The probabilistic approach to automated visual navigation in closed spaces

Olga Artemenko[1,†], Oleksandr Dorenskyi[2,*,†], Taras Terletskyi[3,†], Oleh Kaidyk[3,†] and Andriy Kramar[1,†]

[1] *Private Higher Educational Institution "Bukovinian university", Ch. Darvina Street 2a, 58000 Chernivtsi, Ukraine*
[2] *Central Ukrainian National Technical University, Universytetskyi Avenue 8, 25006 Kropyvnytskyi, Ukraine*
[3] *Lutsk National Technical University, Lvivska Street 75, 43018 Lutsk, Ukraine*

Abstract

The paper reviews current approaches to automated navigation tasks, such as localization and mapping, and proposes a different approach based on probability density approximation. The proposed system uses feature extraction to reduce computation complexity. Descriptors, are then used to match mapped features to a current view, and localization uses iterative pose estimation based on gradient descent. Current system limitations and weaknesses are also shown.

Keywords

visual odometry, mapping, localization, feature extraction, probability density approximation.

1. Introduction

Robotic systems are increasingly used in confined spaces where traditional navigation methods, including GPS, are inaccessible or not accurate enough. The use of infrastructure solutions (beacons, reflectors, markers) requires high costs and limits flexibility. Instead, visual navigation based on computer vision and artificial intelligence algorithms can provide autonomy, accuracy, and adaptability to environmental changes. This makes the development of an automatic visual navigation system an urgent task aimed at improving the efficiency and safety of robotic complexes. In this research, the authors aim to apply a probability distribution approximation approach to localization and mapping problems, review the benefits and challenges this path offers and describe some elements of the proof-of-concept algorithm they are currently working on.

2. Current State of Research in the Field

In modern literature, in the context of visual navigation, the following tasks are considered: localization, mapping, route planning and traffic control (traffic control and obstacle detection) [1]. Localization is the determination of the exact location of an object in a given coordinate system.

There are several areas of computer vision research dealing with localization issues: Visual Odometry (VO), Visual Position Recognition (VPR), and Visual Simultaneous Localization and Mapping (VSLAM).

Visual odometry (VO) solves this problem directly by iteratively calculating the displacement of the robot both with the help of inertial sensors, odometers, and through the analysis of image changes from cameras, in particular, calculating the optical flux.

[*] *AIT&AIS'2025: International Scientific Workshop on Applied Information Technologies and Artificial Intelligence Systems, December 18–19 2025, Chernivtsi, Ukraine*
[1*] Corresponding author.
[†] These authors contributed equally.

olga.hapon@gmail.com (O. Artemenko); dorenskyiop@kntu.kr.ua (O. Dorenskyi); t.terletskyi@lntu.edu.ua (T. Terletskyi); o.kaidyk@lntu.edu.ua (O. Kaidyk); andriykramar@gmail.com (A. Kramar)
0000-0002-4057-1217 (O. Artemenko); 0000-0002-7625-9022 (O. Dorenskyi); 0000-0002-4114-0734 (T. Terletskyi); 0000-0002-3620-270X (O. Kaidyk); 0009-0006-7371-4508 (A. Kramar)

Optical flux in the context of visual odometry is a vector field that describes how pixels in an image are shifted between two consecutive camera frames. In other words, it is an estimate of the apparent movement of the points of the scene relative to the camera.

Usually, it is calculated either on a continuous wide area of the image (dense optical flow) or on individual features/key points (sparse optical flow).

To ensure sufficient accuracy, the calculation of displacement iterations must occur quite often – given limited computational resources, this imposes significant limitations on the algorithms that can be used to calculate the optical flux.

Another problem is that optical flux is not the true motion of objects — it is a projection of three-dimensional motion onto the plane of an image. Therefore, algorithms based on dense optical flux can lose accuracy if there are a significant number of repetitive patterns in the image.

Known methods for calculating dense flow:

1. Horn–Schunck (HS) [2].
2. Lucas–Kanade Dense (Pyramidal LK / Farnebäck) [3].
3. Total Variation L1 (TV-L1) Optical Flow is an extension of the HS model with pyramidal scale and variational solution [4].
4. Brox Optical Flow (Variational, multilevel) [5].

Table 1
Comparison of Algorithms for Calculating Dense Optical Flux

Method	Accuracy	Performance	Application in VO
Horn–Schunck	★★☆☆☆ (low/medium)	★★★★☆ (fast)	Limited, basic scenes
Farnebäck	★★★☆☆	★★★★☆	Very common (baseline VO)
TV-L1	★★★★☆	★★☆☆☆	Stable VO, indoor/outdoor
Brox	★★★★★	★☆☆☆☆	Used for offline VO and ground truth
Pyramidal Farnebäck	★★★★☆	★★★☆☆	Real VO Pipelines
HS / LK Hybrid	★★★☆☆	★★★★☆	Embedded VO, micro drones

On the other hand, algorithms based on feature selection are used in VPR and VSLAM: instead of analyzing the entire frame, the algorithm finds noticeable local fragments (features) — points, corners, texture elements, and contours. A descriptor is built for each such point.

A descriptor is a compact numerical representation of a local view that can be compared between frames or with a memory base.

This allows:

1. Recognize terrain (VPR).
2. Evaluate camera movement and orientation (VSLAM).
3. Build an environment map (VSLAM).

Mapping in VSLAM is the process of building and gradually updating a spatial model of the environment based on images (video stream) and other sensory data.

In VSLAM, this includes the following key aspects:

1. Select and save landmarks (features/landmarks): From camera frames, stable points, objects, or characteristics of the scene are identified, which can be re-recognized during further movement.
2. Map formation in the form of 3D points or their structure: based on triangulation and assessment of the camera pose, a map of the environment (for example, a sparse point cloud) is created.
3. Constant map update: as the work progresses, the map is expanded, clarified, and corrected (via loop closure).
4. Cooperation with localization: mapping is closely related to the assessment of one's own posture. The map is used for better localization, and accurate localization allows you to refine the map.

Loop closure is the process of detecting that a current observation corresponds to an already known location on the map, followed by optimizing the posture and map to eliminate accumulated drift.

Simply put, mapping in VSLAM is the construction of an internal spatial representation of the environment (in the form of landmarks, 3D points, or keyframes) that allows the robot to navigate and localize.

In classical VPR (Visual Place Recognition), the concept of mapping in the usual sense for VSLAM is almost never used, but there is a concept that is similar in meaning.

VPR focuses on recognizing already known places from an image, rather than building a complete map of space. That is, the system receives the current frame and compares it with a database of reference images or features to determine "where I have already been". It is not required to build a 3D map or spatial model.

MMost VPR algorithms use a database of images or descriptors, which can be thought of as a passive observation map. It contains:

1. Global or local visual signs.
2. Coordinates or place indexes (if available).
3. Offline mapping.

In terms of computational resources, feature-highlighting algorithms have certain advantages compared to those that use dense optical flux, but they also lose accuracy in the case of large, uniform areas in the image.

3. Existing Algorithms for Determining the Trajectory of Robots Based on Images

Today, there are a large number of algorithms for determining the trajectory of robots adapted for various functions, designs, and conditions of use of these systems. In particular, for indoor spaces, the following can be used:

1. *PTAM (Parallel Tracking and Mapping)* is one of the first successful real-time visual SLAM algorithms developed by Hern Klaus and Andrew Davison (2007) [6].
2. *ORB-SLAM* – supports monocular, stereo, and RGB-D cameras, as well as inertial (IMU) mode. Includes multi-map mode, the ability to merge maps, and strong initialization with the IMU [7].

3. *VINS-Mono / VINS-Fusion* is a monocular visual-inertial (camera + IMU) algorithm with an optimization "sliding window" scheme that includes IMU integration and an extension that allows multi-touch integration (mono + IMU, stereo + IMU, even with GPS) and map merging [8, 17].
4. *RTAB-Map (RGB-D)* – real-time mapping + localization with a focus on RGB-D cameras (depth + image). Works as a topological map with a "workspace", uses flash tables (bag-of-words) for loop closure and map extension [9].
5. *DROID-SLAM* is a hybrid architecture with deep learning that combines classical optimization methods with neural networks. Makes recurrent iterative position updates. Works with monocular, stereo, or RGB-D video [10].
6. *DSO (Direct Sparse Odometry)* direct (no explicit feature highlighting) approach, works well in conditions with good texture, can be more flexible in scenes with few features, but is sensitive to lighting and exposure changes [11].
7. *DTAM (Dense Tracking and Mapping)* is one of the fundamental dense SLAM algorithms that has become the forerunner of many modern methods of dense 3D reconstruction and tracking.
8. *LSD-SLAM (Large-Scale Direct Monocular SLAM)* is a direct method that builds semi-dense depth maps as the camera moves. Uses filtering of pixels that have a sufficient brightness change with a small parallax change. Allows map construction even in conditions of weak signs, but can be unstable in aggressive traffic or poor lighting [12].
9. *MSCKF* is one of the classic algorithms that works on the basis of an advanced Kalman filter using a camera + IMU without building a dense map.
10. *OKVIS (Open Keyframe-based Visual-Inertial SLAM)* is an optimization (smoothing) system that combines visual measurements and IMU, uses keyframes, and minimizes projection errors [13, 18].
11. *ROVIO (Robust Visual-Inertial Odometry)* is a filtering approach (advanced Kalman filter and tracking of both image fragments and 3D points) for a visual-inertial system [16].
12. *SVO (Semi-Direct Visual Odometry)* is a visual odometry algorithm that occupies an intermediate place between feature-based and direct methods [14].
13. *DeepVO* is one of the first end-to-end visual odometry algorithms to use deep learning to evaluate camera movement without the traditional steps (feature detection, comparison, filtering, etc.) [15].

In the context of the tasks of this work, localization algorithms can be classified according to the possibility of use in closed spaces (indoor), the number of cameras used, the use of additional sensors (IMU), and approaches to image processing (feature highlighting, optical flux calculation, neural networks) (Fig. 1).

Comparison of algorithms in terms of requirements for hardware and computing resources, and general areas of application, is given in Table 2.

Table 2
Comparison of Localization Algorithms

Algorithm	Advantages	Restriction	Typical applications
ORB-SLAM3	Works with mono/stereo/RGB-D/IMU, loop closure, multi-map. High accuracy and stability.	High requirements for computing resources. Limited work in monotonous environments.	Drones, AR/VR, mobile robotics, indoor navigation.
VINS-Mono / VINS-Fusion	Accurate VIO with IMU Autocalibration and loop closure. Fusion with GPS, stereo, multiple sensors.	Without IMU, it works to a limited extent.	Indoor robots, drones, autonomous systems, mobile devices.

RTAB-Map (RGB-D)	Support for large maps. Loop closure + relocalization.	Requires depth (RGB-D). Heavier than VIO or mono SLAM.	Service robotics, warehouses, indoor mapping.
DROID-SLAM	High fidelity on difficult scenesMono/stereo/RGB-D support. Track loss resistance.	GPU is desirable for work. The main core is deep learning.	Autonomous Robots, UAV, Complex Scenes, Academic Applications.
DSO / LSD-SLAM	No identification of signs Works with weak signs. Low dependence on detectors.	Sensitivity to light. No IMU unstable. No full loop closure.	Lightweight odometry. AR experiments, academic applications.
OKVIS / ROVIO	Reliable VIO with IMUOKVIS – optimization. ROVIO is a lightweight EKF.	Without IMUs do not work. No global maps or weak loop closure.	Drones, ground robots, real-time, energy-limited systems.

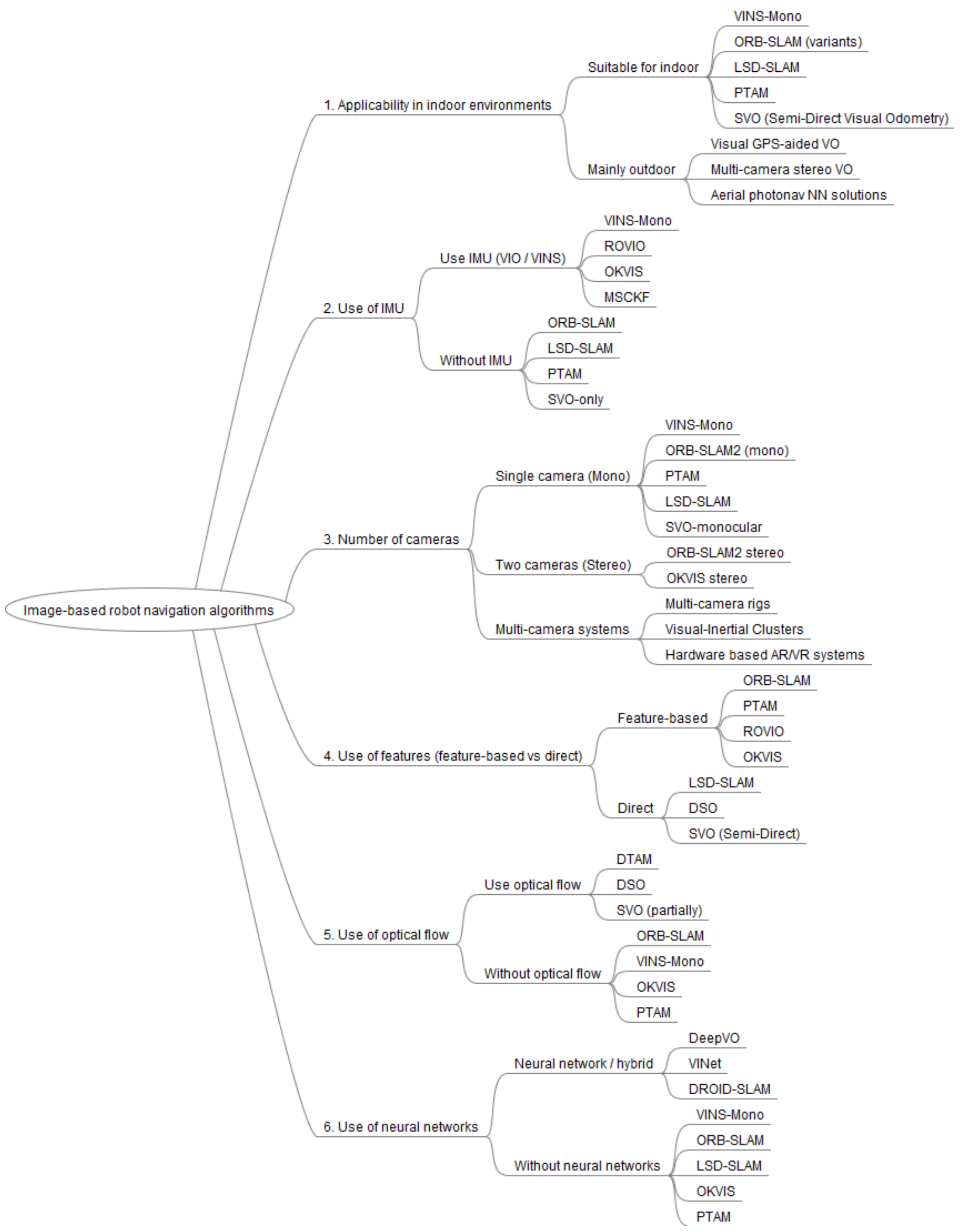

Figure 1: Classification of localization algorithms.

4. Probabilistic Approach to Mapping and Localization

In this study, the authors consider the first two tasks of navigation: route mapping (using training video recording of movement along the route and accelerometric and odometrical data), and robot localization (based on video from the robot's camera, without auxiliary data).

Localization of a robot is the determination of the coordinates of its position (x, y, z) in a predetermined general coordinate system, as well as the determination of the coordinates of the heading vector of its camera in this coordinate system (hx, hy, hz) (Figure 2).

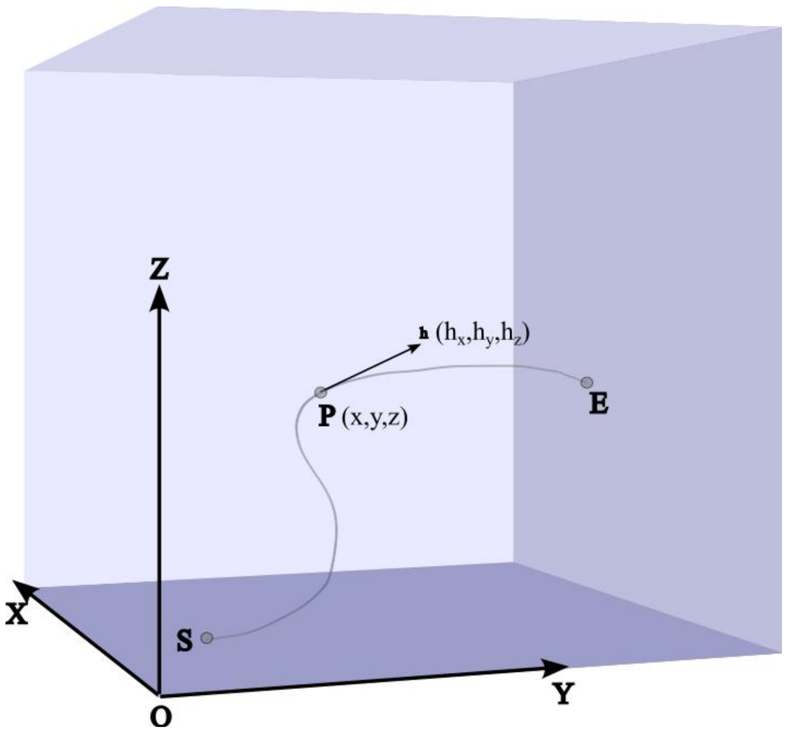

Figure 2: General coordinate system, current position P and heading vector h.

It is important to mention restrictions pertaining to this way of pose description: the robot's local coordinate system rotations are limited in such a way that no axial rotations around are possible, i.e., we have stable "up" and "down" directions.

The mapping process can be presented as highlighting sets of certain features (visible landmarks) on the video and approximation of the distribution of probability density ρ of the robot's presence at a point along the vector $\{(C_V, x, y, z, h_X, h_Y, h_Z)\}$ in the space of input vectors, where C_v is the set of coordinates of the features selected in the frame, x, y, z are the coordinates of the robot, h_x, h_y, h_z, are the coordinates of the camera's guide vector.

Knowing the probability distribution ρ, it is possible to solve the localization problem by finding the position x_o, y_0, z_o, h_{Xo}, h_{Yo}, h_{Yo} of the maximum ρ for a given set of C_v:

$$E\rho(C_V, x_0, y_0, z_0, h_{X0}, h_{Y0}, h_{Z0}) = max_{X,H}(\rho(C_V, x, y, z, h_X, h_Y, h_Z)), \tag{1}$$

In order to simplify calculations, ρ can be represented as the product of probability density distribution functions for individual features ρ_i:

$$\rho(C_V, x, y, z, h_X, h_Y, h_Z) = \prod_{f=1}^{V} \rho_f(u_f, v_f, x, y, z, h_X, h_Y, h_Z). \tag{2}$$

Here u_f, v_f are the screen coordinates of the f-th feature.

Thus, the construction of a navigation model can be viewed as approximation of distributions ρ_i.

The use of (2) and the gradient descent (ascent) method for estimation (1), in our opinion, can provide a flexible iterative process of refining the position of the robot by step-by-step consideration of features.

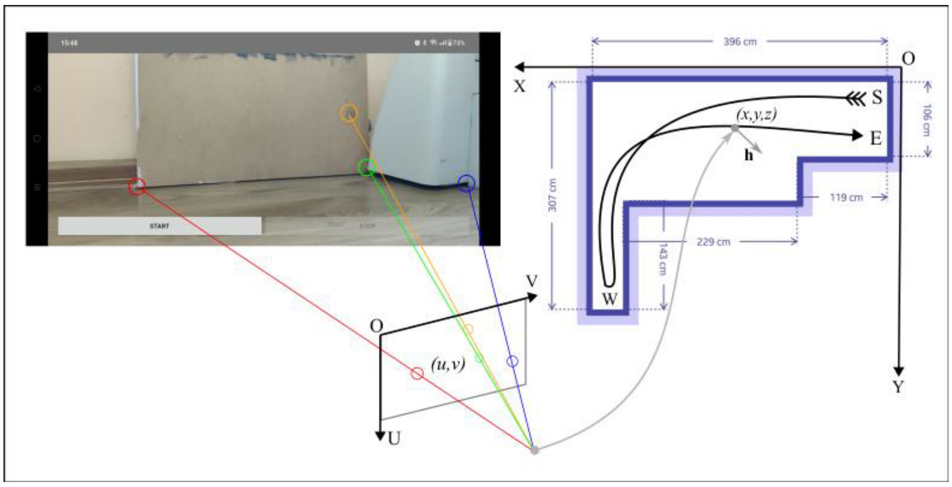

Figure 3: Determination of the position and orientation of the camera by the position of features on video frames.

A distinctive feature of this approach is that we do not build a map of the environment in the sense of VSLAM, but create a database of features with approximated densities of probabilities of them being in certain point of multidimensional space S of position (x, y, z), heading vector of the camera h_x, h_y, h_z, and screen coordinates (u, v): $s = (u, v, x, y, z, h_x, h_y, h_z) \in S$.

At the mapping phase for each feature f we collect a set of observations: $S_f = \{(u_i, v_i, x_i, y_i, z_i, h_{xi}, h_{yi}, h_{zi})\}$. Before starting approximation process it is mandatory to consider rotational symmetry: given different camera orientation h same feature f can be viewed at different screen coordinates (u, v) (see Figure 4).

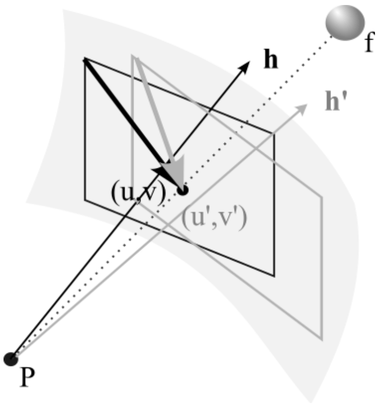

Figure 4: Rotational symmetry considerations.

To account for this fact, we inject in our set S_f additional points for set of pre-determined heading vectors h_{as} for which the calculated (u, v) fits within viewport.

Let's assume that approximation ρ_f is a hypersurface in (ρ, S) space defined by equation:

$$\rho_f(u,v,x,y,z,h_x,h_y,h_z) =$$

$$= \sum_{p_u=0}^{P} \sum_{p_v=0}^{P} \dots \sum_{p_{hx}=0}^{P} \dots \sum_{p_{hz}=0}^{P} a_{p_u, p_v, p_x, \dots, p_{hx}, \dots, p_{hz}} u^{p_u} v^{p_v} x^{p_x} \dots h_x^{p_{hz}} h_z^{p_{hz}} . \qquad (3)$$

Here $a_{p_u, p_v, p_x, \dots, p_{hx}, \dots, p_{hz}}$ – coefficients that must be determined, P – maximal power we consider.

One limitation related to representation (3) that we already see is that feature have to be "local enough" – if there are many view areas producing similar features their corresponding approximations become less useful for pose estimation.

Having probability density in form (3) allows to compute partial derivatives easily, e.g.:

$$\frac{\partial \rho_f}{\partial u} = \sum_{p_u=0}^{P} \sum_{p_v=0}^{P} \dots \sum_{p_{hx}=0}^{P} \dots \sum_{p_{hz}=0}^{P} a_{p_u, p_v, p_x, \dots, p_{hx}, \dots, p_{hz}} u^{p_u - 1} v^{p_v} x^{p_x} \dots h_x^{p_{hz}} h_z^{p_{hz}}. \tag{4}$$

The loss $L_{f,\rho}$ for candidate distribution approximation ρ_f can be expressed as:

$$L_{f,p} = \sum_{i}^{N_f} \left| N_f \rho_f(s_i) w - n(s_i) \right|^2. \tag{5}$$

Here N_f – number of data points in the set, $n_f(s)$ – number of observations of feature f in the small volume w around vector s. The goal of approximation algorithm is to find a set of coefficients $a_{p_u, p_v, p_x, \dots, p_{hx}, \dots, p_{hz}}$ such that $L_{f,\rho}$ is minimal.

Although the mapping phase is computationally intensive, ideally, it can be performed once for a given closed space. The result is a file with a list of stable features with their corresponding approximation coefficients.

The localization problem is solved by detecting features in a video frame and looking up their similarities in the map file. For each feature match the corresponding probability coefficients then used to compute estimated probabilities via (2) and their gradients using (5) that are used iteratively to adjust pose estimation.

5. Overview of Implemented System

To implement the data processing algorithm and navigation algorithm, OS Ubuntu v.24.04, the Python runtime environment v.3.12, and the interactive shell JuPyter are used. The PyPlot and ParaView tools are used for data visualization.

As part of development and testing a data collection and mapping subsystem was developed. To gather video and IMU information a smartphone app was created that records video and IMU data.

The Android Studio environment was used to develop the application to collect training data. The application supports Android OS version 13 and higher.

A screenshot of the application is shown in Figure 5.

Requirements for the application:

1. Ability to record video.
2. Ability to record accelerometer and gyroscope readings of a smartphone with a frequency of at least 25 measurements per second and store them together with the timestamp in CSV format (see Table 3).

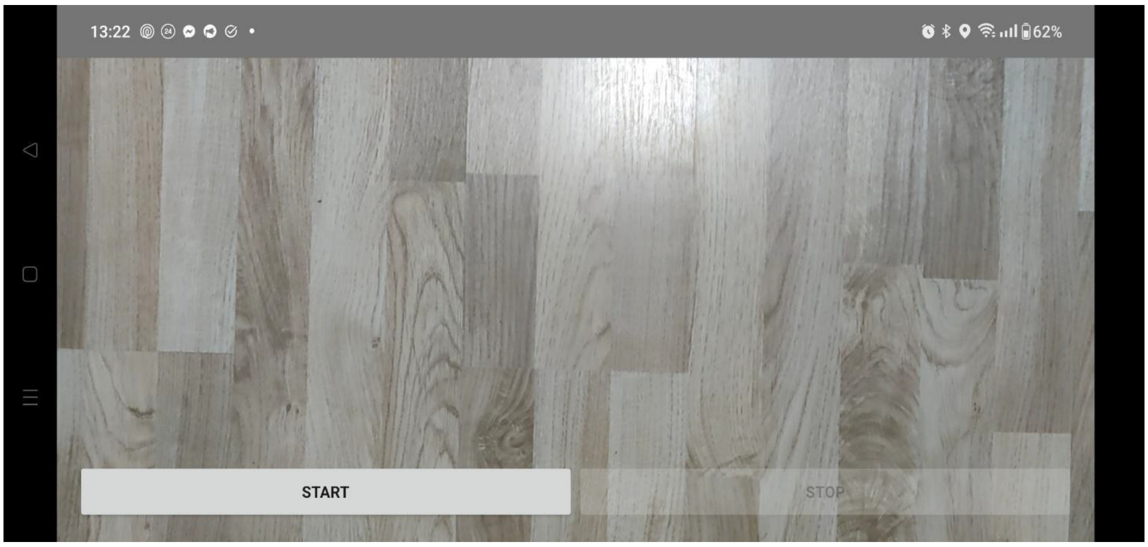

Figure 5: Application interface for data collection.

Table 3
IMU Data Saving Format

The Time Mark, N	Linear acceleration, m/s2			Angular acceleration, rad/s2		
	X	Y	Z	X	Y	Z

An algorithm was also developed for intermediate processing of training data. Stages of the algorithm:

1. Select lighting-resistant feature sets from video frames.
2. Generation of a unique handle for each feature.
3. Define the timestamp of the frame.
4. Calculation of camera position and orientation for a given frame based on timestamps.
5. Saving data in CSV format.

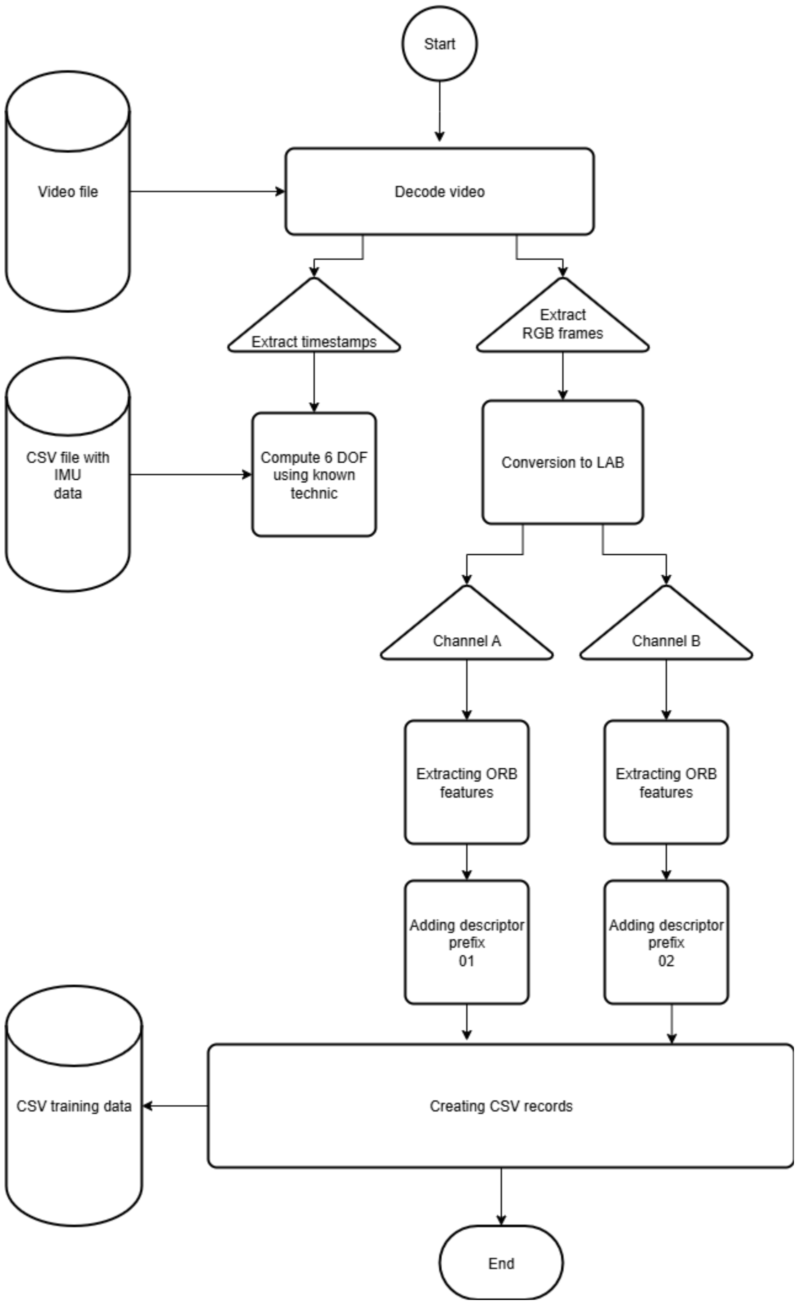

Figure 6: Diagram of the algorithm for preparing training data.

Table 4
Format for Saving Training Data

Descriptor	Screen coordinates		Robot coordinates			Robot orientation		
	U	V	X	Y	Z	h_x	h_y	h_z

In the course of the tasks, it was found that the direct detection of features on the Grayscale or RGB (separate ORB detector per channel) image is vulnerable to changes in lighting from daylight to artificial. The features that stand out under the daytime lighting are not reproduced with artificial lighting, and vice versa. In order to improve the persistence of feature detection, it was decided to convert the image to the LAB color space and select the features from channels A and B separately.

To distinguish features originating from different channels, the prefix 01 (for channel A) and 02 (for channel B) is added to their descriptors.

The current block diagram of the algorithm is as follows (Fig. 6).

A part of the room was used as a test site - an area of complex configuration shown in Figure 7.

For the convenience of calculations, the beginning of the general coordinate system coincides with the starting point of the training trajectory. The training trajectory passes through several points with pre-measured coordinates W1, W2, W3 and ends at point E (coordinates are also measured).

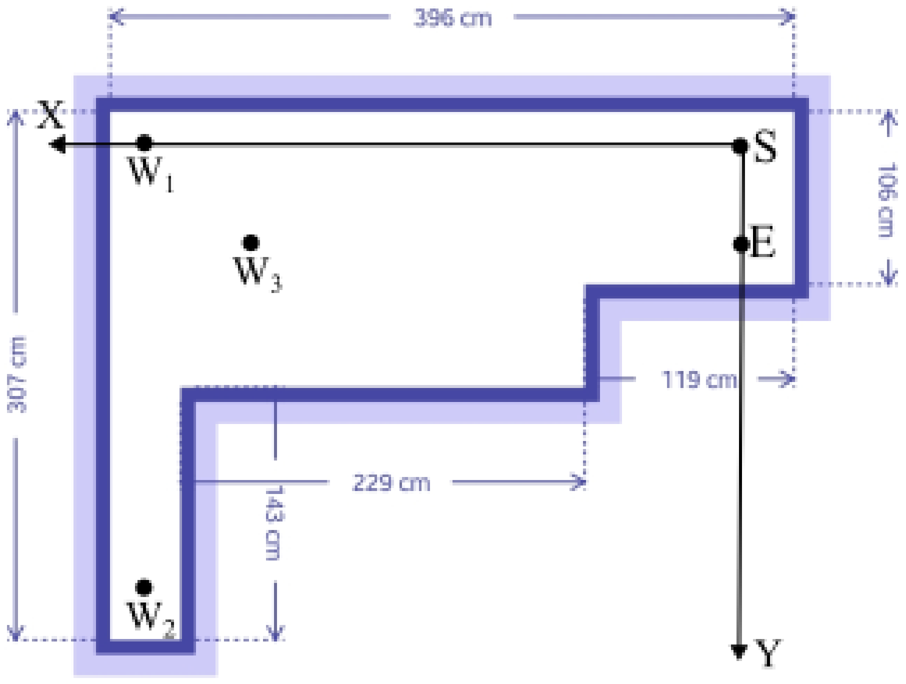

Figure 7: Test site plan.

For storing map data, the following CSV file is currently used:

Table 5
CSV map file structure

Descriptor (hex representation)	Approximation coefficients		
	$a_{0,0,0,0,0,0,0,0}$	...	$a_{P,P,P,P,P,P,P,P}$

The map file generator structure is straightforward:

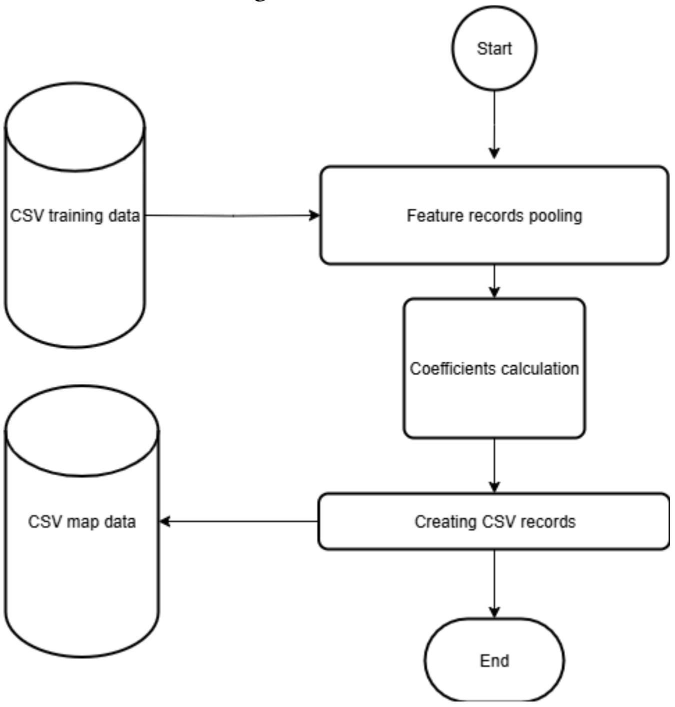

Figure 8: Map generator block schema.

Note: Currently, the map generator and localizer algorithms use a reduced power factor P = 2.
The localizer algorithm schema is shown in Fig. 9.
The output CSV trajectory file structure is presented below:

Table 6
CSV estimated trajectory file structure

Timestamp	Estimated robot coordinates			Estimated robot heading		
	X	Y	Z	h_x	h_y	h_z

An algorithm was also developed for intermediate processing of training data.

6. Experiments

Several data collections were carried out under different lighting conditions:

1. Natural light (daylight from a window on a sunny day).
2. Artificial lighting (cold white light) 100% brightness.
3. Artificial lighting 50% brightness.

The average recording time was 30 seconds.
The ORB detector was configured to allocate up to 500 characters per channel per frame.
A total of 6 data collections were performed:

1. 3 entries – for the version of the algorithm with highlighting features from the Grayscale image in 3 different lighting modes.
2. 3 entries – for the version of the algorithm with the selection of features from LAB A, B channels, also for 3 lighting modes. It is worth noting that the daytime regime in this case differed due to weather conditions.

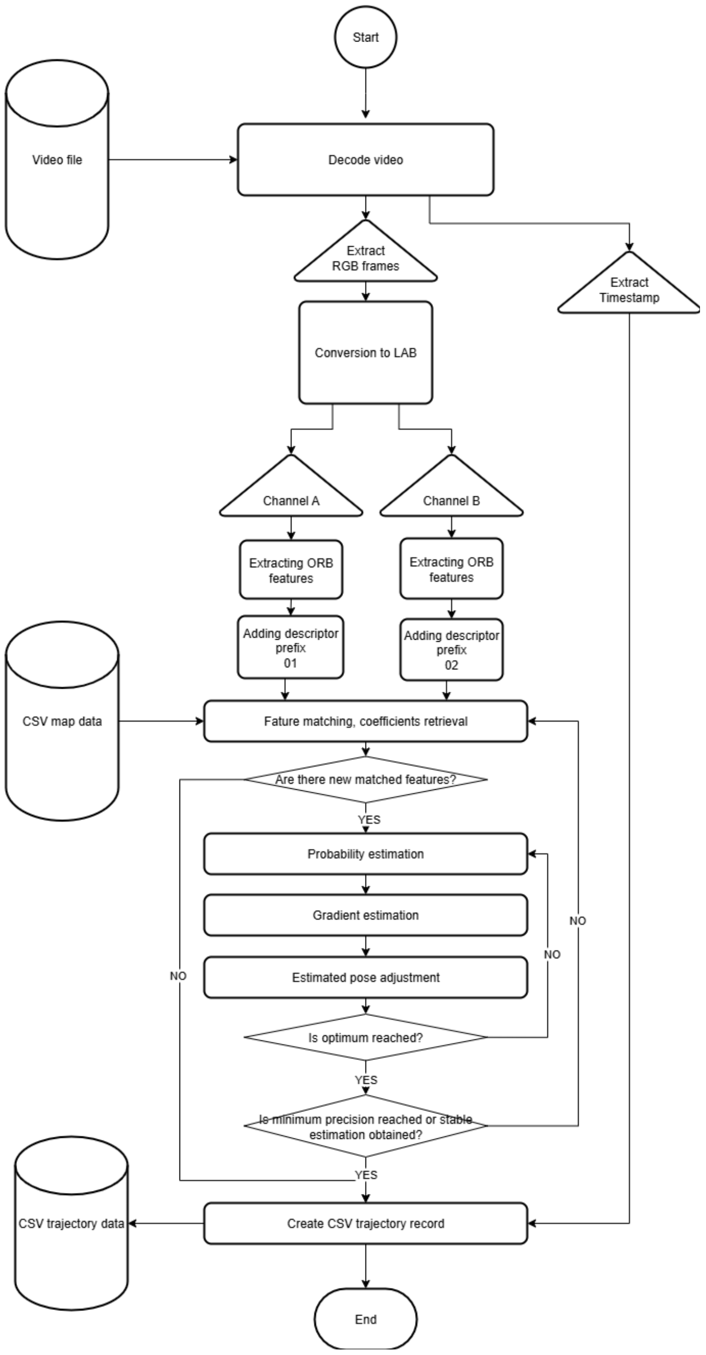

Figure 9: Localizer algorithm block schema.

7. Conclusions

As was mentioned before, the system is in a work-in-progress state. At the moment of this paper writing, only the first training data collection and evaluation were performed, together with mapper and localizer debugging runs on mock-up data. Currently, we are working on proof-of-concept implementation of described system and improving the mapping phase (ideally, to achieve feature observation-time iterative coefficient estimation).

Declaration on Generative AI

During the preparation of this work, the authors used ChatGPT and Grammarly in order to: Grammar and spelling check, and as a smart Search Engine to find related works based on the context of the conversation. After using these tools/services, the authors reviewed and edited the content as needed and take full responsibility for the publication's content.

References

[1] Y. Yasuda, L. Martins, F. Cappabianco, Autonomous visual navigation for mobile robots: a systematic literature review, ACM Comput. Surv. 53 (2020) 1–34. doi:10.1145/3368961.

[2] A. Alfarano, L. Maiano, L. Papa, I. Amerini, Estimating optical flow: a comprehensive review of the state of the art, Comput. Vis. Image Underst. 249 (2024). doi:10.1016/j.cviu.2024.104160.

[3] C. Liu, C.-F. Yeh, C.-C. Lo, Selective intersection flow: a lightweight optical flow algorithm for micro drones, in: Proceedings 2025 IEEE 5th International Conference on Electronic Communications, Internet of Things and Big Data, IEEE ICEIB ’2025, MDPI, Basel, Switzerland, 2025. doi:10.3390/engproc2025108047.

[4] Q. Yang, Y. Wang, L. Liu, X. Zhang, Adaptive fractional-order multi-scale optimization TV-L1 optical flow algorithm, Fractal Fract. 8 (2024). doi:10.3390/fractalfract8040179.

[5] T. Brox, A. Bruhn, N. Papenberg, J. Weickert, High accuracy optical flow estimation based on a theory for warping, in: Proceedings of the 8th European Conference on Computer Vision, ECCV ’2004, Springer, Berlin, Germany, 2004, pp. 25–36. doi:10.1007/978-3-540-24673-2_3.

[6] G. Klein, D. Murray, Parallel tracking and mapping for small AR workspaces, in: Proceedings of the 6th IEEE and ACM International Symposium on Mixed and Augmented Reality, ISMAR ’2007, IEEE, New York, NY, 2007, pp. 225–234. doi:10.1109/ISMAR.2007.4538852.

[7] C. Campos, R. Elvira, J. J. G. Rodriguez, J. M. M. Montiel, J. D. Tardos, ORB-SLAM3: an accurate open-source library for visual, visual–inertial, and multimap SLAM, IEEE Trans. Robot. 37 (2021) 1874–1890. doi:10.1109/TRO.2021.3075644.

[8] J. Kühne, M. Magno, L. Benini, Low latency visual–inertial odometry with on-sensor accelerated optical flow for resource-constrained UAVs, arXiv preprint arXiv:2406.13345 (2024). doi:10.48550/arXiv.2406.13345.

[9] A. Merzlyakov, S. Macenski, A comparison of modern general-purpose visual SLAM approaches, arXiv preprint arXiv:2107.07589 (2021). doi:10.48550/arXiv.2107.07589.

[10] Z. Teed, J. Deng, DROID-SLAM: deep visual SLAM for monocular, stereo, and RGB-D cameras, arXiv preprint arXiv:2108.10869 (2021). doi:10.48550/arXiv.2108.10869.

[11] A. Bougouffa, E. Seignez, S. Bouaziz, F. Gardes, An indoor DSO-based ceiling-vision odometry system for indoor industrial environments, arXiv preprint arXiv:2412.02950 (2024). doi:10.48550/arXiv.2412.02950.

[12] J. Engel, T. Schöps, D. Cremers, LSD-SLAM: large-scale direct monocular SLAM, in: Proceedings of the 13th European Conference on Computer Vision, ECCV ’2014, Springer, Cham, Switzerland 2014, pp. 834–849. doi:10.1007/978-3-319-10605-2_54.

[13] S. Boche, J. Jung, S. B. Laina, S. Leutenegger, OKVIS2-X: open keyframe-based visual-inertial SLAM configurable with dense depth or LiDAR, and GNSS, IEEE Trans. Robot. (2025) 1–20. doi:10.1109/TRO.2025.3619051.

[14] Y. Wang, S. Zhang, J. Wang, Ceiling-view semi-direct monocular visual odometry with planar constraint, Remote Sens. 14 (21) (2022) 5447. doi:10.3390/rs14215447.

[15] E. Simsek, B. Ozyer, Deep learning enhanced monocular visual odometry: advancements in fusion mechanisms and training strategies, Image Vis. Comput. 162 (2025) 105732. doi:10.1016/j.imavis.2025.105732.

[16] M. Bloesch, S. Omari, M. Hutter, R. Siegwart, Robust visual–inertial odometry using a direct EKF-based approach, in: Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS ’2015, IEEE, New York, NY, 2015, pp. 298–304. doi:10.1109/IROS.2015.7353389.

[17] T. Qin, S. Cao, J. Pan, S. Shen, A general optimisation-based framework for global pose estimation with multiple sensors, IET Cyber-Systems Robot. 7 (2025). doi:10.1049/csy2.70023.

[18] J. Huai, Y. Lin, Y. Zhuang, C. Toth, D. Chen, Observability analysis and keyframe-based filtering for visual–inertial odometry with full self-calibration, arXiv preprint arXiv:2201.04989 (2022). doi:10.48550/arXiv.2201.04989.