ROS 2 Implementation of Appearance-based Visual Teach and Repeat Navigation

Yuliia Perevozniuk, Zdeněk Rozsypálek and Tomáš Krajník
Czech Technical University in Prague

***Abstract*— Visual Teach-and-Repeat methods are particularly useful in scenarios where a robot is required to repeatedly traverse the same trajectory, while manual operation during each traversal becomes impractical. This paper presents a ROS 2-based modernization of an appearance-based VT&R navigation framework using classical Bearnav navigation and an improved particle filter sensor fusion. Since the previously commonly used ROS 1 is now deprecated, this work provides the first open-source ROS 2 implementation of appearance-based VT&R methods. The implemented system integrates perception, mapping, route-following, and control into a modular, distributed navigation pipeline that supports both teach-and-repeat navigation modes. The migration from ROS 1 to ROS 2 required architectural changes in communication, synchronization, and asynchronous system execution. The resulting ROS2-based system was evaluated on a real robotic platform, validating the successful migration and stable operation under practical conditions.**

I. Introduction

Autonomous navigation in dynamic and long-term deployment scenarios continues to present significant challenges in robotics, particularly in outdoor environments where visual appearance frequently changes. Visual Teach-and-Repeat (VT&R) methods address these challenges by enabling robots to navigate through matching current observations with previously recorded traversals.

There are two main conceptually different approaches to performing the Teach-and-Repeat process. The first approach constructs a consistent metric map during mapping and attempts to repeat the trajectory within it. Instances of this method are [1], [2], [3], where VT&R3 has a publicly available ROS 2 implementation. The second appearance-based approach does not rely on localisation inside a metric map, but rather on a reactive bioinspired behaviour [4], [5], [6]. The map is a set of consecutive images, and visual servoing is used to steer the robot towards the mapped path. To this date, there is no publicly available ROS 2 implementation of the appearance-based VT&R.

In this paper, we present an open-source ROS 2 [7] implementation of appearance-based VT&R method[1]. The working principle, coined 'convergence theorem' [4] behind the navigation method, allowed us design a system as a set of interconnected ROS 2 nodes. Any of these nodes, including the core modules responsible for feature extraction, image registration, and sensor fusion, can be replaced without the need to alter other ROS 2 nodes. ROS 2 works well for these needs because it has a flexible design, makes it easy for components to work together, and handles real-time data processing. This ensures that data from different sensors can be aligned, processed in real time, and shared efficiently between system parts.

Practical deployment often reveals system-level challenges that are not resolved at the algorithmic level. Specifically, maintaining consistent data flow among perception, mapping, and control components, managing asynchronous execution, and ensuring accurate synchronization of sensor streams remain complex tasks. These challenges have a direct impact on navigation stability and map consistency, particularly during real-time operation. This work shows that organizing the VT&R pipeline as a modular, asynchronous system improves adaptability, focusing on system architecture.

II. System Design

The presented system uses a distributed ROS 2 with dedicated nodes for mapping, feature extraction, sensor fusion, trajectory replay, and control. A classical Bearnav navigation [8] was chosen for the implementation, as it is a reliable feature-based method for navigation. The method relies on visual feature matching combined with odometry information, which makes it relatively robust to moderate environmental changes while maintaining low computational requirements. The particle filter improves navigation by integrating camera images with wheel data to correct distance errors and ensure smooth movement. Instead of relying on a single estimate, the particle filter evaluates multiple hypotheses of the robot state along the taught path, where each particle represents a possible trajectory estimate. During navigation, particle states are predicted using odometry measurements and weighted by the similarity between the current camera image and the corresponding map images. Particles with higher visual consistency receive larger weights. It also enables the robot to select the most suitable map based on environmental conditions.

The *Mapmaker* node acts as an action server during the teach phase, handling map initialization and generation, and recording control commands. During operation, it receives three primary data streams: visual features from camera images, raw image data, and estimated traveled distance. These inputs are temporally aligned through approximate synchronization and processed in a single callback to maintain observation consistency. In this synchronized processing step, the system represents the current robot state by combining the latest feature descriptor, image, and distance estimate.

[1]The source code is publicly available at `https://github.com/Juliaa65/pfvtr_ros2`

Waypoints are created when either a predefined distance threshold is exceeded or when visual displacement between consecutive observations indicates a significant change in viewpoint. Each waypoint is stored with its feature representation, timestamp, and optional alignment information. Together, these waypoints form a recorded route representation that can later be used during repeat traversal. Simultaneously, executed control commands and corresponding odometry are recorded and indexed by traveled distance to support consistent replay during navigation.

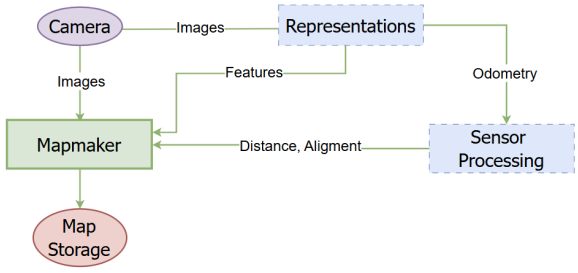
A dedicated perception component manages feature extraction and visual matching by continuously processing camera images and converting them into learned feature representations. When map data is available, it matches current observations with previous frames and stored map features. Visual matching can be performed using feature embeddings generated by a Siamese network. This process generates similarity distributions that indicate how well the current observation aligns with various map regions. By isolating this functionality in a separate node, the system ensures perception operates independently from mapping and control logic.

Sensor fusion is implemented by the *Sensors* node, which estimates the robot's traveled distance and relative alignment by combining odometry data with visual alignment cues from feature matching. The node supports multiple interchangeable sensor-fusion approaches, including a particle filter and classical BearNav. This modular structure allows different navigation strategies to operate within the same ROS 2 architecture while sharing a common communication interface.

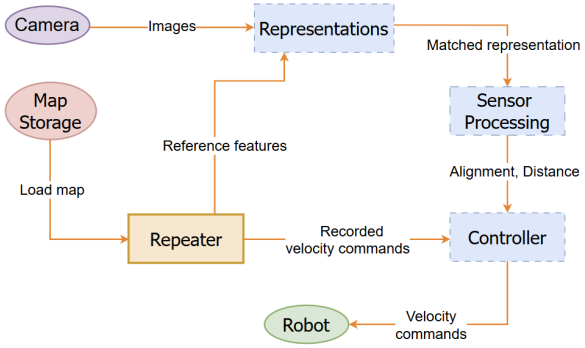
During the repeat phase, the *Repeater* node acts as an action server that loads the previously recorded map and reconstructs its internal representation, including feature descriptors, timestamps, and transitions between waypoints. The system then uses the current estimated position to select a local subset of map data, which it provides to the perception module for matching. This approach helps to ensure that only relevant reference data is used at each time step.

Trajectory replay combines recorded motion commands with visual trajectory alignment. The system corrects these commands using the current alignment estimate to compensate for drift and environmental changes. The control component as *Controller* node adjusts the velocity command in proportion to the estimated alignment error and scales linear velocity independently. This approach keeps the robot on the recorded trajectory while continuously correcting its orientation using visual feedback.

A key feature of the proposed system is the use of ROS 2, which improves the stability and responsiveness of the VT&R pipeline, making it efficient and reliable. The overall architecture of the implemented VT&R system is divided into separate teaching and repeat pipelines, shown in Fig. 1a and Fig. 1b, respectively. Parallel and asynchronous execution is used throughout the system architecture. During the teaching phase, camera images, visual feature extraction, sensor fusion, waypoint recording, and command recording operate concurrently as independent ROS 2 callbacks. During the repeat phase, the system uses ROS 2 *MultiThreadedExecutor*, allowing multiple callbacks to execute concurrently. Route progression estimation, visual-feature publishing, action replay, controller updates, and sensor-processing callbacks operate in parallel while communicating through ROS 2 topics and actions.



(a) Teach phase



(b) Repeat phase

Fig. 1: System architecture of the proposed VT&R framework: (a) the *Mapmaker* teach flow used for route recording, and (b) the *Repeater* flow used for trajectory replay. Visual Representation and Sensor Processing modules can be independently replaced, allowing different methods to be integrated without modifying the remaining system architecture.

III. ROS 2 Integration

The migration of the navigation system from ROS 1 to ROS 2 required modifications across the entire software architecture, including node management, communication interfaces, synchronization mechanisms, and data recording. One of the most noticeable changes was the transition from the centralized ROS 1 communication model, which relied on the ROS master node for topic and service discovery, to the decentralized communication architecture built on the Data Distribution Service (DDS) middleware in ROS 2 [7].

ROS 2 introduced Quality of Service (QoS) settings for topic communication. This affected the migrated system because communication behavior between publishers and subscribers now has to be configured explicitly. Unlike ROS 1, where communication behavior was largely hidden behind the TCPROS transport layer and controlled primarily by queue size, ROS 2 requires communication properties to

TABLE I: ROS2 Quality of Service policies

QoS policy	Description
Depth	Defines the queue size for stored messages. Smaller depth reduces latency, while larger depth allows more buffering.
Reliability	Defines message delivery behavior. `BEST_EFFORT` may drop messages, while `RELIABLE` retries delivery.
Durability	Defines whether messages are stored for late-joining subscribers. `VOLATILE` does not preserve old messages, while `TRANSIENT_LOCAL` keeps them available.

TABLE II: QoS profiles used in the ROS2 implementation

QoS configuration	Purpose

Depth = 1
`BEST_EFFORT`
`VOLATILE`
	Used for high-frequency navigation streams such as camera images and live feature outputs, where low latency is more important than guaranteed delivery.

Depth = 10
`BEST_EFFORT`
`VOLATILE`
	Used for synchronization of multiple sensor streams, where recent messages must remain buffered because data from different topics may arrive with small timing differences.

Depth = 10
`RELIABLE`
`VOLATILE`
	Used for synchronized teach-phase waypoint generation, where loss of synchronized image and feature data could create gaps in the recorded route representation.

be explicitly defined for every publisher-subscriber pair. These QoS policies determine how messages are buffered, delivered, and maintained within the DDS. ROS 2 validates QoS compatibility before establishing a connection between a publisher and a subscriber. If the QoS settings are incompatible, for example when one side uses `RELIABLE` communication while the other uses `BEST_EFFORT`, the connection is not established and messages are silently dropped. In many situations, this does not produce a clear runtime error, making QoS mismatches one of the more difficult debugging problems during the migration process. The QoS policies used in the implemented VT&R system are summarized in Table I.

Several dedicated QoS profiles were created for the migrated ROS 2 implementation, which are summarized in Table II. These policies define queue depth, reliability, and durability behavior.

Synchronizing multiple data streams is essential for accurate waypoint creation. Image data, feature representations, and distance estimates are generated by separate nodes and must be temporally aligned before processing. Approximate time synchronization enables the system to combine consistent observations in a single processing step. Without proper synchronization, the system may use incomplete or misaligned data, resulting in incorrect waypoint generation and reduced navigation performance.

The communication model benefits from separating continuous data streams from on-demand computation. Streaming interfaces handle high-frequency data, such as features and distance estimates, while request-response mechanisms manage operations like alignment or state initialization. This approach enables more predictable execution and simplifies system design.

ROS 1 blocking service calls were replaced by asynchronous ROS 2 services and action servers using future-based execution. The `Mapmaker` and `Repeater` components were migrated to ROS 2 interfaces with dedicated goal, cancel, and execute callbacks, allowing long-running teach and repeat operations to execute asynchronously without blocking the rest of the navigation pipeline.

The ROS 2 executor model was introduced during migration. The migrated implementation uses multi-threaded executors and callback groups to enable parallel execution of computationally intensive operations, such as visual feature extraction. This differs from the largely sequential execution behavior of the original ROS 1 implementation and improves the overall responsiveness of the navigation system.

The bag-recording subsystem was also redesigned during the migration process. The original ROS1 `rosbag` API was replaced with the ROS 2 `rosbag2` using MCAP storage and explicit message serialization. In contrast to ROS 1, where messages could be written directly to a bag file, the ROS 2 implementation requires explicit topic registration, serialization handling, and timestamp management. Storage backends, serialization formats, and bag handling behavior can now be configured independently. Additionally, the MCAP format offers better compatibility with modern ROS 2 tooling for long-term data recording and playback.

IV. Experiments in simulation

The experiments in this section compare two trajectory-following approaches used during the repeat phase of navigation: the Bearnav method and PFVTR. Varying longitudinal and lateral offsets are introduced at the start of a repeat run. The primary objective is to investigate the convergence robustness of each method, and their ability to recover the robot pose after initialization errors in forward, backward, or sideways directions relative to the reference trajectory. The experiments also analyze how navigation performance is influenced by the visual characteristics of the environment.

Two maps were used in the evaluation. The first, called the simple map, contains a relatively simple route structure with long straight segments and smooth turns. The second map, denoted as the difficult map, has a more complex and irregular route structure with sharper turns and more complicated trajectory geometry. Compared to the simple map, the difficult one introduces larger variations in control commands and more significant visual changes between consecutive observations. Example of frame captured from the simulated environment [9] during the experiments is shown in Fig. 2. All experiments were performed in simulation to ensure repeatable, controlled testing conditions.

First, a reference route was recorded during the teach phase and stored as a mapped trajectory. In the repeat phase, the robot was initialized with different position offsets from the original route start, and the corresponding average deviation from the mapped trajectory was calculated.

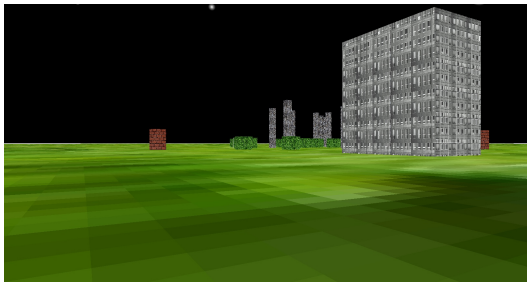


Fig. 2: Example of frame captured in the simulated environment used during evaluation

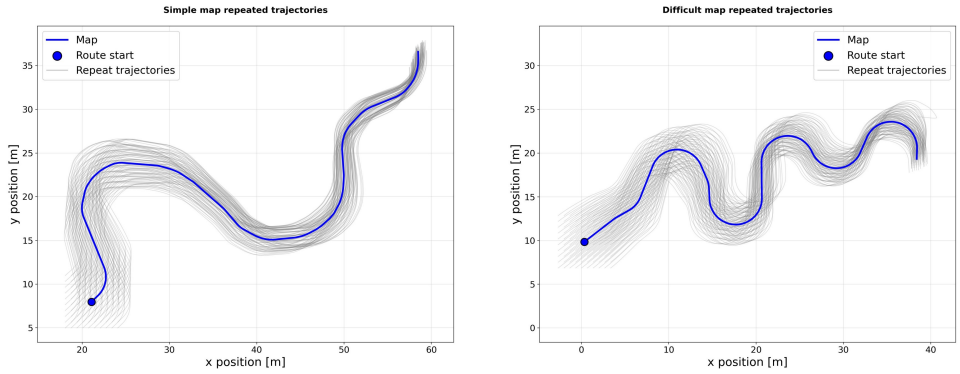
The recorded route was stored as a map consisting of sequentially recorded frames, feature representations, and associated odometry data. In both environments, the recorded map length was approximetly $60\,\mathrm{m}$. During repeat execution, the robot attempted to follow this mapped route while estimating its position relative to the stored trajectory. The resulting reconstructed trajectory was saved for each run and later compared with the mapped one. Fig. 3 shows examples of reconstructed trajectories obtained using the *BearNav* and *PFVTR* methods, respectively.

Before each repeat traversal, the robot started from a shifted initial position. Two types of offsets were applied: longitudinal displacement d_x, moving the robot forward or backward along the route, and lateral displacement d_y, shifting it sideways relative to the route. Different combinations of d_x and d_y were tested using a two-dimensional grid of initial offsets. Each grid cell represents one complete repeat traversal from a specific displaced position.

For every tested offset, the reconstructed repeat trajectory was compared with the original mapped trajectory. The main evaluation metric was the Chamfer distance between the two trajectories. This metric measures how closely the reconstructed path follows the mapped route by computing the average distance between nearby points. Since the trajectories are in map coordinates, all values are expressed in metres. Lower values indicate accurate trajectory reconstruction and successful convergence to the mapped route. Higher values indicate trajectory drift or failure to converge to the mapped trajectory.

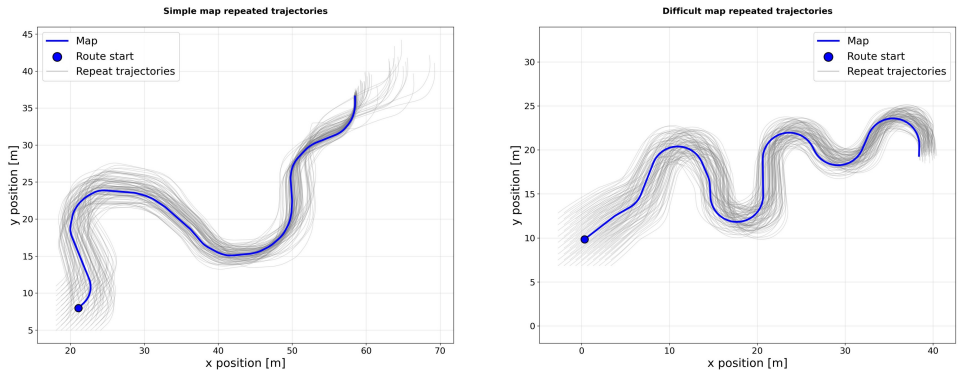
The reconstructed trajectory comparison in the simple map shows that both the classical Bearnav method and PFVTR generally follow the mapped route, although deviations appear for larger initial displacements. The corresponding convergence basins are shown in Fig. 4a.

The *BearNav* method demonstrates smooth and spatially



(a) Simple map BearNav

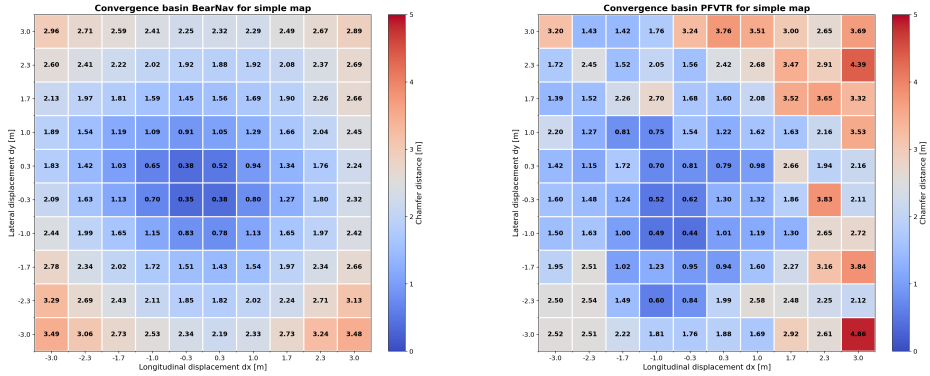
(b) Difficult map BearNav



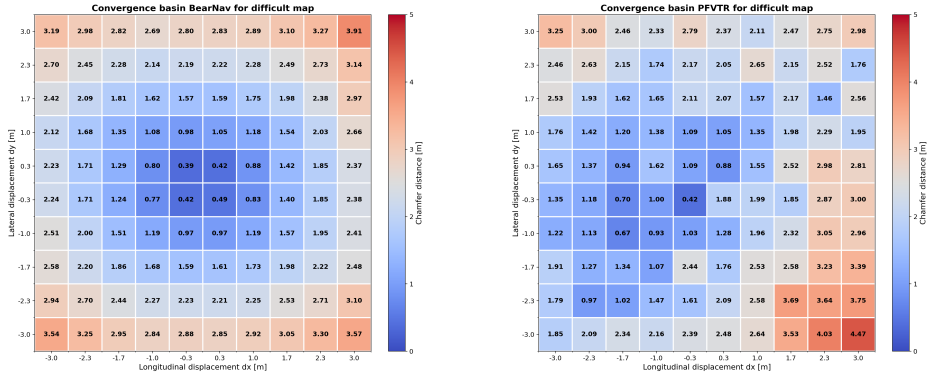
(c) Simple map PFVTR

(d) Difficult map PFVTR

Fig. 3: Examples of reconstructed trajectories obtained using different navigation methods. The blue curve represents the mapped trajectory, while the gray curves correspond to reconstructed traversals from different initial offsets



(a) Simple map



(b) Difficult map

Fig. 4: Convergence basins obtained for the evaluated navigation methods in both simulated environments

consistent convergence behavior across most of the displacement grid. Reconstruction errors remain low near the mapped starting position and gradually increase toward larger offsets, indicating stable degradation of repeat-navigation performance.

PFVTR achieves similarly low reconstruction errors near the center of the displacement grid, indicating successful convergence for moderate initialization offsets. However, larger positive longitudinal displacements and corner regions of the basin produce less stable convergence behavior and larger reconstruction errors.

The smoother behavior of *BearNav* can largely be attributed to the deterministic nature of the simulation environment. Since the simulated robot motion contains minimal odometry noise, replay of recorded control commands remains highly repeatable. In contrast, the simulated environment contains relatively limited texture diversity and simplified visual appearance, reducing the amount of distinctive visual information available for *PFVTR*.

The convergence basins obtained for the difficult map are shown in Fig. 4b. Compared with the simple map, both methods demonstrate larger reconstruction errors due to the increased trajectory complexity caused by sharper turns and more irregular route geometry.

BearNav maintains a smooth and spatially consistent convergence pattern across most of the displacement grid. The minimum Chamfer distance is approximately 0.39–0.49 m near the center of the basin, while the reconstruction error gradually increases toward larger initial offsets and reaches approximately 3.91 m near the outer regions of the grid. The reconstruction error changes smoothly across neighboring regions without sudden error spikes, indicating stable and predictable degradation of trajectory-following performance as the initial displacement increases.

PFVTR achieves similarly low reconstruction errors in the central region of the displacement grid, with minimum values around 0.42–0.70 m. Several surrounding regions also maintain reconstruction errors below approximately 2 m, indicating successful convergence for moderate initial offsets. However, compared to BearNav, PFVTR demonstrates noticeably larger spatial variability across the displacement grid. Neighboring initialization offsets can produce substantially different reconstruction errors, particularly for larger positive longitudinal displacements and corner regions of the basin. In these regions, the Chamfer distance increases up to approximately 4.47 m.

Table III summarizes the mean Chamfer distance values for all evaluated methods and maps.

TABLE III: Mean Chamfer distance values

Method	Simple map [m]	Difficult map [m]
BearNav	1.95	2.10
PFVTR	2.01	2.06

The mean Chamfer distance values of both methods remain relatively close across both environments, while the difficult map yields larger errors and less stable convergence than the simple map for both methods, reflecting its increased structural complexity. These mean values should be read in the context of the simulation conditions rather than as a head-to-head ranking of the two methods. The purpose of this experiment is to verify that the migrated ROS 2 system reproduces the expected convergence behaviour of both methods and recovers from a range of initialisation offsets, which the convergence basins in Fig. 4 confirm.

The comparable mean accuracy of the two methods is the expected outcome under these controlled conditions. Since the simulation provides minimal odometry noise, *BearNav* benefits from stable replay of the recorded control commands, while the simulated environments contain relatively limited texture diversity and simplified visual appearance, which reduces the amount of distinctive visual information available to *PFVTR*. Under these conditions the two methods are effectively equivalent, and the advantage of the learned visual representation used by *PFVTR* instead emerges in the real-world experiment (Sec. V), where the scene is visually richer. The simulation therefore validates the correctness and convergence robustness of the migrated pipeline, while the real-world experiment characterises the methods where their differences matter.

V. Real-world Experiment

Experiments were conducted in a real-world outdoor environment using the *VOP Taros v4 6x6* robotic platform shown in Fig. 5.

Fig. 5: VOP Taros v4 6x6 robotic platform used during the real-world experiments

The goal of these experiments was to evaluate the repeatability and long-term trajectory-following stability of both the classical *BearNav* [10] method and *PFVTR* [8] under realistic operating conditions.

During the teach phase, the robot manually traversed the reference route. These data were used to construct the route representation later utilized during repeat navigation. The recorded trajectory formed a closed loop with a total length of approximately $100\,\mathrm{m}$. Example of camera frame captured along the route is shown in Fig. 6.

TABLE IV: Distance from the starting position after each lap

Method	L1 [m]	L2 [m]	L3 [m]	L4 [m]	L5 [m]
BearNav	0.75	0.42	0.18	0.11	-0.02
PFVTR	0.67	–	–	–	–

Fig. 6: Example of frame captured along the recorded real-world route

During the repeat phase, the robot was initialized with an approximate forward displacement of $2\,\mathrm{m}$ relative to the original starting position. The robot then repeatedly traversed the recorded route while estimating its relative progression along the stored trajectory. To obtain ground-truth measurements, a WhyCode visual marker was used [11].

To evaluate long-term trajectory consistency, repeated laps were performed and the distance between the final robot position and the original mapped starting point was measured, as summarized in Table IV. Lower values indicate smaller accumulated trajectory drift and better repeatability.

The reconstructed trajectories for both methods are shown in Fig. 7 and Fig. 8.

Fig. 7: Repeated traversals using the *BearNav* method. The red curve represents the recorded route, while the blue curves correspond to the reconstructed traversals.

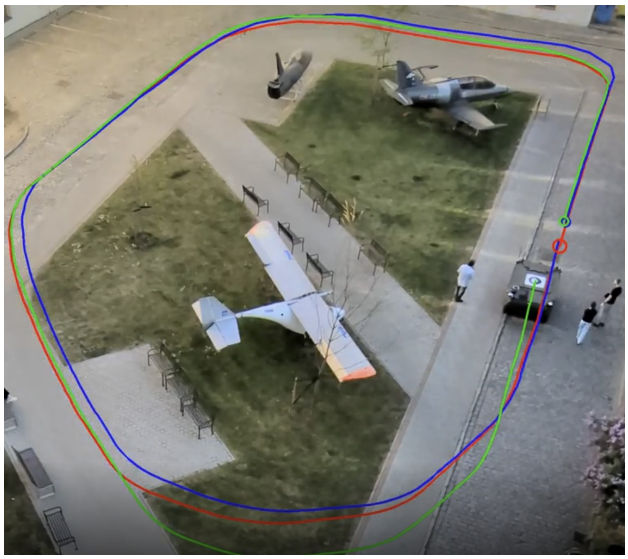


Fig. 8: Repeated traversals using the PFVTR method. The red curve represents the recorded route, the blue curve corresponds to *BearNav*, and the green curve represents the PFVTR traversal.

BearNav demonstrated gradual reduction of accumulated trajectory drift across consecutive laps, indicating stable trajectory correction over time. The final deviation decreased from $0.75\,\mathrm{m}$ after the first lap to approximately zero after five traversals.

PFVTR achieved a slightly lower initial deviation of $0.67\,\mathrm{m}$ and tracked the recorded route more tightly than BearNav, showing stronger lateral convergence to the taught path (Fig. 8). Unlike BearNav, which derives longitudinal progress purely from odometry, PFVTR fuses visual matching into its distance estimate; this generally improves accuracy but makes the distance estimate more sensitive to severe image degradation. During the traversal, direct sunlight saturated nearly the entire camera frame (Fig. 9), temporarily corrupting the visual distance estimate and producing the localised longitudinal deviation visible at the bottom of Fig. 8. Despite the almost completely white image, the lateral alignment correction remained stable and the robot recovered and successfully completed the full traversal. This indicates that the dominant failure mode of PFVTR under extreme glare is longitudinal distance mis-estimation rather than loss of route convergence, and that the learned feature representation is robust enough to keep the robot on the path and finish the loop. We note that BearNav was evaluated over five repeated laps, whereas PFVTR was assessed on a single completed traversal; a multi-lap repeatability comparison for PFVTR is left for future work.

Fig. 9: Sample image affected by severe sunglare during the PFVTR traversal. The almost fully saturated frame temporarily corrupts the visual distance estimate, causing the localised longitudinal deviation visible at the bottom of Fig. 8. Lateral convergence was maintained and the robot completed the loop.

VI. CONCLUSIONS

This work presents a ROS 2-based implementation of appearance-based VT&R navigation. The system has been implemented and evaluated in both simulated and real-world environments. The implementation provides capabilities similar to those of its ROS 1 counterpart while leveraging the ROS 2 ecosystem for modularity and communication features. The navigation system records waypoints during the teach phase and later follows the same trajectory by matching current observations to stored features and applying alignment-based corrections. The implementation combines all main components of the autonomous navigation, including perception, map representation, trajectory following, and robot control, into a single working system. A simple graphical user interface was added to simplify interaction with the system, making it easier to start mapping and run repeat navigation.

Simulation experiments demonstrated that the migrated system produced stable navigation behavior during repeated route traversals across environments in which it was run. In simulation, BearNav achieved better results because it relies mainly on velocity commands and was less affected by the limited visual quality of the simulated environment. In contrast, PFVTR demonstrated better performance in real-world experiments, where the higher visual fidelity allowed more reliable visual matching.

ROS 2 organizes the implementation into independent nodes, enabling parallel operation and simplifying communication between them. Features such as asynchronous execution and configurable QoS policies ensure stable data exchange and minimize blocking between nodes. The system can be enhanced with advanced methods. Its modular design supports scalability and improved user interaction.

REFERENCES

[1] P. Furgale and T. Barfoot, "Visual teach and repeat for long-range rover autonomy," *J. Field Robotics*, vol. 27, pp. 534–560, 2010.

[2] M. Paton, K. MacTavish, M. Warren, and T. Barfoot, "Bridging the appearance gap: Multi-experience localization for long-term visual teach and repeat," *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 1918–1925, 2016.

[3] M. Nitsche, F. Pessacg, and J. Civera, "Visual-inertial teach and repeat," *Robotics and Autonomous Systems*, vol. 131, p. 103577, 2020.

[4] T. Krajnık, F. Majer, L. Halodová, and T. Vintr, "Navigation without localisation: reliable teach and repeat based on the convergence theorem. in 2018 ieee," in *RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2018, pp. 1657–1664.

[5] L. G. Camara and L. Přeučil, "Visual place recognition by spatial matching of high-level cnn features," *Robotics and Autonomous Systems*, vol. 133, p. 103625, 2020.

[6] D. Dall'Osto and Others, "Fast and robust bio-inspired teach and repeat navigation," in *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2021, pp. 500–507.

[7] S. Macenski, T. Foote, B. Gerkey, C. Lalancette, and W. Woodall, "Robot operating system 2: Design, architecture, and uses in the wild," *Science Robotics*, vol. 7, no. 66, p. eabm6074, 2022.

[8] Z. Rozsypálek and Others, "Multidimensional particle filter for long-term visual teach and repeat in changing environments," *IEEE Robotics and Automation Letters*, vol. 8, no. 4, pp. 1951–1958, 2023.
[9] M. Simon, Z. Rozsypálek, and T. Krajnik, "Voxels: A lightweight simulation for mobile robotics," in *ICRA 2026, LoWi workshop*, To be published.
[10] T. Krajník, J. Faigl, V. Vonásek, K. Košnar, M. Kulich, and L. Přeučil, "Simple yet stable bearing-only navigation," *Journal of Field Robotics*, vol. 27, no. 5, pp. 511–533, 2010.
[11] J. Ulrich, P. Lightbody, A. Weinstein, F. Majer, and T. Krajnık, "Whycode: Efficient and versatile fiducial localisation system," *IROS 2018*, 2018.