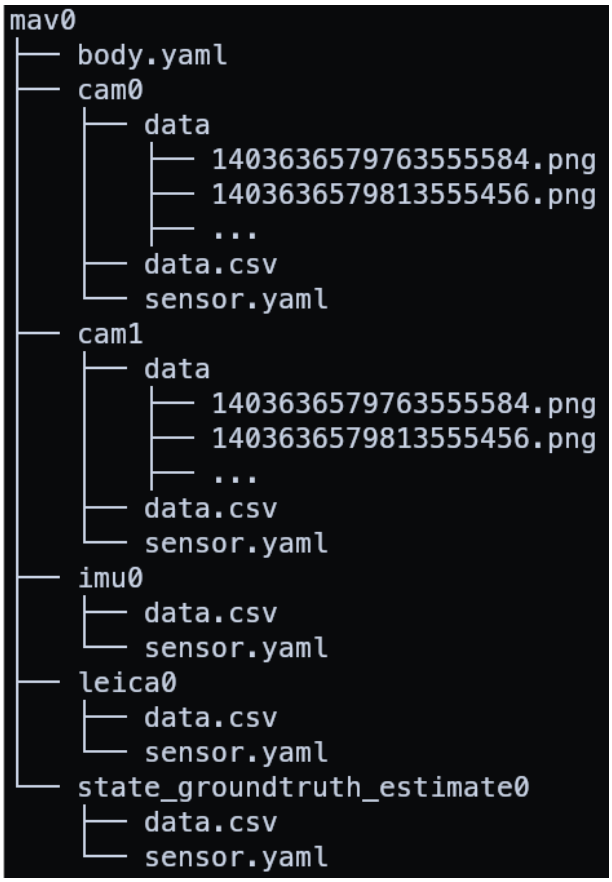

Figure 1 EuRoC File Structure

The EuRoC dataset format explained above assumes that all visual data of the cameras is stored as separate images in the data directory of each camera. To follow as close to this format as possible but still use videos instead of separate images the overall structure is kept the same for this project. Only the data directories are replaced by a single video file called "data.mp4" or "data.webm" per camera. Keeping the rest of the format the same allows all parts of Basalt except the reading of frames to function the same as before and no additional changes need to be made to the system. For this conversion a python script was written that receives a directory containing a dataset following the EuRoC format. Based on this file structure a single video file is created for every camera. All videos in a single dataset are compressed using the same settings. The script ignores all other data. There are also modes for compressing multiple datasets automatically. Also it is possible to compress the same dataset multiple times with different settings. Finally basalt can be run automatically on the compressed dataset and the user can decide to keep the dataset and the results or delete the dataset to save space and only keep the tracking results.

As mentioned before Basalt [5] was chosen as the VIO system. More specifically Basalt for Monado [3] a fork of Basalt improved for tracking XR devices. Upon this foundation the support for using datasets containing video files instead of directories of images was added. For this adaptation OpenCV's VideoCapture class was used to read frames of each video file representing one camera. So multi camera support is still available. To save memory the

video frames are read during runtime instead of fully capturing them on startup. To guarantee synchronization between image streams the frame numbers are not calculated by their timestamp since this can lead to shifts due to rounding and generally is not precise enough. Rather the timestamp information that are provide in the data.csv files defined for EuRoC datasets [2] are used. Sequential reads are always possible and run a lot faster than random reads but with this system random reads are also possible. They are not necessary for tracking but are needed to use Basalt's GUI to jump through the sequence. For this reason the system checks if the next frame in the stream is the frame that is supposed to be read next. If that is the case it can be read sequentially. The implementation changes can be found here: `https://gitlab.freedesktop.org/mateosss/basalt/-/merge_requests/72`.
The extraction of motion vectors is based on the project mv-extractor by Lukas Bommes [9] that uses ffmpeg libraries to create a replacement for OpenCV's VideoCapture class. The only changes made to the project were to substitute deprecated ffmpeg functions/code to the proper modern versions. Those changes will be added to a merge request to the original project. For now they can be found here: `https://github.com/tut14/mv-extractor`. As mentioned on the projects github page, this system can not only read the motion vectors of a frame but also return the decoded frame as an BGR image. This functionality meaning that the usage of OpenCV's VideoCapture mentioned above could be substituted by the custom one provide by mv-extractor. Currently both versions are used simultaneously OpenCV's for reading the image and mv-extractor's for reading the motion vectors. This is necessary because as mentioned above random reads need to be possible for Basalt's GUI to work but mv-extractor currently only provides sequential reading. For this substitution to be possible seeking would need to be added to mv-extractor. As this adds a lot of complexity and might introduce synchronization issues it was decides to stick to this approach using both VideoCaptures.
Similarly to the image the motion vectors are read during runtime. When frame to frame optical flow is calculated the system checks if there are motion vectors stored for the current frame. If so they are used alongside the traditional optical flow to find tracking guesses. This works with fallbacks. The user can specify which method (traditional optical flow or motion vectors) should be used first. After the tracking of points is done with the first calculation method the second method is used on all keypoints that were lost/dropped in the first calculation. All keypoints that were recovered in that process get added. The effectiveness of this approach can be seen in the Results chapter (4).

3.2.1. Preparation of Datasets

To be able to reliably compare the effects of different kinds and levels of compression a system to automatically convert image based datasets to videos was needed. For that a script was written that grabs a fresh dataset and encodes all images for each camera to separate video files and saves them in the same euroc dataset structure. For that FFMPEG is used and different sets of encoding settings can be specified. With this system in place each dataset used was encoded five times to the following settings:

4. Results

4.1. Chosen Datasets and Compression Settings

The available datasets shown in section 2.3 were narrowed down to a pool of 13 sequences across six different environments. They contain all easy sequences from EuRoC [2] and MSD [3]. Also the three sequences with the best scoring tracking accuracy when using the default optical flow on the raw png version. Except Room1 which was an outlier and diverged heavily in the compressed state. This choice was made to showcase the improvements on sequences that already had good tracking using the default optical flow as the difference in ate or rte doesn't mean a lot if the tracking diverges in both cases. The difference between the default optical flow and the usage of motion vectors was explained in section 3.1.6 .
The decision on what compression settings to use was focused on trying to use settings applicable to real world scenarios while also keeping compareability between runs. The compromise was to use the average bitrate setting as the main differentiator and letting the encoder try to achieve the best quality possible while adhering to the bitrate. Setting the average bitrate still leaves the encoder with the possibility to dynamically adjust the bitrate frame by frame depending on the contents. So in scenarios where a lower bitrate is sufficient the encoder can stick to that to be able to use a higher bitrate for more complicated scenes containing a lot of motion or high-frequency details for example. Based on the desired file size with regard to the duration of the sequence a bitrate can be chosen. To investigate the effects across a range of bitrates/filesize four settings wore chosen: 100k, 500k, 1000k and 5000k leading to approximate file size with the sequences used of an average 2MB, 7MB, 14MB and 69MB respectively. To achieve accurate bitrates two pass encoding was used. These bitrate settings were used across the widely used H.264 and the more modern and open source VP9. With H.264 also using the suggested settings from mov-slam [6] to control motion vector behavior. The exact file sizes can be seen in table 1. Example FFMPEG commands:

```
$ 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:bframes=0:ref=1
     -pass 1 -an -f null /dev/null && \
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:bframes=0:ref=1 -
    pass 2 -an -f mp4 MH_01_easy/mav0/cam0/data.mp4

$ ffmpeg -y -framerate 20 -pattern_type glob -i MH_01_easy/mav0/
    cam0/data/*.png -c:v libvpx-vp9 -b:v 100k -minrate 50k -maxrate
    145k -pix_fmt yuv420p -pass 1 -an -f null /dev/null && \
ffmpeg -y -framerate 20 -pattern_type glob -i MH_01_easy/mav0/cam0/
```

```
data/*.png -c:v libvpx-vp9 -b:v 100k -minrate 50k -maxrate 145k
-pix_fmt yuv420p -pass 2 -an MH_01_easy/mav0/cam0/data.webm
```

4.2. Compression Effects on Tracking Accuracy

As explained earlier in the thesis using video compression is interesting for vSLAM for multiple reasons. Firstly there are scenarios where the reduction in file size simply is necessary due to hardware limitations. A possible online or realtime example is not being able to perform the tracking on device but needing to send the data over the air. Knowing the effects on tracking accuracy also impacts offline use cases. If a given vSLAM system is able to handle the problems introduced by video compression and still perform well enough it is possible to use this system on the enormous pool of video data that is collected on the internet. Almost all of which is compressed by some amount. Being able to use vSLAM on those videos could simplify the analysis of the content with downstream learning for neural networks.
The data pool shown in this chapter was selected to show the trade offs between different encoders and encoder settings and how they affect tracking accuracy. The results show a comparison between the modern open source encoder VP9 and the widely used but older H.264. A more detailed explanation of the choices can be found in the previous section.
In this section the focus lies solely on comparing the tracking accuracy results of video and png datasets while only using basalts original optical flow. With this data it can be shown how much different compression settings effect the tracking accuracy while only relying on the optical data, so ignoring the motion vectors leverage explained earlier. The results for that are explained in section 4.4. Additionally to the file size table (table 1) the relevant tables for this section are: 2, 3, 4 and 5, all of which can be found at the end of the thesis.
Looking at the data in the tables and also the following figures (2 ,3, 4 and 5) it is easy to see that when using big bitrates the ate and rte are really close to the original png version. As predicted in the introduction the errors are even lower in some sequences. When VP9 is used the bitrate necessary to become on par or better with the png version is even lower. Using a compressed video instead of raw PNGs reduced the ate in eight or nine sequences depending if H.264 or VP9 was used and the rte in seven sequences. The individual improvements of errors are relatively small but when considering that the data of each camera is only 10% of its original size when using the highest bitrate tested for this thesis (5000kb) the fact that the errors are even close to the png version is great. This finding is promising for any research considering the usage of compression for vSLAM.
The comparison of the different bitrate versions shows that reducing the bitrate increases the errors which is expected. Interestingly it can be seen that going from 5000k to 1000k there are sequences that still work really well especially when using VP9 but there are some outliers here that can be classified as unusable. The MIO sequences used seem to struggle as specially. These outliers are exemplified when looking at the H.264 Versions where MIO07 diverges completely. It should be noted that some those results seem to improve when lowering the bitrate further which shows that in these lower bitrates depending on the scenario it

can be hard to predict how the tracking will perform.

Now to focus on the bottom end of the spectrum, the 100k bitrate. Here the results are not only unreliable with high errors but there even are sequences that fully crash. These results show that there is a limit to what bitrates are necessary to perform optical flow. Also there is a point of diminishing returns as the reduction in file size is already so big when using higher bitrates. Regarding these findings a bitrate of 500k seems to be the sweet spot between save big amounts of file size while keeping the errors manageable. If the reduction in file size provided by a 5000k bitrate is enough for a given use case it is the obvious choice as the errors are looking really promising as mentioned above.

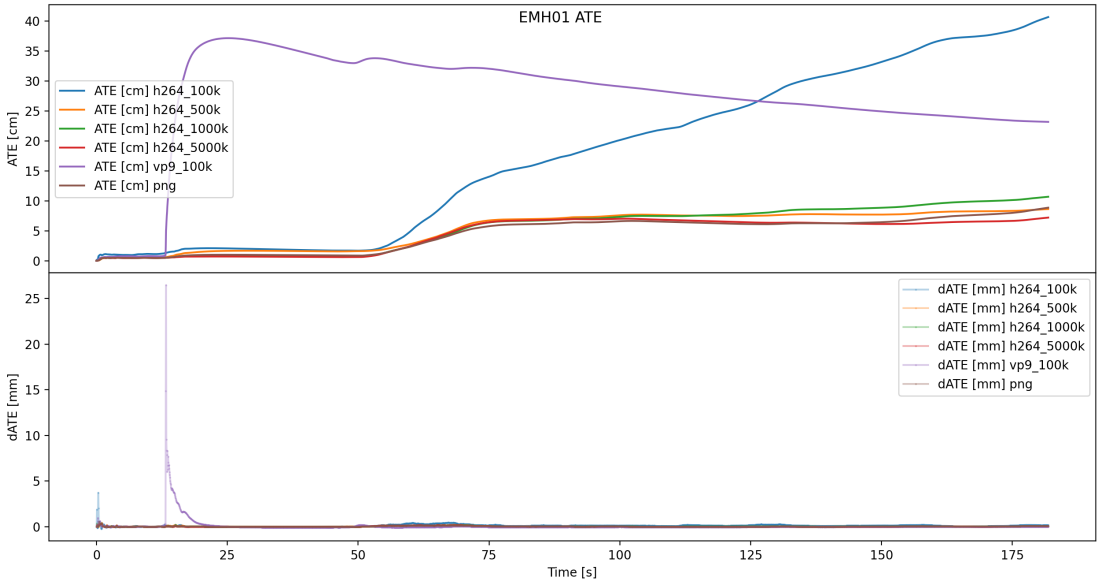

Figure 2 Optical Flow comparison EMH01 ate

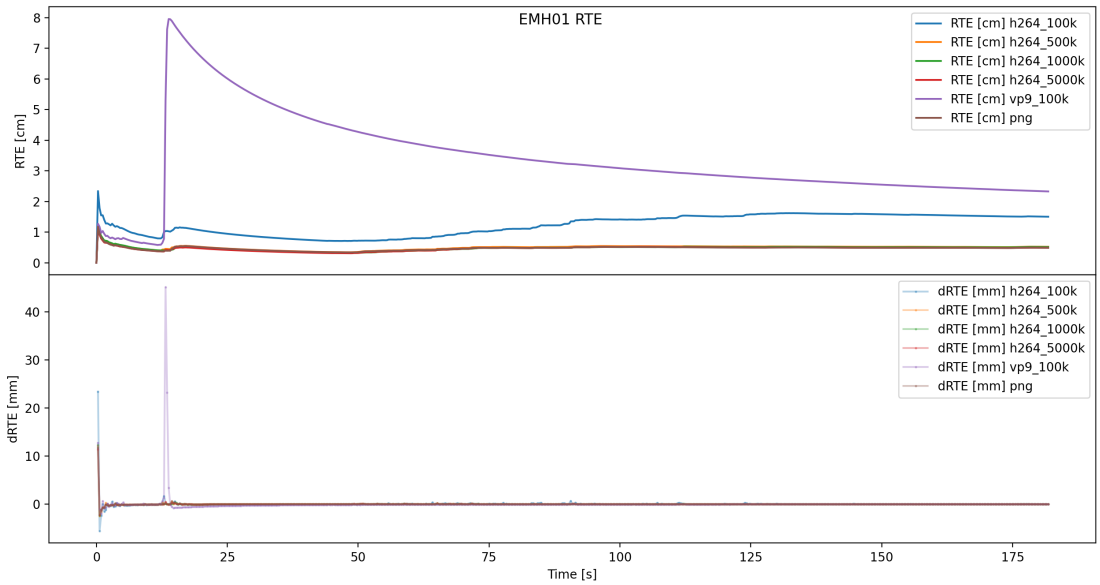

Figure 3 Optical Flow comparison EMH01 rte

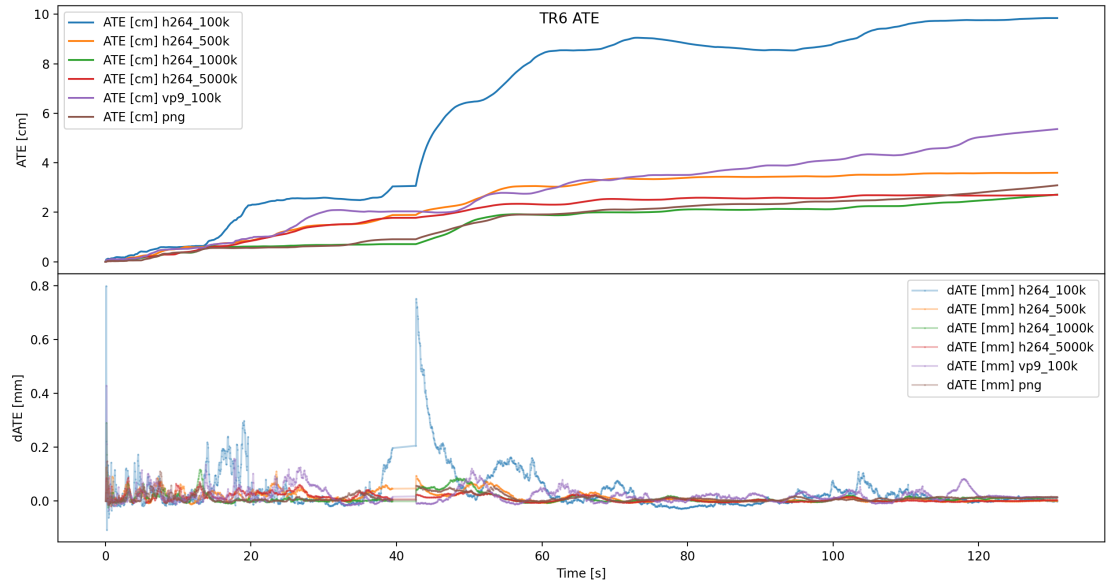

Figure 4 Optical Flow comparison TR6 ate

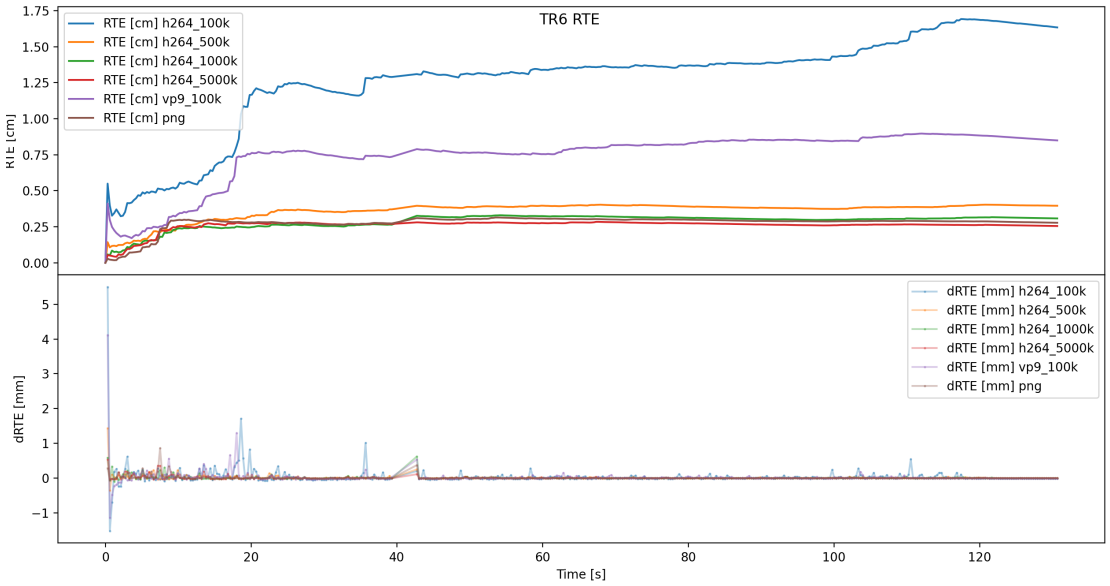

Figure 5 Optical Flow comparison TR6 rte

4.3. Compression Effects on Tracking Speed

There are multiple possible reason why using compressed videos instead of raw PNGs might affect tracking speed in vSLAM. The two that seem the most reasonable are the reduction in file size and the usage of motion vectors. The reduction in file size could increase tracking speed by reducing loading times and freeing up space in memory while the tracking is calculated. As the file size is decreased by such a big amount (as can be seen in table 1) it seems that this affects the tracking speed the most. The other possible reason is that by using motion vectors created by the encoder leads to an improvement of tracking guesses which in turn leads to a higher likeliness to converge or in other words to find the feature. Although it

seems this does not have a big impact on tracking speed as it is so heavily affected by the reduction in file size.
In this section the focus lies on comparing the full end-to-end run times of basalt when using videos as input and when using PNGs. Both versions use optical flow without motion vectors to be comparable. As can be seen in table 6 the execution time of basalt is reduced by an average of 38.27%. The video datasets were encoded with the same settings as in the other runs and the displayed times are when using H.264 with a bitrate of 500k. This set was chosen as it is the recommended one for tracking accuracy as mentioned in section **??**. Also the differences in tracking speed between different encoders and compression settings is negligible.
The test were run on a desktop computer running Linux Mint 22.1 Cinnamon with a 12th Gen Intel© Core™ i7-12700KF, 31.2 GiB of Memory on a Samsung 844 SATA SSD.

4.4. Accuracy Improvements with Motion Vector Prior in Frontend

As mentioned before the idea for this section was to leverage the motion vectors that are created by encoders when compressing video files in the feature tracking stage of basalts frontend. A more in depth explanation of how that works can be found in section 3.1.5. Looking at the tables 7 and 8 where the difference of ATE and RTE between using only optical flow and recovering tracked features with motion vectors can be seen. Here you can see that using motion vectors in combination with optical flow is not only better in 10 of the 13 datasets used but also there is an reduction by 60% in the average ATE and 28% in the average RTE. This shows a significant improvement in tracking accuracy when leveraging motion vectors compared to traditinal optical flow. So the reduction in visual fidelity that is introduced by video compression can possibly be salvaged by this technique.
The figures 6 and 7 show plots that compare using only optical flow for the tracking and combining traditional optical flow with motion vectors. As a reference point optical flow on the PNG version was also added. You can see how much better the mvof version performs especially in preventing the spike at the 20s mark. So rescuing features that would have been lost in the optical flow version prevented the tracking from doing such a significant error. //

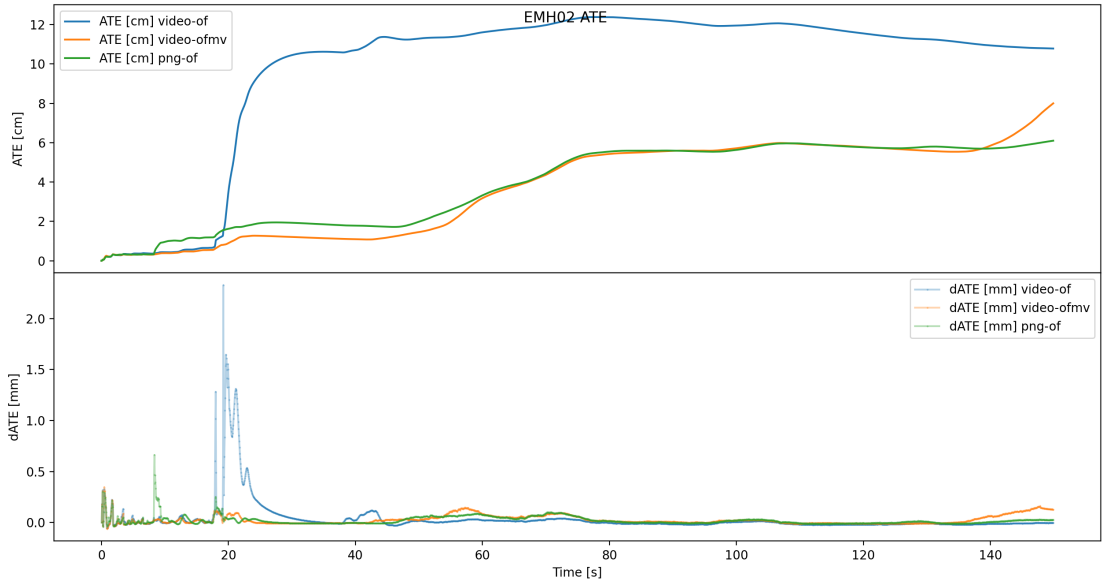

Figure 6 EMH02 Motion Vector Comparison ATE

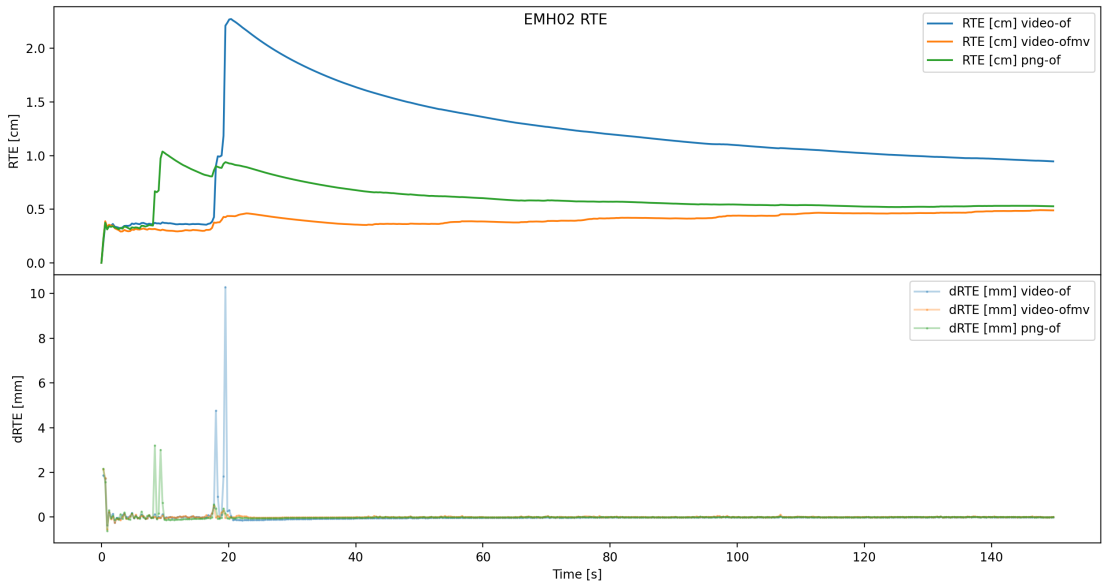

Figure 7 EMH02 Motion Vector Comparison RTE

5. Conclusion and Future Work

5.1. Conclusion

This thesis investigated the impact of video compression on egocentric visual tracking in visual SLAM systems. The aim was to understand how different video encoders and compression settings affect tracking performance and computational efficiency.
The first contribution is a systematic comparison of several video encoders and compression levels, revealing clear trade-offs between file size, visual quality, and tracking accuracy. The second contribution shows that compression can substantially improve runtime performance, as reduced data size and image complexity lead to faster processing. This finding suggests potential for using light compression to accelerate tracking without severely degrading accuracy.
The third contribution demonstrates that codec motion vectors from compressed video can be exploited to improve tracking quality. By using them as priors for sparse optical flow, we achieved more accurate and stable tracking results.
Overall, this work provides a comprehensive view of how video compression influences both the efficiency and accuracy of visual tracking. The findings highlight new opportunities for compression-aware SLAM systems that leverage encoder information to achieve better performance in real-world, resource-limited applications.

5.2. Future Work

Lastly to add what future work could expand the findings present in this thesis and build upon this more fundamental research. The goal of this thesis was to investigate how much video compression effects visual odometry. Now that some basic findings in that regard were made here follow some ideas how to continue from here. At first there are a couple of ideas what other information stored by video encoders could be leveraged for vSLAM.// Something left out in this thesis was bi-directional motion vectors. The focus lied solely on motion vectors with a source in the past. But encoders don't just create those they can also create motion vectors based on future frames. The bi-directional motion vectors could be used for offline video tracking or even delayed-realtime tracking to improve the quality and robustness of motion vectors used for visual odometry.
Another idea could be to use iframes created by encoders and used as keyframes for the compression process as a reason to trigger a keyframe in the tracking system. This could improve the choice of keyframes as the video encoder bases it's choice for i-frames on the same core values as a vSLAM system. So trusting that these encoders are able to tell which frames are of great quality for a keyframe might be a helpful.
Out of scope for this thesis but definitely possible would be to keep track of the motion vectors

reference frames. H.264 not only saves the direction of the source but also the distance so the exact reference frame of a motion vector can be calculated. This information could then be use to instruct the tracking system which frames are still important to keep. Meaning if there still is a motion vector in the current frame with a reference to a previous frame it could be useful to still keep that previous frame around. This would function as an additional source of information when deciding what frames are still needed which is a hard to solve problem.
To conquer blurry low contrast features an idea would be to utilize histogram equalization. By applying this image processing technique the overall contrast of the frame can be increased which in turn should help with some of the artifacting explained in the thesis. It would be interesting to compare the results of using histogram equalization with increasing the minimum FAST response threshold.
Instead of software encoders used in this thesis you could focus on leveraging hardware-encoding-apis like NVENC, VA-API, D3D12 Motion Estimation API, Vulkan Optical Flow, OpenCL cl_intel_motion_estimation from vendors for feature tracking in the frontend. The speed advantage of using them over software encoders could make the idea of applying a quick compression before feature tracking to speed up visual odometry as was shown is possible viable in the real world.
Another way to leverage encoder information for slam could be Macroblock-based slam. This effectively turns the encoder into an asynchronous and hardware-accelerated frontend.

Bibliography

[1] L. Lin, S. Yu, T. Zhao, and Z. Wang, *Pea265: Perceptual assessment of video compression artifacts*, 2019. arXiv: 1903.00473 [cs.CV]. [Online]. Available: https://arxiv.org/abs/1903.00473.

[2] M. Burri, J. Nikolic, P. Gohl, *et al.*, "The euroc micro aerial vehicle datasets," *The International Journal of Robotics Research*, 2016. DOI: 10.1177/0278364915620033. eprint: http://ijr.sagepub.com/content/early/2016/01/21/0278364915620033.full.pdf+html. [Online]. Available: http://ijr.sagepub.com/content/early/2016/01/21/0278364915620033.abstract.

[3] M. de Mayo, D. Cremers, and T. Pire, *The monado slam dataset for egocentric visual-inertial tracking*, 2025. arXiv: 2508.00088 [cs.CV]. [Online]. Available: https://arxiv.org/abs/2508.00088.

[4] D. Schubert, T. Goll, N. Demmel, V. Usenko, J. Stueckler, and D. Cremers, "The tum vi benchmark for evaluating visual-inertial odometry," in *International Conference on Intelligent Robots and Systems (IROS)*, Oct. 2018.

[5] V. Usenko, N. Demmel, D. Schubert, J. Stuckler, and D. Cremers, "Visual-inertial mapping with non-linear factor recovery," *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 422–429, Apr. 2020, ISSN: 2377-3774. DOI: 10.1109/lra.2019.2961227. [Online]. Available: http://dx.doi.org/10.1109/LRA.2019.2961227.

[6] R. N. Turner, N. K. Banerjee, and S. Banerjee, "Mov-slam: Using motion vectors for real-time single-cpu visual slam," in *2023 Seventh IEEE International Conference on Robotic Computing (IRC)*, 2023, pp. 51–58. DOI: 10.1109/IRC59093.2023.00015.

[7] J. H. .-. A. D. Adrian Grange - Google Peter de Rivaz - Argon Design, *Vp9 bitstream & decoding process specification - v0.7*, Version 0.7, Feb. 2017. [Online]. Available: https://storage.googleapis.com/downloads.webmproject.org/docs/vp9/vp9-bitstream-specification-v0.7-20170222-draft.pdf.

[8] B. Lucas and T. Kanade, "An iterative image registration technique with an application to stereo vision (ijcai)," vol. 81, Apr. 1981.

[9] L. Bommes, X. Lin, and J. Zhou, "Mvmed: Fast multi-object tracking in the compressed domain," in *2020 15th IEEE Conference on Industrial Electronics and Applications (ICIEA)*, 2020, pp. 1419–1424. DOI: 10.1109/ICIEA48937.2020.9248145.

List of Figures

List of Tables

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

Table 1 File Size of single camera in MB

sequence	h264 100k of	h264 500k of	h264 1000k of	h264 5000k of	png of
EMH01	0.407 ± 0.150	0.086 ± 0.024	0.107 ± 0.039	**0.072 ± 0.033**	0.089 ± 0.036
EMH02	1.197 ± 0.416	0.108 ± 0.059	0.173 ± 0.108	0.079 ± 0.030	**0.061 ± 0.021**
EV101	0.538 ± 0.139	0.077 ± 0.030	0.062 ± 0.021	0.053 ± 0.018	**0.050 ± 0.017**
EV201	0.481 ± 0.418	0.373 ± 0.323	0.263 ± 0.222	**0.063 ± 0.035**	0.119 ± 0.097
MGO05	—	0.027 ± 0.011	0.028 ± 0.014	**0.023 ± 0.010**	0.024 ± 0.008
MGO07	0.695 ± 0.308	0.024 ± 0.009	0.025 ± 0.009	0.021 ± 0.008	**0.021 ± 0.008**
MIO05	—	4.329 ± 0.693	0.569 ± 0.279	**0.146 ± 0.085**	0.699 ± 0.303
MIO07	—	1.003 ± 0.587	295.916 ± 187.672	**0.074 ± 0.041**	0.277 ± 0.219
MOO05	0.867 ± 0.336	0.028 ± 0.014	**0.021 ± 0.009**	0.024 ± 0.010	0.034 ± 0.010
MOO07	1.478 ± 0.514	0.047 ± 0.028	0.033 ± 0.018	**0.015 ± 0.007**	0.019 ± 0.010
TR2	21.877 ± 10.667	0.221 ± 0.153	0.098 ± 0.050	0.072 ± 0.034	**0.064 ± 0.028**
TR4	6.939 ± 2.541	0.941 ± 0.522	0.542 ± 0.466	0.335 ± 0.281	**0.032 ± 0.010**
TR6	0.099 ± 0.042	0.036 ± 0.015	0.027 ± 0.012	**0.027 ± 0.012**	0.031 ± 0.013
[AVG]	3.458 ± 1.553	0.562 ± 0.190	22.913 ± 14.532	**0.077 ± 0.047**	0.117 ± 0.060
[MED]	0.781 ± 0.376	0.086 ± 0.030	0.098 ± 0.039	0.063 ± 0.030	**0.050 ± 0.017**

Table 2 H.264 optical flow ate comparison

sequence	h264 100k of	h264 500k of	h264 1000k of	h264 5000k of	png of
EMH01	0.015031 ± 0.010029	0.005206 ± 0.003059	0.005148 ± 0.003011	0.004892 ± 0.002921	**0.004888 ± 0.002883**
EMH02	0.090413 ± 0.087491	0.009458 ± 0.008214	0.005810 ± 0.004378	**0.004668 ± 0.002686**	0.005270 ± 0.003485
EV101	0.039555 ± 0.031638	0.013495 ± 0.006726	0.013435 ± 0.006630	0.013089 ± 0.006275	**0.013076 ± 0.006273**
EV201	0.132614 ± 0.131463	0.120534 ± 0.119720	0.051988 ± 0.051521	**0.004998 ± 0.003774**	0.017876 ± 0.017058
MGO05	—	0.003669 ± 0.002109	0.003281 ± 0.001815	**0.003116 ± 0.001761**	0.003322 ± 0.001942
MGO07	0.187224 ± 0.170971	0.004754 ± 0.002754	0.004213 ± 0.002437	0.004103 ± 0.002344	**0.004093 ± 0.002327**
MIO05	—	0.115622 ± 0.114270	0.011861 ± 0.010803	**0.008556 ± 0.007947**	0.023714 ± 0.022937
MIO07	—	0.051140 ± 0.049302	13.412025 ± 13.351699	**0.023031 ± 0.022609**	0.041502 ± 0.041244
MOO05	0.115770 ± 0.104446	0.004964 ± 0.003399	0.004112 ± 0.002621	**0.003524 ± 0.002034**	0.003675 ± 0.002141
MOO07	0.228337 ± 0.198789	0.006388 ± 0.004247	0.005866 ± 0.004047	0.003982 ± 0.002239	**0.003932 ± 0.002234**
TR2	0.483442 ± 0.430578	0.031591 ± 0.029659	0.019234 ± 0.017886	0.010543 ± 0.009014	**0.006713 ± 0.005049**
TR4	0.470945 ± 0.431745	0.152574 ± 0.149296	0.059233 ± 0.056116	0.031360 ± 0.029331	**0.017003 ± 0.016332**
TR6	0.016851 ± 0.010076	0.004093 ± 0.002096	0.003225 ± 0.001781	**0.002872 ± 0.001492**	0.002980 ± 0.001601
[AVG]	0.178018 ± 0.160723	0.040268 ± 0.038065	1.046110 ± 1.039596	**0.009133 ± 0.007264**	0.011388 ± 0.009654
[MED]	0.124192 ± 0.117955	0.009458 ± 0.006726	0.005866 ± 0.004378	**0.004892 ± 0.002921**	0.005270 ± 0.003485

Table 3 H.264 optical flow rte comparison

sequence	vp9 100k of	vp9 500k of	vp9 1000k of	vp9 5000k of	png of
EMH01	0.232 ± 0.163	0.099 ± 0.032	0.087 ± 0.025	**0.086 ± 0.023**	0.089 ± 0.036
EMH02	3.597 ± 2.383	0.088 ± 0.037	0.068 ± 0.023	0.065 ± 0.034	**0.061 ± 0.021**
EV101	0.085 ± 0.035	0.055 ± 0.022	0.067 ± 0.024	0.056 ± 0.021	**0.050 ± 0.017**
EV201	—	0.067 ± 0.030	**0.085 ± 0.054**	0.072 ± 0.039	0.119 ± 0.097
MGO05	—	0.027 ± 0.011	**0.021 ± 0.008**	0.024 ± 0.010	0.024 ± 0.008
MGO07	0.071 ± 0.031	0.024 ± 0.009	0.022 ± 0.009	0.023 ± 0.009	**0.021 ± 0.008**
MIO05	—	**0.333 ± 0.107**	0.416 ± 0.293	0.400 ± 0.251	0.699 ± 0.303
MIO07	0.487 ± 0.241	0.132 ± 0.072	1.684 ± 1.087	**0.108 ± 0.063**	0.277 ± 0.219
MOO05	0.062 ± 0.035	0.030 ± 0.012	**0.022 ± 0.008**	0.025 ± 0.010	0.034 ± 0.010
MOO07	0.475 ± 0.315	0.019 ± 0.009	**0.016 ± 0.006**	0.016 ± 0.009	0.019 ± 0.010
TR2	38.296 ± 31.851	0.093 ± 0.046	0.103 ± 0.043	**0.060 ± 0.026**	0.064 ± 0.028
TR4	3.283 ± 1.227	0.597 ± 0.496	0.391 ± 0.321	0.034 ± 0.015	**0.032 ± 0.010**
TR6	0.054 ± 0.024	0.032 ± 0.013	0.037 ± 0.016	**0.030 ± 0.014**	0.031 ± 0.013
[AVG]	4.664 ± 3.630	0.123 ± 0.069	0.232 ± 0.147	**0.077 ± 0.040**	0.117 ± 0.060
[MED]	0.354 ± 0.202	0.067 ± 0.030	0.068 ± 0.024	0.056 ± 0.021	**0.050 ± 0.017**

Table 4 VP9 optical flow ate comparison

sequence	vp9 100k of	vp9 500k of	vp9 1000k of	vp9 5000k of	png of
EMH01	0.023282 ± 0.022050	0.005283 ± 0.003140	0.005010 ± 0.002923	0.005064 ± 0.003003	**0.004888 ± 0.002883**
EMH02	0.480347 ± 0.479470	0.006678 ± 0.005182	0.004821 ± 0.002968	**0.004282 ± 0.002289**	0.005270 ± 0.003485
EV101	0.015442 ± 0.008358	0.013391 ± 0.006611	0.013250 ± 0.006460	0.013125 ± 0.006307	**0.013076 ± 0.006273**
EV201	—	**0.007817 ± 0.006827**	0.012855 ± 0.012224	0.016469 ± 0.015797	0.017876 ± 0.017058
MGO05	—	0.003319 ± 0.001879	**0.003113 ± 0.001760**	0.003431 ± 0.002069	0.003322 ± 0.001942
MGO07	0.014327 ± 0.012322	0.004301 ± 0.002408	0.004214 ± 0.002445	0.004246 ± 0.002423	**0.004093 ± 0.002327**
MIO05	—	0.015626 ± 0.014967	**0.009966 ± 0.009301**	0.012587 ± 0.012004	0.023714 ± 0.022937
MIO07	0.034240 ± 0.028651	0.016203 ± 0.015466	0.159635 ± 0.159235	**0.009008 ± 0.008112**	0.041502 ± 0.041244
MOO05	0.014220 ± 0.012011	0.003708 ± 0.002170	0.003441 ± 0.001911	**0.003349 ± 0.001890**	0.003675 ± 0.002141
MOO07	0.041249 ± 0.038702	0.007580 ± 0.006203	0.003985 ± 0.002344	0.003933 ± 0.002351	**0.003932 ± 0.002234**
TR2	7.342164 ± 7.327002	0.056792 ± 0.055842	0.008094 ± 0.005772	0.006748 ± 0.004897	**0.006713 ± 0.005049**
TR4	0.249112 ± 0.229882	0.161232 ± 0.160136	0.065804 ± 0.064415	0.017829 ± 0.017122	**0.017003 ± 0.016332**
TR6	0.009554 ± 0.006247	0.003794 ± 0.001881	0.003383 ± 0.001711	**0.002934 ± 0.001492**	0.002980 ± 0.001601
[AVG]	0.822394 ± 0.816469	0.023517 ± 0.021747	0.022890 ± 0.021036	**0.007924 ± 0.006135**	0.011388 ± 0.009654
[MED]	0.028761 ± 0.025351	0.007580 ± 0.006203	**0.005010 ± 0.002968**	0.005064 ± 0.003003	0.005270 ± 0.003485

Table 5 VP9 optical flow rte comparison

sequence	video exec time in s	png exec time in s	improvement in %
EMH01	29.31	37.13	21.04
EMH02	24.84	32.32	23.12
EV101	22.54	28.78	21.67
EV201	16.23	21.49	24.46
MGO05	50.53	89.72	43.68
MGO07	18.96	34.41	44.91
MIO05	92.4	154.61	40.23
MIO07	54.95	93.38	41.15
MOO05	21.05	33.87	37.86
MOO07	8.33	15.65	46.77
TR2	19.04	76.98	75.27
TR4	16.4	25.89	36.67
TR6	18.5	31.17	40.66
[AVG]			38.27
[MED]			40.23

Table 6 basalt execution time comparison of video and PNG version

sequence	h264 500k of	h264 500k ofmv
EMH01	**0.086 ± 0.024**	0.111 ± 0.035
EMH02	0.108 ± 0.059	**0.080 ± 0.046**
EV101	0.077 ± 0.030	**0.061 ± 0.020**
EV201	0.373 ± 0.323	**0.141 ± 0.108**
MGO05	0.027 ± 0.011	**0.026 ± 0.010**
MGO07	0.024 ± 0.009	**0.024 ± 0.008**
MIO05	4.329 ± 0.693	**0.865 ± 0.442**
MIO07	1.003 ± 0.587	**0.376 ± 0.15**1
MOO05	0.028 ± 0.014	**0.024 ± 0.012**
MOO07	0.047 ± 0.028	**0.033 ± 0.021**
TR2	**0.221 ± 0.153**	0.680 ± 0.577
TR4	0.941 ± 0.522	**0.470 ± 0.295**
TR6	0.036 ± 0.015	**0.031 ± 0.013**
[AVG]	0.562 ± 0.190	**0.225 ± 0.134**

Table 7 motion vector ate comparions h264 500k

sequence	h264 500k of	h264 500k ofmv
EMH01	0.005206 ± 0.003059	**0.005173 ± 0.003029**
EMH02	0.009458 ± 0.008214	**0.004888 ± 0.002736**
EV101	0.013495 ± 0.006726	**0.013445 ± 0.006648**
EV201	0.120534 ± 0.119720	**0.023785 ± 0.023212**
MGO05	0.003669 ± 0.002109	**0.003558 ± 0.002047**
MGO07	0.004754 ± 0.002754	**0.004701 ± 0.002675**
MIO05	0.115622 ± 0.114270	**0.035178 ± 0.033448**
MIO07	0.051140 ± 0.049302	**0.046648 ± 0.044644**
MOO05	0.004964 ± 0.003399	**0.004949 ± 0.003265**
MOO07	**0.006388 ± 0.004247**	0.024085 ± 0.022939
TR2	**0.031591 ± 0.029659**	0.120909 ± 0.119994
TR4	0.152574 ± 0.149296	**0.086823 ± 0.083216**
TR6	**0.004093 ± 0.002096**	0.004196 ± 0.002338
[AVG]	0.040268 ± 0.038065	**0.029103 ± 0.026938**

Table 8 motion vector rte comparions h264 500k