TECHNISCHE UNIVERSITÄT
BERGAKADEMIE FREIBERG
The University of Resources. Since 1765.

Faculty of Mathematics and Computer Science
Institute of Computer Science
Virtual Reality and Multimedia Group

Seminar paper

Comparison of Feature-Based Pose Estimation and Localization Methods in Dark Environments

Jonas Fleischer

Applied Computer Science
Specialization: Robotics

Matriculation register: 61146

August 30, 2024

Tutor/First Proofreader:
Prof. Dr. Bernhard Jung

Second Proofreader:
Robert Lösch

Comparison of Feature-Based Pose Estimation and Localization Methods in Dark Environments

Jonas Fleischer
Technische Universität Bergakademie Freiberg
Faculty of Mathematics and Computer Science
Institute of Computer Science
Virtual Reality and Multimedia Group
Bernhard-von-Cotta-Straße 2
09599 Freiberg

***Abstract*—This paper presents a comparative analysis of feature-based pose estimation and localization methods, focusing on traditional algorithms such as Scale-Invariant Feature Transform (SIFT), Oriented FAST and Rotated BRIEF (ORB), and Accelerated-KAZE (AKAZE). These methods are crucial for autonomous navigation in environments where Global Navigation Satellite Systems (GNSS) are unreliable or unavailable, such as indoor settings. Additionally, the focus is on dark environments where no visual data is available. Therefore, depth images captured from sensors like LIDAR are intended to be used. To ensure reliability during testing, synthetic test data generated from sensors designed for low-light environments were utilized. However, due to implementation errors, the expected results were not fully realized. While the algorithms themselves are known for their robustness and accuracy, the limitations observed in this study stem from the current state of the implementation rather than the inherent capabilities of the algorithms. This work highlights the importance of careful implementation and testing, and it outlines the next steps required to achieve accurate pose estimation in future work.**

I. Introduction

Autonomous localization and navigation of robots and autonomous vehicles constitute a central research topic in robotics and artificial intelligence. In many scenarios, such as underground mines, deep oceans, or space, Global Navigation Satellite Systems (GNSS), including GPS, are either unavailable or unreliable. Consequently, developing methods for autonomous localization and navigation that rely solely on visual data is of crucial importance [1], [2]. In this context, sensors that provide depth data, such as LiDAR, Radar, and Structured Light sensors, play a significant role. These sensors enable robots and autonomous vehicles to perceive their environment in 3D, thereby facilitating precise localization and navigation [1], [2]. However, dark environments pose a particular challenge, as they can significantly degrade the quality of visual data [1], [2]. Simultaneous Localization and Mapping (SLAM) is an active research area in mobile robotics, particularly for autonomous tasks based on a robot's perception in unknown environments [3]. In recent decades, research and development in autonomous robotics have predominantly focused on ego localization and the estimation of 3D robot poses over time within an unknown environment [3]. Initially, the Global Positioning System (GPS) provided an efficient solution for robot localization with satisfactory precision. However, GPS has limitations, particularly in indoor environments, which have led to its replacement by other sensors [3]. These emerging technologies offer greater flexibility in localization and are more easily adapted to environmental perception via SLAM and visual SLAM (vSLAM) [3]. SLAM systems perform two simultaneous tasks—localization and mapping—to achieve the necessary accuracy for robot localization and environmental perception [3]. Several approaches have been developed for feature-based pose estimation. Classical algorithms, such as Scale-Invariant Feature Transform (SIFT), Oriented FAST and Rotated BRIEF (ORB), and Accelerated-KAZE (AKAZE), have been widely utilized due to their robustness and efficiency [4]. These methods typically detect keypoints in images and describe them using feature descriptors, which are then matched between consecutive frames to estimate the relative pose [4]. In contrast, machine learning approaches, particularly those based on deep learning, have gained popularity in recent years. These methods leverage large datasets to learn feature representations and estimate poses directly from images. While they have demonstrated impressive results, they often require significant computational resources and extensive training data, which may render them less suitable for real-time applications in dark environments [5], [6]. Despite the advancements in machine learning, classical feature-based algorithms remain relevant due to their computational efficiency and robustness in various conditions. However, there remains a gap in the research regarding the performance of these classical algorithms in dark environments using depth images and point clouds. This paper aims to address this gap by evaluating the effectiveness of SIFT, ORB, and AKAZE algorithms for pose estimation under these challenging conditions. In this study, we will compare various feature-based methods for pose estimation and localization in dark environments. Our objective is to evaluate the performance of these methods and identify those most suited for use in such conditions. The focus will be on methods for processing depth data applicable to visual odometry and SLAM.

II. METHODS

A. Explanation of Approaches and Their Mathematical Backgrounds

a) SIFT (Scale-Invariant Feature Transform): The Scale-Invariant Feature Transform (SIFT) is an algorithm for detecting and describing local features in images. Developed by David Lowe in 1999, SIFT is known for its robustness to changes in scale, rotation, illumination, and noise [7]. The algorithm consists of four main steps:

1) **Scale-Space Extrema Detection:** The image is transformed into different scales to detect features of various sizes.
2) **Keypoint Localization:** Local maxima and minima in the scale-space are identified as potential features.
3) **Orientation Assignment:** An orientation is assigned to each keypoint to achieve rotation invariance.
4) **Keypoint Descriptor:** A descriptor is created for each keypoint, capturing the local image information around the keypoint.

b) AKAZE (Accelerated-KAZE): AKAZE is an algorithm for detecting and describing features, based on the KAZE method but significantly faster [8]. AKAZE uses nonlinear diffusion to create the scale-space representation and is known for its efficiency and accuracy. The algorithm consists of the following steps:

1) **Nonlinear Diffusion:** The image is filtered through nonlinear diffusion to detect features of various sizes.
2) **Feature Detection:** Local maxima and minima in the diffused image are identified as potential features.
3) **Feature Description:** A descriptor is created for each feature, capturing the local image information around the feature.
4) **Feature Matching:** The descriptors are used to match features between different images.

c) ORB (Oriented FAST and Rotated BRIEF): ORB is a fast and robust algorithm for detecting and describing features, based on the FAST and BRIEF methods [9]. ORB combines the speed of FAST with the robustness of BRIEF and adds an orientation component to make the features rotation-invariant. The algorithm consists of the following steps:

1) **Feature Detection:** The FAST algorithm is used to detect features in the image.
2) **Feature Description:** The BRIEF descriptor is used to describe the local image information around the features.
3) **Orientation Correction:** The features are oriented to achieve rotation invariance.
4) **Feature Matching:** The descriptors are used to match features between different images.

d) Descriptors and Detectors: A **detector** finds interesting points in an image, while a **descriptor** converts these points into numerical "fingerprints" that can be used for matching [10].

B. Depth Images and Flexion Images

In this study, we utilize two types of datasets: depth images and flexion images.

a) Depth Images: Depth images can be captured using various sensors, including LiDAR, RADAR, and structured light sensors. It is important to differentiate between traditional depth images, which are visual representations of the environment. These images can be continuous or consist of individual data points within the image file. Each pixel in a depth image represents the distance from the viewer to the surface of an object. In this study, we use continuous images that represent specific distances using grayscale values. Depending on the implementation of recording and processing, different configurations are possible. For instance, the mapping of distance from near to far can be represented from black to white or vice versa. Additionally, the specific sensor determines the distance range it can cover, and thus what black or white represents as the maximum distance value for a concrete maximum distance to the viewer. It is also possible to visualize depth data originating from a point cloud. The KITTI dataset is an example of this [11]. In this case, the depth images are not continuous but consist of pixel grids where each pixel represents the distance value of a measurement point. The underlying sensor in the case of the KITTI dataset apparently provides only individual, discrete measurement points 1.

b) Flexion Images: Flexion images are a further processing of depth images. In this case, depth images are used as the data basis. The image is iterated pixel by pixel, and the surrounding pixels of each pixel are considered. Since these represent a distance value, the flexion (curvature) of the surface can be determined. The normals are formed from the two diagonals and from the horizontal and vertical. The flexion of the considered pixel is then calculated using the equation:

$$\mathcal{F} = \|\vec{n}_1 \cdot \vec{n}_2\|_2 \tag{1}$$

where $\vec{n}_1$ and $\vec{n}_2$ are the normal vectors. Since the vectors $\vec{n}_1$ and $\vec{n}_2$ lie between 0 and 1, $\mathcal{F} \in [0, 1]$ (see Equation 1) [12]. This results in a representation that provides edges with high contrast. Surfaces that have the same distance to the viewer across the entire surface are colored uniformly. For surfaces that recede, a gradient is created. This representation leads to the hypothesis that these images provide more distinctive features for the feature detection algorithm, allowing it to better recognize the features.

C. Implementation of the Approaches

The implementation is based on a repository that provides a framework for feature detection and matching [13]. The following code snippets show the implementation of the SIFT, ORB, and AKAZE methods:

```
# Call function SIFT
def SIFT():
# Initiate SIFT detector
SIFT = cv.xfeatures2d.SIFT_create()
return SIFT
# Call function ORB
```

```
def ORB():
# Initiate ORB detector
ORB = cv.ORB_create()
return ORB
# Call function AKAZE
def AKAZE():
# Initiate AKAZE descriptor
AKAZE = cv.AKAZE_create()
return AKAZE
```

The created keypoints and matchings were then used in a custom implementation to reconstruct the pose using the OpenCV methods `findEssentialMat()` and `recoverPose()` [14].

1) Essential Matrix and OpenCV Functions: The Essential Matrix is a fundamental concept in computer vision, particularly in the context of stereo vision and structure from motion. It encapsulates the intrinsic geometry between two views and is used to relate corresponding points in stereo images. The Essential Matrix $\mathbf{E}$ is defined as:

$$\mathbf{E} = \mathbf{t} \times \mathbf{R}$$

where $\mathbf{t}$ is the translation vector and $\mathbf{R}$ is the rotation matrix between the two camera views [15]. The Essential Matrix maps points from one image to epipolar lines in the other image, enforcing the epipolar constraint [16].

a) findEssentialMat(): The 'findEssentialMat()' function in OpenCV estimates the Essential Matrix between two sets of points in stereo images. The function signature is:

```
cv::Mat findEssentialMat(InputArray points1,
 InputArray points2,
 InputArray cameraMatrix, int method = RANSAC,
 double prob = 0.999, double threshold = 1.0,
 OutputArray mask = noArray());
```

- **points1, points2:** Arrays of corresponding points between the two images.
- **cameraMatrix:** The intrinsic camera matrix.
- **method:** The method for computing the Essential Matrix (e.g., RANSAC).
- **prob:** The probability that the estimated matrix is correct.
- **threshold:** The maximum distance from a point to an epipolar line in pixels.
- **mask:** Output mask for inliers.

The function uses the RANSAC algorithm to robustly estimate the Essential Matrix by iteratively selecting random subsets of correspondences and computing the matrix that best fits the majority of points [17].

b) recoverPose(): The 'recoverPose()' function in OpenCV decomposes the Essential Matrix to recover the relative rotation and translation between two camera views. The function signature is:

```
int recoverPose(InputArray E,
 InputArray points1, InputArray points2,
 InputArray cameraMatrix,
 OutputArray R, OutputArray t,
 InputOutputArray mask = noArray());
```

- **E:** The input Essential Matrix.
- **points1, points2:** Arrays of corresponding points between the two images.
- **cameraMatrix:** The intrinsic camera matrix.
- **R:** Output rotation matrix.
- **t:** Output translation vector.
- **mask:** Input/output mask for inliers.

The function works as follows:

1) Input Validation: The function checks the validity of the input parameters, including the Essential Matrix and the corresponding points.
2) Decomposition: The Essential Matrix $\mathbf{E}$ is decomposed into the rotation matrix $\mathbf{R}$ and the translation vector $\mathbf{t}$. This involves singular value decomposition (SVD) of $\mathbf{E}$ to obtain the possible solutions for $\mathbf{R}$ and $\mathbf{t}$.
3) Chirality Check: The function performs a chirality check to ensure that the reconstructed points are in front of both cameras. This step helps in selecting the correct solution from the possible decompositions.
4) Output: The function returns the number of inliers that pass the chirality check and outputs the rotation matrix $\mathbf{R}$ and the translation vector $\mathbf{t}$ [18].

The 'recoverPose()' function is crucial for determining the relative pose between two camera views, which is essential for applications such as visual odometry and 3D reconstruction.

D. Custom Implementation and Discussion

After creating the matchings, the camera intrinsics, matches, and keypoints of the two images are loaded into the function. The camera intrinsics are necessary to describe the internal geometry and optical properties of the camera [19]. The following functions are used to reconstruct the pose:

```
#Iterating over all keypoints found in the
 images
pts1 = np.float32([keypoints1[m.queryIdx].pt
 for m in matches])
pts2 = np.float32([keypoints2[m.trainIdx].pt
 for m in matches])

# Compute the Essential Matrix
E, mask = cv.findEssentialMat(pts1, pts2,
 camera_matrix)
_, R, t, mask = cv.recoverPose(E, pts1,
 pts2, camera_matrix)
```

The calculated changes in pose between the considered image pair, consisting of the rotation matrix $\mathbf{R}$ in a 3×3-format, as well as `change_x`, `change_y`, `change_z`. Later on the $\mathbf{R}$-matrix is converted to **euler** angle. These changes are then processed and stored in an array, which is later used for interactive graphical representation.

E. Framework Conditions and Discussion

The framework conditions for this study are defined by the use of specific methods for feature detection and matching, as well as the constraints imposed by the synthetic test data generated from Blender. The following points elaborate on these conditions and the rationale behind them:

a) Use of Synthetic Test Data: The primary reason for using synthetic test data generated from Blender is the availability of absolute positions for each keyframe. This allows for a direct comparison between the estimated poses and the ground truth, providing a clear measure of the accuracy of the pose estimation algorithms. Attempts were made to find additional test data, such as the KITTI dataset. However, it was quickly discovered that the implementation could not process these images. This is likely due to the fact that the KITTI images are point clouds stored as PNG files, which are not continuous 1. One potential solution to this problem would be to make the images continuous by removing all black spaces between the data points, so that all recorded data points are pixel-to-pixel. However, it is not possible to validate whether each image in the sequence has the measurement points at the same pixel (i.e., whether it is like a raster mask that consistently lies at exactly the same position on all images in the sequence). Additionally, there are no absolute positions available for these image sequences, which prevents a comparison between the estimation and the absolute position.

Fig. 1. Example from depth_data_velodyne [11]

b) Rendered Images: In addition to depth images, attempts were made to use rendered images. However, there appears to be an issue with the file type, which prevents feature detection from being performed. As a result, the program only initializes and then terminates.

c) Flexion Images: Another test run was conducted using Flexion Images. These Flexion Images are derived from the original depth images and are merely a further processed version of them. With these images, the entire program ran successfully, meaning that both features and matches were found, and these were then used for pose reconstruction.

d) Hardware and Software Constraints: The evaluation of the benchmarks was performed on a 12-core (i7-8750H) laptop with 64GB RAM and the integrated GPU (iGPU). The calculation of the pose, features, and matchings was done solely on the CPU. No GPU acceleration was used. It is assumed that an implementation utilizing the GPU would achieve significantly higher performance.

e) Limitations of OpenCV Implementation: The non-proprietary implementation of OpenCV was used, which limits the selection of feature detection algorithms. The methods used for feature detection and matching were SIFT, ORB, and AKAZE, as these are well-supported by OpenCV and provide a good balance between accuracy and computational efficiency.

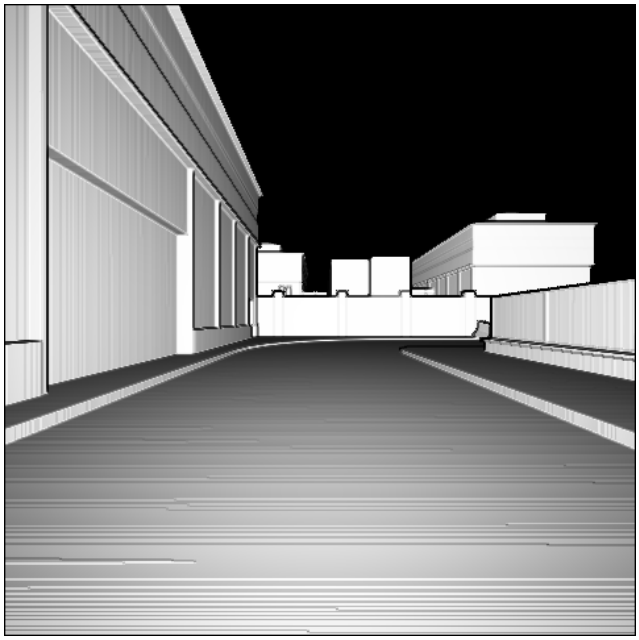

Fig. 2. Example of an Flexion Image

f) Evaluation Metrics: The evaluation metrics for the benchmarks include the time taken for pose reconstruction, the cumulative deviation from the original path in the X, Y, and Z directions, and the minimum, maximum, and average errors in the three directions. These metrics provide a comprehensive assessment of the performance of the pose estimation algorithms.

III. Benchmark Description

The benchmark involves evaluating the performance of the SIFT, ORB, and AKAZE algorithms in terms of pose estimation accuracy and computational efficiency. The depth images as well as the flexion images used in this study were generated from a Blender map using the ”blainder” plugin and a self-implemented tool for converting depth images into flexion images. The evaluation metrics include the time taken for pose reconstruction/estimation per image and for the complete data set, the cumulative deviation from the original path in the X, Y, and Z directions, and the minimum, maximum, and average errors in the three directions.

The tests were conducted with a fully charged battery and the computer connected to the power supply. The power mode was set to ”Performance” to ensure maximum possible performance. This is important because some laptops tend to throttle CPU performance when running on battery power. Additionally, all other applications were closed during the tests to allow the feature detection and matching program with pose estimation to utilize all available resources.

The tests were performed using both the dataset of depth images and the dataset of flexion images. The FLANN algorithm was used for matching the descriptors in each test. For each test run, both the descriptor and detector algorithms were identical (i.e., SIFT, ORB, and AKAZE). After each test,

a pause of approximately 5 minutes was taken to allow the device to cool down, preventing potential thermal throttling at the start of a new test. A total of six tests were conducted: three for depth images with SIFT, AKAZE, and ORB, and three for flexion images with SIFT, AKAZE, and ORB.

For all six tests, the output/writing of the descriptors and keypoints as text files and the generation of matchings as side-by-side images with corresponding markings using colored lines between the keypoints of the two images were disabled.

The detailed procedure for conducting the tests is as follows:

1) **Preparation:** Ensure the laptop is fully charged and connected to the power supply. Set the power mode to "Performance" and close all other applications.
2) **Dataset Selection:** Select the dataset to be used (either depth images or flexion images).
3) **Algorithm Selection:** Choose the feature detection and matching algorithm (SIFT, ORB, or AKAZE).
4) **FLANN Matching:** Use the FLANN algorithm to match the descriptors. Note that FLANN is used for matching descriptors, not keypoints.
5) **Pose Estimation:** Run the pose estimation algorithm using the selected dataset and feature detection method. Record the time taken for pose reconstruction/estimation per image and for the complete dataset.
6) **Cooling Period:** After each test, allow the laptop to cool down for approximately 5 minutes to prevent thermal throttling.
7) **Repeat:** Repeat the above steps for each combination of dataset and feature detection algorithm.

The evaluation metrics for the benchmarks include:

- Time taken for pose reconstruction/estimation per image and for the complete dataset.
- Cumulative deviation from the original path in the X, Y, and Z directions.
- Minimum, maximum, and average errors in the three directions.

A. Benchmark Results

a) General Expectations: Under the assumption that the implementation of the pose estimation is correct, similar results in terms of the quality of pose estimation should be expected across all test runs with the corresponding algorithms and input data. Ideally, the absolute path should deviate little to none from the estimated path. Consequently, the representation of the difference between the absolute path and the reconstructed path should approximate the shape of a sphere. This would indicate that the errors/deviations at each data point are within a certain order of magnitude. Thus, the total deviation should be within a certain order of magnitude for all three directions. If a different shape emerges, it suggests that errors in a particular direction or rotational axis are larger.

Under this assumption, the cumulative error in all directions should grow, but only because it consists of the absolute values of the deviations between the estimation and the absolute position. Both the minimum, maximum, and average deviations of each axis should be within a similar order of magnitude. Additionally, these deviations for each axis should be within a similar order of magnitude. Otherwise, it could indicate that the reconstruction/pose estimation is not as reliable/stable in certain directions as in others.

The actual results, unfortunately, do not meet the expectations.

B. Results

a) SIFT: The SIFT algorithm showed the following results:

- Fastest Processing Time: Depth Images
- Slowest Processing Time: Flexion Images
- Lowest Cumulative Error: Depth Images
- Highest Cumulative Error: Flexion Images

The SIFT algorithm performed better with depth images compared to flexion images. The processing times were faster, and the cumulative errors were lower for depth images. This suggests that SIFT is in this case more efficient and accurate when working with depth images.

b) AKAZE: The AKAZE algorithm showed the following results:

- Fastest Processing Time: Depth Images
- Slowest Processing Time: Flexion Images
- Lowest Cumulative Error: Depth Images
- Highest Cumulative Error: Flexion Images

Similar to SIFT, the AKAZE algorithm performed better with depth images. The processing times were faster, and the cumulative errors were lower for depth images. This indicates that AKAZE is in this case more efficient and accurate with depth images.

c) ORB: The ORB algorithm showed the following results:

- Fastest Processing Time: Depth Images
- Slowest Processing Time: Flexion Images
- Lowest Cumulative Error: Depth Images
- Highest Cumulative Error: Flexion Images

The ORB algorithm also performed better with depth images. The processing times were the fastest among the three algorithms, and the cumulative errors were lower for depth images. This suggests that ORB is highly efficient and more accurate with depth images.

1) Overall Comparison: When comparing all three algorithms and both datasets, the following observations can be made:

- Fastest Algorithm: ORB with depth images
- Slowest Algorithm: SIFT with flexion images
- Lowest Cumulative Error: SIFT with depth images
- Highest Cumulative Error: AKAZE with flexion images
- Highest Maximum Error: AKAZE with flexion images

There is a significant difference between the use of flexion images and depth images. All three algorithms performed better with depth images in terms of processing time and cumulative error. The flexion images resulted in higher cumulative errors and longer processing times, indicating that depth

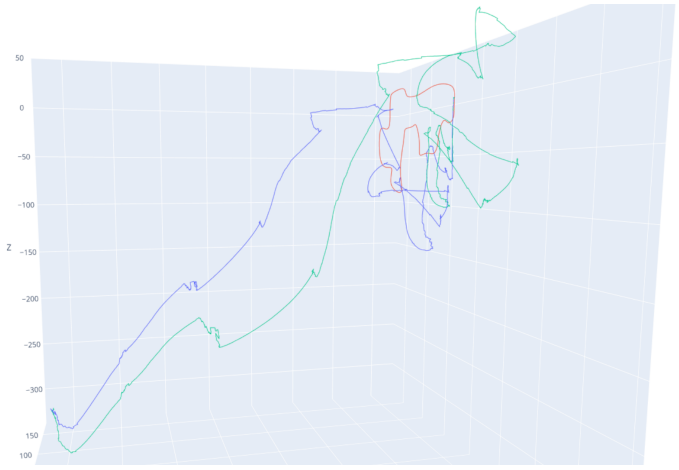

Fig. 3. Interactive Plot of Pose Estimation with SIFT and Flexion Images, red: Ground Truth, blue: Pose Estimation, green: Difference

Image processing time (FPS)	SIFT	AKAZE	ORB
depth images:			
min	2.13	2.67	2.63
max	5.57	7.10	12.76
avg	3.87	4.62	7.68
flexion images:			
min	1.80	2.31	2.49
max	4.08	4.90	8.30
avg	2.51	3.29	5.05

TABLE I
TABLE OF PROCESSING TIMES PER ALGORITHM AND DATA SET

images are more suitable for pose estimation tasks, at least with this implementation of the feature detection, matching and pose estimation.

The exact results in terms of processing time can be found in the table III-B1.

Upon examining the magnitude of the errors, it is evident that the cumulative errors in all scenarios are substantial. The interactive graphics of the test runs suggest that these errors might be due to incorrect orientation alignment. During the implementation, all possible combinations of rotations and reference axes were tested, including a general offset of the rotation. In none of the tested configurations could the pose estimation be aligned with the absolute path. This indicates an implementation error. Regardless, if the quality of the pose reconstruction were adequate, there should at least be a visual similarity. This is not the case. In some test runs, ”knots” appear, indicating that the algorithm detected abrupt changes in direction at these points. However, when examining the matchings, no significant errors are apparent. Although matchings are occasionally incorrect, the majority of matchings in each image pair are correct within a range that would result in small deviations in pose estimation, but not to the extent observed. Notably, in all test scenarios, the flexion images appear more stable and contain fewer of these ”knots” when viewed in the interactive plots.

In summary, the comparison has limited significance. The performance is insufficient for real-time applications. It is likely due to the Python implementation and the lack of graphics acceleration. The interactive plots also indicate that flexion images result in more stable pose estimations than pure depth images. To obtain more meaningful results, the pose reconstruction function needs to be revised. It is likely that there was some form of misalignment of the axes and rotations.

It should also be noted that using flexion images requires preprocessing the respective depth image, which consumes computation time. With the implementation used here, real-time capability is far from being achieved [20].

All results, test data, and the tool can be found in the repository [19].

IV. CONCLUSION

In this study, various test runs were conducted using different datasets. Unfortunately, the results did not meet expectations, which appears to be solely due to the implementation of the pose estimation. It can be assumed that the cause is the swapping of data for the respective axes and rotations. This issue could not be resolved by the time the work was completed. Consequently, the results of the feature-based pose estimation do not show any resemblance to the ground truth. However, this seems to be merely a methodological problem of the implementation.

Regardless, it is evident that there are differences between the types of datasets used. The flexion images show a significantly more stable path compared to the normal depth images. Therefore, it can be assumed that these are advantageous for more accurate pose estimation, but only under the condition that a real-time capable implementation for the calculation of these flexion images is used, as in [12]. In all cases, real-time capability cannot be achieved, as all tested algorithms with the specific implementation only reached frame rates in the low double-digit range.

Future work could focus on revising the existing framework to ensure that pose estimation or orientation works reliably. For practical applications, it would also be essential to improve the performance of the implementation. This could be achieved by using graphics acceleration or by using hardware-near programming languages such as C++. If the approach of flexion images is further pursued, it would also be necessary to integrate this calculation into the process chain.

REFERENCES

[1] C. Sminchisescu, L. Bo, C. Ionescu, and A. Kanaujia, *Feature-Based Pose Estimation*. London: Springer London, 2011, pp. 225–251. [Online]. Available: https://doi.org/10.1007/978-0-85729-997-0_12

[2] S. Dubey and M. Dixit, “A comprehensive survey on human pose estimation approaches,” *Multimedia Systems*, vol. 29, no. 1, pp. 167–195, 2023. [Online]. Available: https://doi.org/10.1007/s00530-022-00980-0

[3] H. Chen, R. Feng, S. Wu, H. Xu, F. Zhou, and Z. Liu, “2d human pose estimation: A survey,” 2022. [Online]. Available: https://arxiv.org/abs/2204.07370

[4] Y. Chen, Z. Li, Q. Li, and M. Zhang, “Pose estimation algorithm based on point pair features using pointnet + +,” *Complex & Intelligent Systems*, 2024. [Online]. Available: https://doi.org/10.1007/s40747-024-01508-x

[5] J. Liu, W. Sun, H. Yang, Z. Zeng, C. Liu, J. Zheng, X. Liu, H. Rahmani, N. Sebe, and A. Mian, "Deep learning-based object pose estimation: A comprehensive survey," 2024. [Online]. Available: https://arxiv.org/abs/2405.07801
[6] Y. Shavit and R. Ferens, "Introduction to camera pose estimation with deep learning," 2019. [Online]. Available: https://arxiv.org/abs/1907.05272
[7] P. D. E. Weitz, "Sift - scale-invariant feature transform," http://weitz.de/, 2016. [Online]. Available: http://weitz.de/sift/index.html
[8] unknown, "Akaze local features matching," https://docs.opencv.org/. [Online]. Available: https://docs.opencv.org/3.4/db/d70/tutorial_akaze_matching.html
[9] G. Levi, "A tutorial on binary descriptors – part 3 – the orb descriptor," https://gilscvblog.com/, 2013. [Online]. Available: https://gilscvblog.com/2013/10/04/a-tutorial-on-binary-descriptors-part-3-the-orb-descriptor/
[10] unkown, "Understanding features," https://docs.opencv.org/, 2024. [Online]. Available: https://docs.opencv.org/3.4/df/d54/tutorial_py_features_meaning.html
[11] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun, "Vision meets robotics: The kitti dataset," *International Journal of Robotics Research (IJRR)*, 2013.
[12] J. Toth, "Post-processing of depth images and laser scan data for feature-based pose estimation," pp. 0–97, Jun. 2020. [Online]. Available: https://github.com/JonasToth/depth-conversions/blob/master/docs/master-thesis-jonas-toth-compressed.pdf
[13] A. Raibolt, "Feature detection and matching," github.com, 2022. [Online]. Available: https://github.com/whoisraibolt/Feature-Detection-and-Matching
[14] O. team, "Opencv documentation," https://docs.opencv.org/, 2024. [Online]. Available: https://docs.opencv.org/
[15] Baeldung, "Difference between fundamental matrix and essential matrix," https://www.baeldung.com/, 2024. [Online]. Available: https://www.baeldung.com/cs/fundamental-matrix-vs-essential-matrix
[16] O. team, "Opencv: Epipolar geometry," https://docs.opencv.org/, 2024. [Online]. Available: https://docs.opencv.org/master/da/de9/tutorial_py_epipolar_geometry.html
[17] unknown, "Cv2.findessentialmat method (inputarray, inputarray, inputarray ..." 2024. [Online]. Available: https://shimat.github.io/opencvsharp_docs/html/12c25bd5-c6f9-9d8b-1ee3-68b61fc78fb9.htm
[18] ——, "recover_pose in opencv::calib3d - rust - docs.rs," https://docs.rs/, 2024. [Online]. Available: https://docs.rs/opencv/latest/opencv/calib3d/fn.recover_pose.html
[19] J. Fleischer, "seminar_sose_2024," https://www.github.com, 2024. [Online]. Available: https://github.com/halbersacknuesse/seminar_sose_2024.git
[20] E. L. M. O. Jonas Fleischer, Nico Heinrich, "Schriftliche ausarbeitung zum seminar virtuelle realität," pp. 1–58, Mar. 2023.