# Visual Odometry

Bohdan Milian
*Computer Science, UCU*
*Lviv, Ukraine*
Zhovkva
milian.pn@ucu.edu.ua

Sofiia Sampara
*Computer Science, UCU*
*Lviv, Ukraine*
Ternopil
sampara.pn@ucu.edu.ua

Sofiia Popeniuk
*Computer Science, UCU*
*Lviv, Ukraine*
popeniuk.pn@ucu.edu.ua

Ostap Pavlyshyn
*Computer Science, UCU*
*Lviv, Ukraine*
pavlyshyn.pn@ucu.edu.ua

Yaroslav Korch (mentor)
*Computer Science, UCU*
*Lviv, Ukraine*
korch.pn@ucu.edu.ua

***Abstract*—Visual Odometry (VO) is a crucial technique for estimating a camera's trajectory in a three-dimensional environment using an image sequence. This project aims to develop an efficient and accurate monocular visual odometry system, with a focus on real-time performance, feature extraction, motion estimation, and pose updating. The implementation will be benchmarked against OpenCV-based solutions, rigorously tested on real-world datasets, and optimized through parallelization and GPU acceleration.**

***Index Terms*—RANSAC, fundamental matrix, feature matching, feature extraction**

## I. Introduction

Visual odometry (VO) is a key technique in autonomous navigation Despite its advantages, VO faces challenges such as feature robustness, scale ambiguity in monocular setups, computational efficiency, and error accumulation over time. To address these, VO employs feature-based, direct, or hybrid methods for motion estimation.

This project aims to develop a real-time monocular VO system, focusing on feature extraction, motion estimation, and pose updating, benchmarked against OpenCV-based implementations and optimized for performance through parallelization and GPU acceleration.

## II. Problem Statement

The goal of this project is to estimate the camera's poses in SE(3) space using only image sequences. This involves:

- Detecting features and matching them across frames.
- Estimating motion via epipolar geometry and Perspective-n-Point (PnP).
- Updating the camera pose based on incremental transformations.

To improve accuracy and efficiency, the system may incorporate additional sensor inputs such as Inertial Measurement Units (IMU).

**Expected Outcome**

- A working monocular visual odometry system capable of real-time operation.
- Quantitative performance comparison with OpenCV's implementation.
- A foundation for further developments in SLAM and sensor fusion.

## III. Project Objectives

Methodology
1. Feature Extraction
2. Naive Feature Matching
3. Fundamental matrix via RANSAC
4. Polishing of F (after RANSAC)

## IV. Feature Extraction

Feature extraction in visual odometry refers to the process of identifying and selecting distinct points in images that can be reliably tracked across multiple frames.

### A. What is a corner

A corner is a point in an image where two edges meet at an angle, making it a distinct and well-localized feature. There are a couple of more reasons why we are taking it as an attribute we want to track:

- **Distinctiveness** – Corners are unique in their local neighborhood, making them easier to recognize in different images.
- **Scale and Rotation Invariance** – corner detector will find angles that remain stable under changes in scale, rotations, and a slight change in illumination.
- **Feature Matching** – Because the corners are distinct, they are used to match key points between multiple images of the same place.

### B. Preprocessed the image

Colorful images are represented in the RGB spectrum, where each pixel consists of three intensity values corresponding to the Red, Green, and Blue color channels. These three channels collectively define the color composition of the image. We convert the picture into a gray-scale reducing computational complexity and avoiding inconsistent gradient responses across channels.

The formula for gray-scale conversion:

$$\text{Gray} = 0.299R + 0.587G + 0.114B$$

Next, we want to reduce unneeded noise which may be present in our image due to sensor imperfections, lighting

variations and compression artifacts. Those defects interfere with edge and corner detection by introducing unwanted intensity disparity, making corner detection less reliable.

To address this, we apply Gaussian smoothing using a Gaussian kernel[1] and convolution[2]. The Gaussian kernel is defined as:

$$G(x,y) = \frac{1}{2\pi\sigma^2} \exp\left(-\frac{x^2+y^2}{2\sigma^2}\right)$$

The convolution is performed by sliding the Gaussian kernel over the image, multiplying its values with the corresponding pixel intensities, and summing the results. Mathematically, this operation can be expressed as:

$$I'(x,y) = \sum_{i=-k}^{k} \sum_{j=-k}^{k} I(x-i, y-j) \cdot G(i,j)$$

### C. Finding the direction gradients

After the image was preprocessed, we are ready to find the directional gradients of pixel intensities. Corners are identified based on changes in intensity in multiple directions.

This process is done by computing the weighted *sum of squared differences* (SSD) between these two patches, denoted $S$, is given by:

$$S(x,y) = \sum_u \sum_v w(u,v) \left(I(u+x, v+y) - I(u,v)\right)^2$$

Using Taylor expansion, we obtain:

$$I(u+\Delta u, v+\Delta v) \approx I(u,v) + \begin{bmatrix} J_x & J_y \end{bmatrix} \begin{bmatrix} \Delta u \\ \Delta v \end{bmatrix}$$

Which leads to:

$$f(x,y) \approx \sum_{(u,v)\in W_{xy}} \left( \begin{bmatrix} J_x & J_y \end{bmatrix} \begin{bmatrix} \Delta u \\ \Delta v \end{bmatrix} \right)^2$$

Written in matrix form as:

$$f(x,y) \approx \sum_{(u,v)\in W_{xy}} \begin{bmatrix} \Delta u \\ \Delta v \end{bmatrix}^\top \begin{bmatrix} J_x^2 & J_x J_y \\ J_x J_y & J_y^2 \end{bmatrix} \begin{bmatrix} \Delta u \\ \Delta v \end{bmatrix}$$

Move the sums inside the matrix:

$$f(x,y) \approx \begin{bmatrix} \Delta u \\ \Delta v \end{bmatrix}^T \begin{bmatrix} \sum_W J_x^2 & \sum_W J_x J_y \\ \sum_W J_x J_y & \sum_W J_y^2 \end{bmatrix} \begin{bmatrix} \Delta u \\ \Delta v \end{bmatrix}$$

Where M = $\begin{bmatrix} \sum_W J_x^2 & \sum_W J_x J_y \\ \sum_W J_x J_y & \sum_W J_y^2 \end{bmatrix}$ is called a **structure matrix** and $J_x$ $J_y$ $J_x J_y$ are partial derivatives over some region $W$.

[1]Kernel - a small matrix used to apply some transformation to a portion of pixels in convolution operations.

[2]Convolution - a process of applying a kernel to an image by sliding it over the pixel grid, multiplying the kernel values with corresponding pixel intensities, and summing the results.

Those derivatives are computed as follows:

$$\mathbf{J}_x = \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} * \left( \begin{bmatrix} 1 & 0 & -1 \end{bmatrix} * \mathbf{A} \right)$$

$$\mathbf{J}_y = \begin{bmatrix} +1 \\ 0 \\ -1 \end{bmatrix} * \left( \begin{bmatrix} 1 & 2 & 1 \end{bmatrix} * \mathbf{A} \right)$$

$$\mathbf{J}_x \mathbf{J}_y = \begin{bmatrix} 1 \\ 0 \\ -1 \end{bmatrix} * \left( \begin{bmatrix} 1 & 0 & -1 \end{bmatrix} * \mathbf{A} \right)$$

Where $\mathbf{A}$ is a matrix of the region $W$

We introduce the eigenvalues $\lambda_1$ and $\lambda_2$, which are defined:

$$\lambda_{1,2} = \frac{\text{trace}(M) \pm \sqrt{\text{trace}(M)^2 - 4\det(M)}}{2}$$

Depending on those values of the structure matrix $M$ we will have:

- $\lambda_1 \approx 0$, $\lambda_2 \approx 0$ Little to no variation in any direction. This characterize a *flat region*
- $\lambda_1 \gg \lambda_2$ (or vice versa). One dominant gradient girection. This characterize an *edge of an image*
- Both eigenvalues are large: $\lambda_1$ and $\lambda_2$ are significantly greater than zero. This means there is strong intensity change in multiple directions, *characteristic of a corner*.

### D. Response functions

For now, we chose two response functions

1) **Harris corner detection** where the response is given by:
$$R = \lambda_1\lambda_2 - k(\lambda_1 + \lambda_2)^2$$
where $k$ is an empirical constant
2) **Shi-Tomasi corner detection** where the response is given by:
$$R = min(\lambda_1, \lambda_2)$$

### E. Choosing only valid corners

Having found the response matrix, now only valid corners have to be chosen. This is achieved by applying a threshold to filter out weak responses, keeping only points where the function is big enough.

$$Threshold = 0.01 \cdot \max(R)$$

Next, using non-maximum suppression, we identify local maxima within a neighborhood window, ensuring that only the strongest corner responses are retained while suppressing weaker or redundant ones. The remaining key points represent the most prominent features in an image.

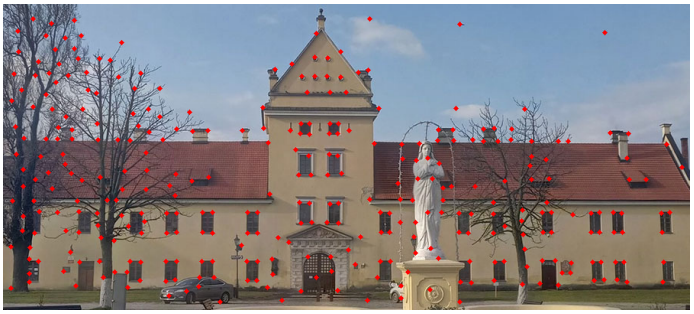

Fig. 1. Detected corners with the Shi-tomasi response function

### F. Feature descriptors

A feature descriptor is a representation of a feature point that captures its local appearance in a way that is invariant to transformations such as rotation, scaling, and illumination changes. They allow us to have a reliable feature matching across different images despite changes.

### G. FREAK: Fast retina keypoint descriptor

The FREAK feature descriptor is a robust and efficient method that we use to describe keypoints in images. This algorithm is inspired by the sampling pattern of the human retina, which allows its efficient computation.

*1) Choosing the neighborhood to describe the key point:* After the testing done by the authors of FREAK, it is stated that by mimicking the fragmentary distribution of photoreceptors in the human eye, the algorithm captures more detail around critical regions and less in areas that are less informative. Then we construct tests[3] which are done in the form:

$$T = \begin{cases} 1, & \text{if } I(P_{r_1}) < I(P_{r_2}) \\ 0, & \text{otherwise} \end{cases}$$

Where $P_{r_1}$ and $P_{r_2}$ are the intensity values in the location of predefined points.

[3]Test - binary intensity comparison between two predefined sample points in the key point's neighborhood

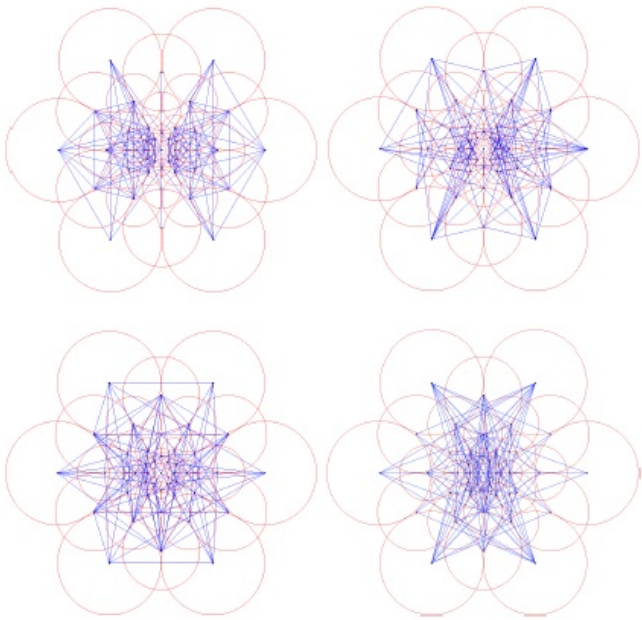

Fig. 2. The tests in the form of retinal sampling

*2) Reaching the rotation invariance:* Rotation invariance in FREAK is achieved by first estimating the dominant orientation of a keypoint and then aligning the sampling pattern accordingly. The angle of the main intensity direction is calculated as:

$$O = \frac{1}{M} \sum_{P_o \in G} \left(I(P^o_{r_1}) - I(P^o_{r_2})\right) \frac{P^o_{r_1} - P^o_{r_2}}{\|P^o_{r_1} - P^o_{r_2}\|}$$

$M$ is the number of predefined points in $G$ $P^o_{r_i}$ is a 2D vector of points in the patch.

## V. Feature Matching

After extracting descriptors for the key points detected in each frame, the next step is to establish correspondences between them.

### A. What is a Matcher?

A matcher is a technique or algorithm used to establish correspondences between feature points in two different frames. These correspondences are vital for reconstructing the scene's geometry and estimating the camera's motion between frames.

### B. Brute-Force Matcher

Brute-force matchers are straightforward, but computationally intensive. They compare each descriptor of one image with every descriptor from the next image and select the best match based on a chosen distance metric.

For binary descriptors such as FREAK, we use the Hamming distance, which counts differing bits between two binary vectors:

$$H(A, B) = \sum_{i=1}^{n} (A_i \oplus B_i)$$

For floating-point descriptors, the Euclidean distance is used:

$$d(A, B) = \sqrt{\sum_{i=1}^{n} (A_i - B_i)^2}$$

While faster alternatives like FLANN or hashing-based matchers exist, we opted for brute force due to its simplicity, and compatibility with binary descriptors. Performance overhead is alleviated by multithreading.

### C. Matching Algorithm Steps

1) **Descriptor Matching**
   For each descriptor $d_1^i$ of the first image:
   - Compare against all descriptors $d_2^j$ in the second image using Hamming distance:

$$H(d_1^i, d_2^j) = \sum_{k=1}^{N} \left( d_1^i[k] \oplus d_2^j[k] \right)$$

   - Select the best and second-best matches $j_1$ and $j_2$.
2) **Apply Ratio Test**
   Accept match $(i, j_1)$ only if:

$$\frac{H(d_1^i, d_2^{j_1})}{H(d_1^i, d_2^{j_2})} < \text{threshold}, \quad \text{where threshold} = 0.75$$

3) **Parallel Execution**
   - The descriptors are divided into blocks.
   - A thread pool distributes blocks across threads for concurrent processing.
4) **Cross-Validation with OpenCV**
   - Use `cv::BFMatcher` with `cv::NORM_HAMMING`.
   - Evaluate match overlap using Jaccard similarity:

$$J = \frac{|\mathcal{M}_{\text{custom}} \cap \mathcal{M}_{\text{OpenCV}}|}{|\mathcal{M}_{\text{custom}} \cup \mathcal{M}_{\text{OpenCV}}|}$$

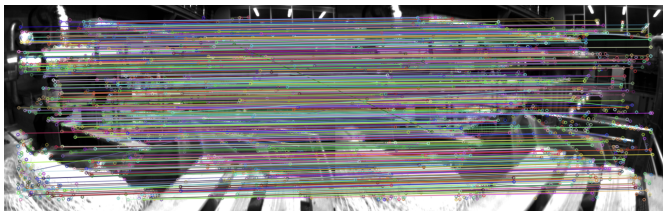

Fig. 3. Feature matches between two frames. Although the dense colored lines obscure the image content, their variety indicates a large number of successful keypoint correspondences.

## VI. RANSAC and the 8-Point Algorithm for Fundamental Matrix Estimation

### A. Robust Estimation with RANSAC

In real-world computer vision tasks, such as estimating the fundamental matrix between two views, a significant portion of feature correspondences may be corrupted by noise or outliers. Direct least-squares estimation methods fail under such conditions. The Random Sample Consensus (RANSAC) algorithm addresses this challenge by iteratively sampling subsets of data and selecting the model that best explains the majority of the data (the inliers).

RANSAC Overview:

1) Randomly sample a minimal subset of point correspondences (8 points for the fundamental matrix).
2) Estimate the model parameters (i.e., compute a candidate fundamental matrix).
3) Measure how well the model fits all data points using an error metric (e.g., Sampson error).
4) Count the number of inliers — points with error below a predefined threshold.
5) Repeat the above steps for a fixed number of iterations or until convergence criteria are met.
6) Select the model with the highest inlier count and re-estimate the parameters using all inliers.

The number of required iterations $k$ to ensure a probability $p$ of selecting an all-inlier subset is given by:

$$k = \frac{\log(1-p)}{\log(1-(1-\epsilon)^s)}$$

where $\epsilon$ is the outlier ratio and $s$ is the sample size (8 for the 8-point algorithm).

### B. The 8-Point Algorithm

The 8-point algorithm is a classic linear method for computing the fundamental matrix $\mathbf{F}$ given at least 8 point correspondences between two images. Each pair of corresponding points $(\mathbf{x}, \mathbf{x}')$ satisfies the epipolar constraint:

$$\mathbf{x}'^{\top} \mathbf{F} \mathbf{x} = 0$$

where $\mathbf{x} = [x, y, 1]^{\top}$, $\mathbf{x}' = [x', y', 1]^{\top}$ are homogeneous coordinates.

*Algorithm Steps::*

1) Normalization: Improve numerical stability by translating and scaling points so that their centroid is at the origin and the average distance to the origin is $\sqrt{2}$.
2) Construct the Matrix $\mathbf{A}$: For each correspondence, compute a row in matrix $\mathbf{A} \in \mathbb{R}^{n \times 9}$ such that:

$$\mathbf{A}_i = \begin{bmatrix} x_i' x_i & x_i' y_i & x_i' & y_i' x_i & y_i' y_i & y_i' & x_i & y_i & 1 \end{bmatrix}$$

3) Solve for $\mathbf{f}$: Find the right singular vector corresponding to the smallest singular value of $\mathbf{A}$, yielding a vector $\mathbf{f}$ which is reshaped into $\mathbf{F}$.
4) Enforce Rank-2 Constraint: Since the fundamental matrix must be of rank 2, perform SVD on $\mathbf{F}$ and set its smallest singular value to zero.
5) Denormalize: Apply the inverse of the initial normalization transforms to get the final $\mathbf{F}$.

*Sampson Error:* To evaluate the quality of a fundamental matrix, the Sampson error is used as an approximation of the geometric reprojection error. For a correspondence $(\mathbf{x}, \mathbf{x}')$, the Sampson error is:

$$\text{Err}(\mathbf{x}, \mathbf{x}') = \frac{(\mathbf{x}'^{\top}\mathbf{F}\mathbf{x})^2}{((F\mathbf{x})_1^2 + (F\mathbf{x})_2^2 + (F^{\top}\mathbf{x}')_1^2 + (F^{\top}\mathbf{x}')_2^2)}$$

This metric allows robust inlier filtering during RANSAC.

## VII. Pose estimation

The core idea is to reconstruct the relative pose from the fundamental matrix, which encodes epipolar geometry between two views. The estimated pose is then transformed into the body frame and scaled using a known ground truth displacement to ensure consistency in the trajectory.

### A. Input data and calibration parameters

The algorithm begins with the following inputs:

- Fundamental matrix F computed from matched keypoints.
- Matched image points points1 and points2 from the previous and current frames.
- Camera intrinsic matrix K, distortion coefficients, and known transformations:
  - $\mathbf{T}_{\text{BS}}$ – transformation from camera to body frame.
  - $\mathbf{R}_{\text{CS}}$ – camera-to-sensor frame rotation (because axes of camera and sensor coordinates differ).
  - Previous pose $(\mathbf{R}_{\text{prev}}, \mathbf{t}_{\text{prev}})$ in world coordinates.
  - Ground truth norm of position displacement (used for scale recovery).

### B. Essential matrix decomposition

The essential matrix E is computed from the fundamental matrix and the intrinsic matrix:

$$\mathbf{E} = \mathbf{K}^{\top}\mathbf{F}\mathbf{K}$$

A Singular Value Decomposition (SVD) is then applied to E to extract two possible rotation matrices $(\mathbf{R}_1, \mathbf{R}_2)$ and a translation vector t (up to scale). Four possible configurations of (R, t) are generated:

$$(\mathbf{R}_1, \quad \mathbf{t}), (\mathbf{R}_2, \quad \mathbf{t}), (\mathbf{R}_1, \mathbf{t}), (\mathbf{R}_2, \mathbf{t})$$

### C. Disambiguation using cheirality check

To select the physically correct pose, the algorithm triangulates 3D points for each candidate using undistorted image points and projection matrices $\mathbf{P}_1$ and $\mathbf{P}_2$ derived from the candidate pose.

A cheirality check counts how many points lie in front of both cameras (i.e., have positive depth values in both views). The candidate with the highest number of valid 3D points is chosen as the correct pose.

### D. Transformation to body frame

The estimated pose is originally expressed in the camera frame. It is converted to the body frame using:

$\mathbf{T}_{\text{body}} = \mathbf{T}_{\text{BS}} \cdot \mathbf{T}_{\text{cam}} \cdot \mathbf{T}_{\text{BS}}^{-1}$

Where:

- $\mathbf{T}_{\text{body}}$ is a $4 \times 4$ transformation matrix constructed from the chosen $\mathbf{R}$ and $\mathbf{t}$.
- $\mathbf{T}_{\text{BS}}$ is the known transformation from the body frame to the stereo camera frame.

Then we receive estimated $\mathbf{R}$ and $\mathbf{t}$ from $\mathbf{T}_{\text{cam}}$ and apply them to $\mathbf{R}_{\text{prev}}$ and $\mathbf{t}_{\text{prev}}$.

## VIII. Parallelization

### A. Why use parallelization

Each of the subprocesses will require a substantial amount of time to compute with the high-dimensional pictures. Thus, this project needs to improve its performance that involves taking additional resources from the computer. Since each step: corner detection, descriptor generation, and matching is dependent on one another, parallelization only lets you boost the execution of each respective part.

### B. Parallel implementation

The *thread pool* is used for our needs, which safely lets each part be decomposed into smaller sub-tasks while retaining the threads alive. Obviously, before the execution of the next part of the pipeline, we wait for each and every piece of work to end not causing data races.

### C. GPU acceleration

Furthermore, the feature extraction stage leverages GPU acceleration to enhance computational performance. To achieve this, we employed the **OpenCL** framework, selected for its broad device compatibility and strong cross-platform support. Each component of the pipeline is designed with a high level of safety and robustness. The feature extraction process involves a large number of fully independent operations at the pixel level, making it particularly well-suited for parallel execution on GPUs.

### D. SIMD Optimization

In addition to multithreading, our matching algorithm benefits from low-level SIMD acceleration. Since Hamming distance involves XOR and bit counting across binary vectors, it naturally maps to vectorized operations. Using intrinsics significantly reduces the per-comparison cost and boosts overall performance, particularly on large descriptor sets.

## IX. Performance estimation

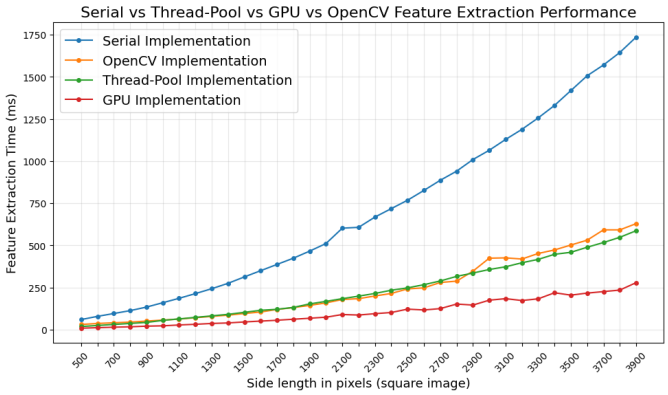


Fig. 4. Time performance of Feature Extraction using different computation methods

*1) Feature Extraction performance:*

- For the *serial implementation*, we utilized the initial codebase presented during the first presentation.
- For the *OpenCV-based implementation*, we used the Shi-Tomasi corner detection method and the FREAK feature descriptor provided by the native OpenCV library, which is running in parallel by design.
- For the *thread-pool implementation*, we adopted the codebase developed and presented during the second presentation.
- For the *GPU-based implementation*, we applied our current OpenCL-based implementation.

The number of threads allocated for testing was selected taking into account the available CPU core count.

As demonstrated by the plot, GPU programming is particularly well suited for assignments of this nature. The task can be efficiently formulated in terms of matrix operations, such as convolutions and box filters.

## X. The hurdles we faced during our work

**Limitations in synchronization with NVIDIA GPUs**. During the implementation phase, several challenges were encountered when using OpenCL on NVIDIA GPUs. In particular, the absence of atomic floating point operations in OpenCL C presented a significant limitation. Additionally, the use of custom structures resulted in undefined behavior, further complicating development and debugging.

## XI. Limitations of Our Visual Odometry Implementation

Despite its utility in estimating camera motion from image sequences, monocular visual odometry faces several limitations.

- **Lack of scale or ground truth.** Monocular VO cannot recover the absolute scale of motion without additional information. Ground truth poses and integration with external sensors such as an IMU are required for scale recovery.
- **Sensitivity to dynamic scenes.** If moving objects are present in the scene, the algorithm may incorrectly interpret their motion as ego-motion, leading to inaccurate estimation of the fundamental matrix and camera pose.
- **Error accumulation over time.** Visual odometry gradually accumulates error due to small inaccuracies in feature matching and motion estimation. Over long trajectories, this leads to significant drift in the estimated path.
- **Challenges in texture-poor environments.** In scenes with low texture, features are sparse and less distinctive, making matching and pose estimation unreliable.

## XII. Our progress from the last presentation

1) Feature extraction computation with GPU acceleration
2) Reducing matching time by SIMD operations
3) RANSAC parallelization
4) Finished pose estimation

## References

[1] Andreas Geiger and Philip Lenz and Raquel Urtasun, Are we ready for Autonomous Driving? The KITTI Vision Benchmark Suite, https://www.cvlibs.net/datasets/kitti/eval_odometry.php.

[2] C. Stachniss A photography course at Youbute Available: https://www.youtube.com/watch?v=SyB7Wg1e62A&list=PLgnQpQtFTOGRYjqjdZxTEQPZuFHQa7O7Y.

[3] Original paper of FREAK feature descriptor A. Alahi, R. Ortiz, and P. Vandergheynst, "FREAK: Fast Retina Keypoint," 2012 IEEE Conference on Computer Vision and Pattern Recognition, Providence, RI, USA, 2012, pp. 510-517, doi: 10.1109/CVPR.2012.6247715. Available: https://www.researchgate.net/publication/258848394_FREAK_Fast_retina_keypoint/citations

[4] OpenCL for Parallel Programming of Heterogeneous Systems Available: https://www.khronos.org/opencl/