DRONE-SLAM: Dense Registration and Odometry for Near-field Environments with UAVs

Diego Navarro [1,2], Ezio Malis [2], Raphael Antoine [1], and Philippe Martinet [2]
[1]ENDSUM team. Cerema Normandie-Centre, France
[2]ACENTAURI team. Centre Inria d'Université Côte d'Azur, France
e-mail: diego-navarro.tellez@inria.fr

***Abstract*—Structural health monitoring applications (SHM) require large volumes of data that can only be acquired with automatic methods, such as robots or UAVs. However, localizing a drone in close proximity to the target structure poses significant challenges, particularly when the GPS is not precise enough. A possible solution is to use vision-based localization in a previously build model of the target. The localization can be however very challenging if, for safety reasons, the reconstruction of the model has been acquired far from the target and the target has a low-texture. These issues often result in reduced data density and obstruct convergence to the true position. The majority of previous work deal with the localization at far-field regions, while only few methods address the vision-based localization close to targets with low-texture. This paper presents a novel Dense Visual SLAM method tailored for close-range localization to surfaces using unmanned aerial vehicles (UAVs) in GPS degraded conditions. Our method uses a custom registration method to enable realistic rendering with dense maps, designed for close-range visual odometry and surface modeling. The system operates in two steps: First, the UAV performs an exploratory flight with a stereo camera to build a dense map, modeling surfaces as ellipsoids. Secondly, the system exploits the map to generate reference data, enabling dense visual odometry (DVO) in close proximity to the surfaces without the need of stereo data. Experiments in realistic simulated environments demonstrate the system's capability to localize the drone within 16 cm accuracy at a distance of 2 m from the surface, outperforming existing state-of-the-art approaches. Tests on real data confirm this performance in real low-texture scenarios.**

Keywords-robotics; autonomous vehicles; vision and scene understanding; volumetric image representation.

I. Introduction

This work builds upon the hybrid SLAM method introduced in [1], keeping the two-step workflow while incorporating significant advancements. Such methodological improvements reflect the adaptation of localization techniques for UAV-based monitoring and inspection tasks. UAVs have experienced a rapid increase in adoption across diverse fields [2] in recent years. Natural hazards and civil engineering sectors are no exception, with growing enthusiasm for utilizing these advanced tools in classification surveys and inspections [3][4]. UAVs can operate either manually, with pilots directly controlling the drone, or autonomously, using technologies such as environmental sensing and self-localization. Recent technological advancements have led to the development of autonomous drones capable of navigating intricate and geometrically complex environments [5]. As a result, their role has evolved in applications such as infrastructure inspections, cliff surveys, and more generally in environmental monitoring [6].

While some surveilling applications can be performed with current UAV localization techniques [7], more demanding tasks like close-range inspection are still an issue. Such tasks enable a more effective analysis and diagnosis through SHM techniques [8]. With richer data, like radar measurements ([9], [10]), the quality of structure's digital twins and their pertinence can be greatly improved. However, these measurements require proximity to the surface, which is often done manually.

Manual acquisition not only poses significant risks to human agents but also lacks in coverage. Although UAVs have the potential to solve this problem but there are some challenges to address first. Accurate localization is a critical requirement for deploying autonomous robots for SHM tasks. However, conventional techniques often struggle with the accuracy of the pose estimation, specially in cluttered environments or close to structures.

Traditionally, UAVs are located using GPS or even GPS-RTK modules, while they are convenient, their performance depends heavily on the operation conditions. There are two principal issues. First, the coverage of the GPS signal, which can deteriorate or even fail sometimes. In the case of GPS RTK, the effects of bad reception can degenerate by the proximity to the correction transmission station and the availability of correction data. Secondly, obstacles between the signal sources and the reception antenna can greatly impoverish the quality of the location. Multiple obstacles, typically in urban environments, produce multi-path interference, further degrading localization quality [11].

When GPS signals are unavailable or unreliable, V-SLAM provides a robust alternative for UAV localization. Visual data can serve for localization without worrying about multi-path interference or data availability. However, there are challenges to solve before being able to locate an UAV during close-range inspections. UAVs face strict weight and energy constraints, which limits their capacity to carry both measurement and high-performance localization equipment simultaneously. This can be mitigated by dividing the computational load in two steps: A mapping step with a high-capacity computation payload and a second step with a simplified tracking unit and measurement equipment. With the deactivation of the mapping module during the second step, the tracker can only rely on a pre-generated map as reference.

The lack of map updates introduces other challenges, localizing far from the original map trajectory often results in reduced tracking accuracy. The reasons may vary depending

on the method, one of the most common for feature-based V-SLAM methods is the perspective narrowing. As cameras approach the inspection surface, their field of view narrows, reducing available visual information and rendering previously mapped features unrecognizable from the new perspective.

In this paper, we present a V-SLAM framework tailored for close structure inspection using UAVs. To address the challenges of this use case, we propose a two-step V-SLAM framework. First, a stereo camera captures images during a mapping flight, using a feature-based method for ego-pose estimation. The system's modular design allows flexibility on the choice for mapping ego-localization methods.

The resulting dense map serves dual purposes: aiding mission planning and enhancing localization in the second step. In this phase, precise localization is performed using a lightweight monocular camera, reducing the weight and energy demands typically associated with additional measurement equipment like radar or thermal cameras. Key contributions of our method are:

- A registration method that enhances accuracy and robustness of the localization system.
- Generation of dense map that enables multi-session localization, increasing system versatility.
- Use of dense map for precise localization in close-range inspection scenarios.
- A EWA volume splatting variant, tailored for low density point-cloud rendering.
- Support for agent localization across mixed hardware configurations.

The remainder of this paper is organized as follows: Section II reviews related work that influenced this study. Section III describes the proposed method, with details of the mapping workflow presented separately from the localization modules. Section IV presents the experimental results that validate system performance, along with the test conditions. Finally, the conclusion discusses the results and outlines future research directions to improve the method.

II. Related work

While some existing methods attempt to handle issues like perspective narrowing, few specifically address the challenges posed by scale changes under extreme conditions. Classic V-SLAM approaches propose different strategies for pose estimation. On one hand there are the feature-based methods, such as ORB-SLAM and its successors [12][13], are widely used due to their computational efficiency. However, they perform poorly in low texture scenarios, specially in localization mode, where the system conserves computational resources by relying on pre-generated maps rather than real-time mapping.

On the other hand, dense direct methods [14], [15], derived from SfM techniques (Structure from motion) are capable of locating the agent's location and demonstrate robustness in low-texture regions. This robustness derives form the use of pixel intensity gradient over the entire image to better estimate the pose changes, particularly in conditions where feature-based methods may struggle. However, the reduced field of view near inspection surfaces not only decreases point density, it also introduces gradient inconsistencies, which can compromise the system's localization performance.

Advances in V-SLAM have explored mixed approaches to benefit from the strengths of both feature-based and dense methods. Prior work has explored the creation of dense maps from sparse Key Frames, as presented in [16]. Zhang and Shu integrated a dense mapping component into the ORB-SLAM2[12] framework using stereo data. While effective for merging overlapping point cloud regions, this approach does not address localization challenges in close-range inspections. Specifically, the ORB-SLAM tracker struggles to match descriptors in such environments, likely due to the limitations of scale invariance, where descriptors lose discriminative power at extreme proximity. Moreover, the map generated by Zhang et al.'s dense mapping thread neither addresses this problem nor demonstrates the necessary capabilities to resolve it.

One approach to mitigate perspective changes is to improve the ability to generate pose hypotheses from multiple viewpoints. For example, Kerb et al. [17] proposed a novel modeling method that enables realistic rendering by optimizing 3D ellipsoid parameters using outputs from SfM algorithms. These ellipsoids can be rendered from new viewpoints via Gaussian splatting, an old rasterization technique that is now more affordable thanks to advancements in hardware architectures and rendering methods. This work reignited interest in dense mapping approaches, leading to the emergence of new SLAM techniques inspired by [17]. With proper equipment, VSLAM methods can benefit from more realistic images, increasing their robustness in far-field regions of the map.

Although Gaussian-splatting can mitigate the perspective narrowing issues, it does not guarantee the geometric accuracy of the ellipsoids because its optimization focuses on visual quality rather than spatial precision. Moreover, high quality images remain computationally expensive, even with an optimized implementation. While effective in structured and information-rich environments, they are limited in low texture scenarios. Additionally, their energy consumption and hardware requirements make them impractical with UAV platforms.

Compared to the initial paper [1], this paper introduces new improvements focused to improve the robustness of the system in close-range observations. First, the rasterization method is improved, transitioning from an ellipsoid representation based on the statistical properties of the observations to a model that prioritizes geometric accuracy and incorporates occlusion handling. Secondly, this work expands on the graph-based map management introduced in the previous paper by detailing the logic behind node creation and adding a critical feature: the detection and handling of pixel resolution changes to enhance DVO performance This enhancement enables DRONE-SLAM to detect when incoming images improve the surface model's detail and prioritize their integration into the map. Finally, this paper introduces new experimental validation with real-world data acquired in conditions relevant to close-range

UAV inspections. While [1] primarily focuses on generating a dense map to address the point density problem in close-range observations, this paper extends the approach significantly. It emphasizes geometric accuracy, realistic scene rendering, and detailed implementation, bringing the system closer to a fully integrated dense SLAM framework.

III. Proposed method

To address the challenges of close-range structure inspection we propose a visual localization system. In contrast to GPS-localization that can easily be occluded by the elements in the environment, this kind of methods do not rely solely in fixed sources of information and use the environment in their favor. While the system can work with conventional cameras, it was designed for the application case of UAV deployment. In the current state of implementation, the proposed method is designed to work in two steps. This section explains the main elements of the proposed method. First, we will elaborate over the details of the mapping step, the workflow, how the 3D data is computed and the aggregation of the different key-frames. Then, the elements of the scanning step will be explained in the scanning section along with other key elements of the workflow related to the estimation of the drone pose relative to the environment.

A. Mapping

In the first step the pilot of the mission shall perform a simple exploratory fight around the structure at a safe distance with the mapping payload composed by a pair of synchronized stereo cameras. The main components of the mapping workflow (see Figure 1) are described as follows:

1) System initialization: In this implementation, the UAV is supposed to start close to a common takeoff position for both mapping and scanning steps. This assumption is necessary because, currently, the system lacks a loop closure module capable of recognizing previously visited locations. During the first scene observation, the mapping module computes the first data-cloud from the stereo images and saves it as the first reference data. The selection of the initial observation is crucial for high-quality mapping, as the rest of the map will be forced to be coherent to the scale of this frame. This initial data-cloud is assumed to provide a fair reference, as the flight plan includes a preliminary parameter verification. This can either be done with a calibration check or an on-site adjustment of the depth estimation module's parameters.

2) Depth estimation: The current implementation relies in two different methods for scene depth estimation. For low-texture unstructured scenarios, the depth map is computed using the stereo Block Matching (BM) algorithm provided by the OpenCV library [18]. With proper parametrization, this method can approximate the overall geometry of the scene even in low texture scenes. However, its performance depends heavily on the parametrization, which has to be done manually for each scenario, compromising mapping precision. Moreover, the disparity estimation quality of this module vary depending on the scenario, requiring additional adjustments. In case of very low-textured environments, this algorithm is obliged to chose between sensibility for geometry detection and noise-free depth-maps. That said, the use of this module is provisional until the implementation of a better method is achieved.

The second module used is the CREStereo ML model, proposed by Li et al. [19], which performs well in structured scenarios, offering a full depth-map estimation (BM often skips parts of the images when the texture uniqueness parameter is too strict). For this implementation we used a pre-trained version of the model available on the project's GitHub repository. However, as this model was not specifically trained for our unstructured environments, often mistakes the surfaces of natural formations (i.e. cliff scenario) by planes.

In both cases, the disparity images are collected and used to compute a depth-map that will then be used to build the data-clouds that from now on we will call Surfaces ($\mathbf{S}_x$).

3) Environment modeling: These data-structures model more than a simple point-cloud. While many point-cloud registration methods assume that cameras, very much like lidars, observe a set of perfect 3D points in the space.

Our Surface model, however, considers the fact that cameras are sensors that discretize the space in pixels. Due to technical restrictions linked to the construction of the camera sensor, pixels cannot be infinitely small. In consequence, this data-structure considers each pixel as a patch of the surface that is observed at a certain position to conserve as much information as possible from the observation model. To implement the patch hypothesis, each j pixel is saved with the following data:

- 3D mean position $\mathbf{m}_j$ computed and updated through map registration
- A shape matrix $\mathbf{M}_j$ representing the patch covered by the pixel
- Color information of the pixel $\mathbf{c}_j$

The shape matrix of each pixel $\mathbf{p}_j$ is computed so the ellipsoid can cover a plane matching the size of the pixel in the real world (see Figure 2):

$$w_{px_j}(\mathbf{p}_j) = \frac{d_j w_{cam}}{w_{img} f_{length}} \tag{1}$$

$$h_{px_j}(\mathbf{p}_j) = \frac{d_j h_{cam}}{h_{img} f_{length}} \tag{2}$$

Here, d_j is the depth value at $\mathbf{p}_j$, w_{cam} and h_{cam} are the camera sensor size in meters, w_{img} and h_{img} are the image size in pixels and f_{length} is the focal length in meters. The shape matrix can be composed as follows:

$$\mathbf{M}_j = \begin{bmatrix} \left(\frac{w_{px_j}}{\sqrt{2}}\right)^2 & 0 & 0 \\ 0 & \left(\frac{h_{px_j}}{\sqrt{2}}\right)^2 & 0 \\ 0 & 0 & \left(\frac{h_{px_j}+w_{px_j}}{2}\right)^2 \end{bmatrix} \tag{3}$$

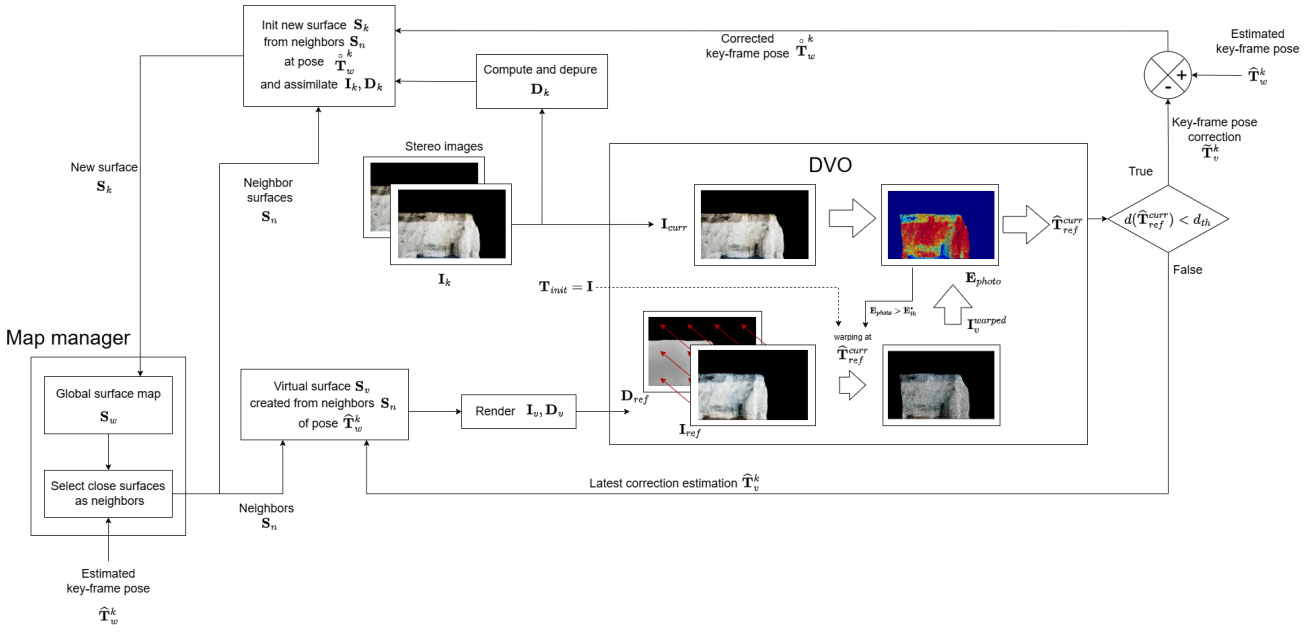

Figure 1. Mapping thread workflow. Input images $\mathbf{I}_k$ are frames selected when the current estimated pose $\widehat{\mathbf{T}}_w^n$ checks any criteria for node creation becoming then $\widehat{\mathbf{T}}_w^k$. DVO module is initialized with identity as the estimated pose and the stereo-images pose should be the same. Tracking thread uses similar structure but instead using virtual reference data $\mathbf{I}_v, \mathbf{D}_v$ the DVO module uses the data stored in the closest surface to the current estimated pose $\widehat{\mathbf{T}}_w^n$.

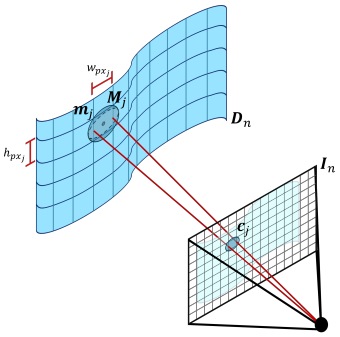

Figure 2. For each n incoming frame there is a color image $\mathbf{I}_n$ and its respective depth-map $\mathbf{D}_n$, the shape matrix is computed so the corners of the pixel plane are tangent to the ellipse border. The surplus ellipsoid coverage is assumed to be forget by the weighted average.

The shape matrices are first computed in the camera frame and then rotated to the global frame so they can be updated with the next observations. The 3D position of the ellipsoid is determined by the weighted average of the point observations along the camera axis. The color information is saved in the form of CIELAB L,A,B coefficients and also fused trough weighted average. This approach contemplates homogeneous lighting across the scenes, further work on light modeling will improve the robustness of the system in more realistic lighting conditions.

4) Pose correction: As the agent moves, new stereo images are continuously acquired. These new observations are used by the tracking module to keep an estimation of the pose of the agent. Simultaneously, the mapping module will select specific frames as *key-frames* $\mathbf{I}_k, \mathbf{D}_k$. At each k selected frame, the incoming stereo images are aligned with the closest reference surface in the map. To achieve this, a virtual observation form the actual estimated pose is computed using the rasterization module. The rendering of the virtual observation will be explained in Section III-C.

Once rendered, the dense odometry module (DVO) computes the pose between the virtual observation of the map $\mathbf{I}_v$, generated at the current estimated pose $\widehat{\mathbf{T}}_w^k$, and the current stereo frame $\mathbf{I}_k$. The resulting pose, $\widetilde{\mathbf{T}}_k^v$, represents the error of the tracking module. In consequence, the true pose of $\mathbf{S}_s$ can be computed as:

$$\overset{\circ}{\mathbf{T}}{}_w^k = \widehat{\mathbf{T}}_w^k \left(\widetilde{\mathbf{T}}_v^k\right)^{-1} \tag{4}$$

Since the visibility of the map ellipsoids depends on the position of the camera, the computation of the correction $\widetilde{\mathbf{T}}_v^k$ is done iteratively. This continues until the position and rotation of the pose are under a threshold defined by the user. For each iteration, a new set of virtual reference image $\mathbf{I}_v$ and depth-map $\mathbf{D}_v$ are computed (feedback loop at the bottom of Figure 1).

5) Surface data-structure: Once the pose is corrected, the surface is supposed aligned with the global map. Under this assumption, information in the incoming frame can either be used to update an existing reference surface or to create a new one.

Initialization: New surfaces are initialized with data from the neighbors and the incoming frame that triggered the creation of the new surface. Since all frames are aligned, neighbor data

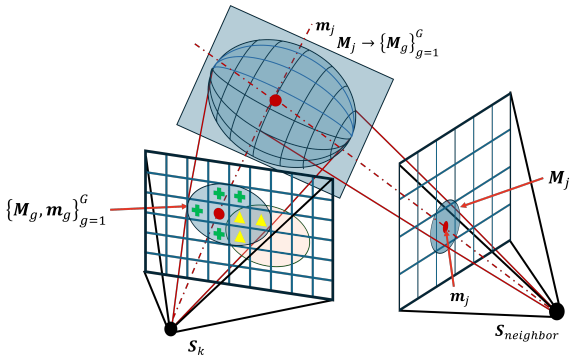

Figure 3. Reference surfaces are initialized with neighbor's data. For instance, the shape matrix $\mathbf{M}_j$ member of a neighbor surface $\mathbf{S}_N$ is projected into $\mathbf{S}_k$ using a linearized projection model around $\mathbf{m}_j$. Perfect correspondences are passed into the grid (red circle), unoccupied pixels covered by the projection are initialized with new ellipsoids (green crosses), occupied cells (orange ellipsoid) are updated following the initialization criteria (yellow triangles). When multiple cells are covered by the projection, the ellipsoid $(\mathbf{M}_j, \mathbf{m}_j)$ becomes the group of the G ellipsoids covered by the projection: $\{\mathbf{M}_j, \mathbf{m}_j\}_{g=1}^{G}$. In $\mathbf{S}_n$ the group will be updated collectively but in $\mathbf{S}_{new}$ each cell preserves individuality.

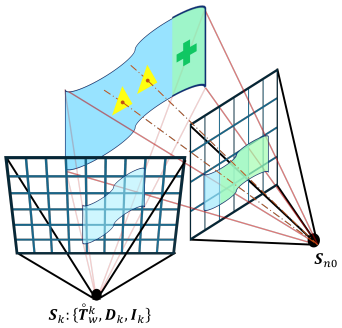

Figure 4. After initialization or when the incoming key-frame does not meet any of the criteria to create a new node, the data is assimilated in the closest reference surface $\mathbf{S}_{n0}$. The selected data $\mathbf{D}_k$ and $\mathbf{I}_k$ are projected into the camera frame of $\mathbf{S}_{n0}$ using an intermediate surface $\mathbf{S}_k$. Unoccupied ellipses are added into the grid (green) and existing ones are updated along the camera axis of $\mathbf{S}_{n0}$ (yellow). We call this operation retro-splatting since a 3D reconstruction of the incoming data in camera frame is *splatted back* into the surface frame.

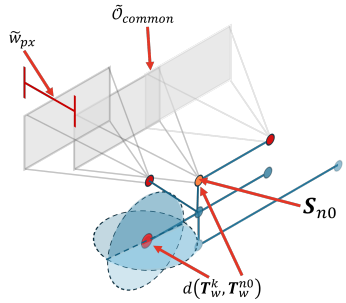

Figure 5. There are different ways to trigger the creation of a new node: The distance inter-surface $d(\mathbf{T}_w^k, \mathbf{T}_w^{n0})$, the pixel size difference $\widetilde{w}_{px}$ and the rate of overlap $\widetilde{\mathcal{O}}_{common}$. These metrics are computed with respect to the closest surface $\mathbf{S}_{n0}$ data.

can be passed to the new surface through a simple projection into the current camera frame (see Figure 3).

In each frame, the data is organized as a grid of ellipsoids to simplify the manipulation of the data, this grid emulates the image plane. As each neighbor pixel is transferred into the new surface, the ellipsoids are projected following the algorithm proposed in [20]. To achieve this, the image projection matrix is linearized at the center of the ellipsoid. Then, based on the dimensions of the ellipsoid, the coverage of cells is computed to determine whether a reference to the neighbor cell is placed or a new ellipsoid is created.

Since ellipsoids might cover more than one pixel, new instances are created and stored in the new gird. In consequence, the old instance is divided to contain the group of ellipsoids generated by the projection. Now $\mathbf{M}_j$ becomes $\mathbf{M}_G = \left\{\mathbf{M}_i^{(g)} | g = 1, \cdots, G\right\}$, as shown in Figure 3. This ensures that updates to the current surface affect each individual ellipsoid, while updates form a point of view that sees the group as one pixel will affect all the ellipsoids at once. When multiple ellipsoids fall in the same pixel, they are also grouped but only if they are close enough. If they are farther than a certain threshold, only the closest one is kept.

This maintains spatial relationship of adjacent pixels since the update of only visible pixels might derive in noisy surfaces.

Update: Once the new surface has all the neighbor ellipsoids that can be projected onto this plane, the surface assimilates the information coming from the current camera observations $\mathbf{I}_k, \mathbf{D}_k$ (see Figure 4). First, the position of the ellipsoids are updated pixel by pixel, if the cell is empty, a new ellipsoid will be created. The update of the position of the ellipsoid is done computing a weighted average of the pixel depth values along the camera axis. This simplify computations and ensure the respect of the projection model and the alignment of the ellipsoids of the map in the image plane. The current implementation updates the conic matrices with a weighted average in order to get a new matrix that can cover both old and new ellipsoids.

Is worth noting that, during the initialization of a new surface the depth-map $\mathbf{D}_k$ and the color image $\mathbf{I}_k$ are assimilated directly since they are observed in the pose that triggered the creation of the new surface. However, in the case of the update of an existing surface, the depth-map is used to create an intermediate surface $\mathbf{S}_k$ that will then be used to render a new set of $\mathbf{I}_{vk}$ and $\mathbf{D}_{vk}$ observed from the reference surface pose. In summary, during the update and initialization an operation of retro-splatting is used to project the new information into the grid, either to update the surface or to initialize it. After full initialization, the new surface $\mathbf{S}_{new}$ becomes member of the map nodes ($\mathbf{S}_w$).

6) Mapping criteria: One of the main objectives of this algorithm is to be able to retain as much as information as possible to enable the localization at proximity to surfaces. With this in consideration, the system includes a set of criteria to implement this policy (see Figure 1). Each condition is verified at the end of the alignment of a new key-frame. First, the graph will trigger the creation of a new surface when any distance from the last reference surface ($d_t(\mathbf{T}_w^k, \mathbf{T}_w^{n0})$ or

$d_R(\mathbf{T}_w^k, \mathbf{T}_w^{n0})$) exceeds a threshold configured by the user or when no reference surface is found:

$$d_t(\mathbf{T}_w^k, \mathbf{T}_w^{n0}) = \|\mathbf{W}_{\text{pos}} \cdot (\mathbf{t}_k - \mathbf{t}_{n0})\| \quad (5)$$

$$d_R(\mathbf{T}_w^k, \mathbf{T}_w^{n0}) = 2 \cdot \arccos(\mathbf{q}_k \cdot \mathbf{q}_{n0}) \quad (6)$$

Where $\mathbf{t}$ is the position component, $\mathbf{q}$ the orientation in the form of unitary quaternion and $\mathbf{W}_{\text{pos}}$ is a set of weights to adjust axis sensibility if needed. This metric is designed to ensure tracking continuity and decrease the accumulation of errors product of the projection pixel-discretization.

Secondly, there is the overlap criterion $\widetilde{\mathcal{O}}_{common}$:

$$\widetilde{\mathcal{O}}_{common} = \frac{|\mathcal{O}_{n0} \cap \mathcal{O}_k|}{|\mathcal{O}_{n0}|} \geq \tau \quad (7)$$

Here, $\mathcal{O}_k$ represents the occupied pixels in the grid of $\mathbf{S}_k$ and τ is an user defined threshold. This policy exits to ensure continuity of the mapping operation, specially during the first stages of the mission. Since there is few information stored in the map, the lack of overlap can cause the DVO module to fail, even if the distance traveled is not long. With this criterion, the graph is prone to detect when new zones are discovered to ensure sufficient conditions for the DVO module to work. At the same time the distance threshold can be set higher to reduce unnecessary node creation.

Then there is the resolution criterion $\widetilde{w}_{px}$:

$$\widetilde{w}_{px} = \frac{\frac{1}{J_{n0}} \sum_{j=1}^{w_{\text{img}} h_{\text{img}}} w_{px_j}^{(n0)}}{\frac{1}{J_k} \sum_{j=1}^{w_{\text{img}} h_{\text{img}}} w_{px_j}^{(k)}} \quad (8)$$

Here, $w_{px}^{(k)}$ represents the size of the pixel in milliliters when projected in the real world (see Figure 2) and J_{n0} represents the total count of pixels with valid depth at the surface $\mathbf{S}_{n0}$. As the agent evolves in the environment, existing data can often be observed in better conditions. Since the objective of the map is to enable precise localization at close-range, the graph manager prioritizes the creation of new nodes when the difference of resolution exceeds a threshold defined by the user. In the current implementation, a resolution improvement of 50% over the original resolution triggers the creation of a new node. The computation is similar to the computation of the ellipsoid shape. The mean pixel size of the incoming frame is compared to the mean pixel size of the closest reference surface as long as the overlap between them is greater than a user defined threshold. This ensures that the new node genuinely represents an improvement in resolution and overrides the distance criterion when necessary. In this special case, ellipsoids are overwritten with new information mixed from previous observations but prioritizing the storage of the new high resolution data to redefine existing data.

B. Performance

The current implementation of the mapping thread processes each frame in 355 seconds, with most of this time spent on creating and manipulating point instances. While most operations take few milliseconds, the creation of point instances and the application of geometric transformations alone account for 181 seconds of the total processing time. The remaining execution time is distributed between image rasterization (11 seconds) and the odometry implementation (30 seconds), both of which are slowed by the creation of intermediate surface structures. Odometry time includes not only the operations of the DVO module but also pose estimation iterations, virtual surface creation. Each iteration also requires a rasterization call, further contributing to the processing time. While similar operations on arrays of data points can be up to 30 times faster, the need to maintain consistent point references across different surfaces limits the use of certain Python optimization tools.

As SLAM systems require an execution time close to the real time for deployment in robotic applications, this algorithm still needs to improve its execution time to be able to be deployed in a real drone. However, this execution time can be greatly improved when implemented in proper conditions. The current version has been written in Python 3, data manipulation is done with the numpy library accelerated with the help of the numba library in some operations.

A significant limitation of the prototype comes from implementation language. The first issue is that Python usually executes a single tread process which is not adequate for the application. While threading is possible at big scale (i. e., running tracking and mapping in parallel), many potentially parallelizable tasks remain executed sequentially. The majority of the operations are constrained to a single thread due to numba's compatibility only with native numpy objects and a limited subset of operators. Furthermore, at the beginning of the call, the interpreter still has to execute the numba overhead to use the compiled executable of the code. Although this overhead is shorter than running un-optimized Python code, it remains slower than a fully multi-threaded implementation in a compiled language.

Another limitation derived from the choice of the programming language is the memory management. To avoid data redundancy, each ellipsoid is modeled as an object instance that contains all the information to describe the geometric entity. When creating a surface, hundreds of thousands of instances have to be created at each node and this operation consumes more time and memory than equivalent operations in C or C++. For instance, with a resolution of 1280x720 pixels, the map manager takes approximately 60 seconds to create a surface and initialize it with the information of its neighbors (dependent on the number of neighbors). This process cannot be parallelized with numba since this library does not work with Python object instances. By contrast, implementing this operation in C or C++ would allow faster indexing by memory allocation and faster data processing with parallelized processing.

Finally, the implementation of the current version does not use any graphic computation resources. This means that operations like the render of the depth and color images could be greatly accelerated for both tracking and mapping processes. This omission contributes further to the system's suboptimal

performance in its current form. However, with targeted optimizations in language, threading, memory management, and GPU utilization, the proposed algorithm shows strong promise for real-time deployment in robotic systems.

C. Localization

In the next step, the drone is assumed to use the information of the dense map generated from the first flight and perform a scanning flight with the measurement payload. This payload includes any specialized sensing system required for the inspection and a localization camera, which does not need to be stereo.

1) Dense visual odometry (DVO): The alignment of the key-frames and the precise tracking of the system depends on the DVO module [21], provided by the OpenRox library. This module uses a reference image and a depth-map to compute the pose between the reference data and another input image. To integrate this module into the current implementation, a Cython wrapper is used to call its functions, as the module itself is developed in C.

In broad terms, the module minimizes the photometric error function between the current image and the warped reference image until it reaches a threshold. The module uses the zero mean cross-correlation index (ZNCC) to evaluate the quality of the alignment between the warped image and the input image. That said, both tracking and mapping threads reject the estimation if this index falls bellow the 70%. Such a low correlation is interpreted as insufficient similarity between the images, indicating that the estimation may be unreliable for accurate localization.

As mentioned earlier, the module is accessed trough a Python warper of the OpenROX library, thus, no splatting technique during the warping operation. Considering that the warping operation often

2) Pixel splatting: To feed the DVO module either for tracking or for key-frame alignment, the map must be rendered into a virtual image plane. In both cases, the map is projected into the image plane of a virtual camera placed at the last estimated pose using the method of Elliptical Weighted Average (EWA) described in [20]. With this method each j point in the point-cloud has to be associated with a shape matrix representing an ellipsoid $\mathbf{M}_j$. First, the camera projection model is linearized around the 3D coordinates of the ellipsoid center $\mathbf{u}_j$. As indicated in the work of Zwicker et al., the linearized projection model of a pinhole camera around a 3D point $\mathbf{u}_j$ can be obtained with the following Jacobian matrix:

$$\mathbf{J}_j = \begin{pmatrix} \frac{f_{px}}{(\mathbf{u}_j)_z} & 0 & -\frac{f_{px}(\mathbf{u}_j)_x}{(\mathbf{u}_j)_z^2} \\ 0 & \frac{f_{px}}{(\mathbf{u}_j)_z} & -\frac{f_{px}(\mathbf{u}_j)_y}{(\mathbf{u}_j)_z^2} \\ \frac{(\mathbf{u}_j)_x}{|\mathbf{u}_j|} & \frac{(\mathbf{u}_j)_y}{|\mathbf{u}_j|} & \frac{(\mathbf{u}_j)_z}{|\mathbf{u}_j|} \end{pmatrix} \tag{9}$$

where f_{px} is the focal distance of the camera in pixels and $(\mathbf{u}_j)_z$ is the z component of the 3D point $\mathbf{u}_j$. This version has been modified from the original text to include the camera intrinsic parameters. Using this matrix, the projected 3D shape matrix $\mathbf{V}_{3D_k}$ can be obtained with the following equation:

$$\mathbf{V}_{3D_k} = \mathbf{J}_k \mathbf{R} \mathbf{M}_k \mathbf{R}^T \mathbf{J}_k^T \tag{10}$$

Here, $\mathbf{R}$ is the rotation matrix in the viewing transformation, which brings the points into the camera frame. A 2×2 subset of the transformed matrix is used to represent the projected 2D ellipse in the image plane. The resulting matrix can now be used to obtain the size of the ellipsoid in the image plane thus the size of the splatting kernel.

The splatting operation is performed at a zone defined by the size of the kernel and the aggregation of color/depth is done using the radial distance metric to ponder the addition. For a point in the kernel $\mathbf{x}_i$, which lies in the kernel of $\mathbf{M}_j$, the radial distance from the ellipsoid center $\mathbf{x}_0$, can be expressed as:

$$r_i(\widetilde{\mathbf{x}}) = \widetilde{\mathbf{x}}_i^T \mathbf{Q} \widetilde{\mathbf{x}}_i \quad \text{where} \quad \widetilde{\mathbf{x}}_i = \mathbf{x}_0 - \mathbf{x}_i \tag{11}$$

Here, $\mathbf{Q}_j$ is the conic matrix, inverse of the shape matrix $\mathbf{M}_j$, which defines the geometry of the ellipsoid in the image plane. To save some computation time, the weights for the aggregation are precomputed and stored into a lookup table w_{tab}. To align the radial distance to the indexes in the lookup table, $\mathbf{Q}_j$ is scaled to match the size of w_{tab}. If $r(\widetilde{\mathbf{x}})$ is bigger than 1 this means that the point is outside the ellipsoid thus does not have to be considered.

In the original code proposed by Zwicker et al. this weights come from the Gaussian distribution expression. However, in this paper it was changed for an exponential decay so the overlap between two close kernels wont generate any blur, particularly at the centers of ellipsoids.

Pixel occlusion When a kernel pixel falls inside the ellipse, the corresponding weight in w_{tab} to determine the weight of the color/depth value in the aggregation. That said, additional considerations have to be observed given the particular case of close-range inspections. Due to the proximity to the surface and the geometry of the scene, ellipsoids that should be occluded may incorrectly change the image if they fall in unoccupied areas of the grid. To deal with the occlusion of ellipses two new weights are added to the traditional weighted average, the occluded term (o_{ded}) and the occluding term (o_{ing}).

$$o_{ded} = \min\left(1, \exp\left(-\frac{1}{2}\frac{d_{old} - d_{new}}{\mathbf{V}_{zz}}\right)\right) \tag{12}$$

$$o_{ing} = \min\left(1, \exp\left(-\frac{1}{2}\frac{d_{new} - d_{old}}{\mathbf{V}_{zz}}\right)\right) \tag{13}$$

Where $\mathbf{V}_{zz}$ is the z component in the shape matrix of the ellipsoid and indicates the sensibility of the decay for this coefficient, the bigger the ellipsoid, slower the occluded value will fade. Inversely, the occluding term will gain in weight as the new point places in front of the old one. Both coefficients are limited to 1 to avoid the saturation of the values as the only interest of these terms is to ponder information weights in the case of occlusion. Once these terms are computed, the

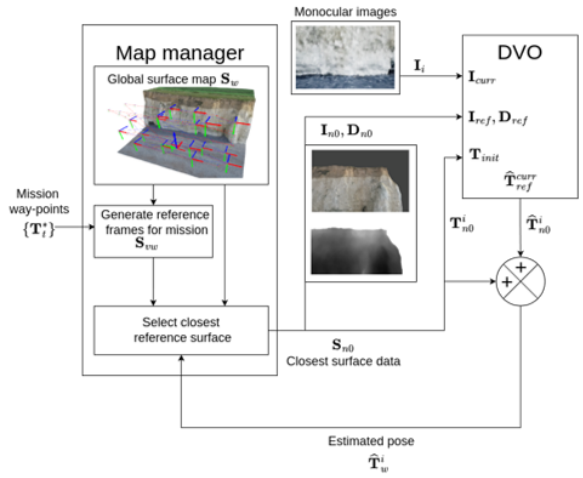

Figure 6. After the mapping flight, a collection of reference surfaces $\mathbf{S}_w$ will be available. Then, according to the trajectory of the flight, a set of specific frames $\mathbf{S}_{vw}$ is also rendered to save computation time around expected poses.

update of the j pixels in the image can be done by an iterative weighted average:

$$\mathbf{c}_j = \frac{o_{ing} w_i \mathbf{c}_i + o_{ded} \mathbf{c}_j}{o_{ing} w_i + o_{ded} w_j} \quad (14)$$

$$\text{where } w_{vj} = w_{tab}(r_i(\widetilde{\mathbf{x}}_i))) \quad (15)$$

Here, pixel j of the Image grid is updated with the values of the kernel pixel i whose weight is defined by its distance to the center of the ellipse. This approach ensures that both occluded and occluding points are accounted for appropriately, preserving the spatial accuracy of the rendered image. A similar aggregation function is applied for the computation of the depth image.

3) Scan workflow: After the mapping flight, the collection of reference surfaces can be exploited for subsequent flights even with reduced optical capacities (monocular instead stereo camera). In *free mode*, each reference surface is rendered at its position to save computation time. Alternatively, if the user decides that the drone has to follow a particular trajectory, a set of mission way-points $\{\mathbf{T}^{t*}_w | t = 1, \cdots, T\}$ must be fed to the map manager (see Figure 6 at the left). Using these points as reference, the map manager can generate a set of virtual positions that ensure the convergence of the DVO module when the agent moves along the vicinity of the desired trajectory. The current implementation takes the shortest path between each pair of way-points and then samples a set of intermediate positions based on a overlap criterion. To do this, the virtual positions are at a user-defined certain distance behind the trajectory to ensure visibility. Their orientation is assumed to be perpendicular to the target trajectory.

The overlap is computed by projecting the path between two way-points and selecting the segment of the path within in the borders of the virtual camera. The next camera center is selected in function of the portion of the projected line covered by the virtual camera and the overlap criterion. The camera FOV overlap is computed assuming the next virtual frame will cover the same portion of the line. This collection of virtual frame poses is also included in the global map as a simplified key-frame, composed only by the color and depth information $\mathbf{I}_{vk}$ and $\mathbf{D}_{vk}$.

Figure 7. Example of the images seen by the simulated drone during the scan flight. As shown in this figure, there are little to no strong details that can help to formulate a correct pose estimation.

During the flight, the map manager continuously selects the closest reference data, key-frame or surface and performs dense odometry relative to the reference key-frame $\mathbf{S}_{vk}$. The DVO module is now initialized with the estimated pose between the reference frame and the last pose estimation $\widehat{\mathbf{T}}^i_w$. If the control data of the agent is available, a preparatory dead-reckoning (DR) step is performed to decrease convergence time (performed in the feedback loop in Figure 6). The current actual implementation the DR step is only for initialization purposes, its implementation into a filtered estimator, such as a Kalman or Particle filter, could provide smoother pose estimations and correctly model the its uncertainty.

IV. Results

A. Simulation data

The performance of the system was first tested in simulations with realistic data collected at the Sainte-Marguerite-sur-Mer cliff (Normandy), monitored in the framework of the Defhy3geo project [22]. The scene is composed of a segment of a cliff model reconstructed using the Agisoft metashape software. The simulation environment was built from the 3D model of the cliff generated form aerial geo-referenced images. A mapping flight was carried on the field, with geo-referenced targets placed to ensure a model alignment accuracy at centimeter-level. The visible area covers a 60x20 meter section of the cliff with a non-structured texture (see Figure 7), providing a challenging environment for texture-based localization.

This test demonstrates the importance of map consistency, particularly its role in maintaining reliable localization results. If the mapping module fails to capture environmental details accurately, an inaccurate initial pose estimate can cause the DVO module to deviate significantly from the correct solution. Despite measures to handle local minima, the lack of sharp geometric features slows pose convergence, potentially reducing precision over time. However, the use of a dense

Figure 8. The video used to test the system presents challenges on multiple levels. First, the lack of significant gradient variations tests the DVO module's performance under extreme conditions. Additionally, the geometry of the environment makes difficult to locate based only in the shape of the surface.

map and realistic rendering in DRONE-SLAM ensures stable performance even under these challenging conditions.

B. Real data

The algorithm was evaluated using a set of real-world data to assess its performance under practical conditions. The test scenario covers a low-texture facade of a building at the Cerema Normandie-Centre institute, providing a challenging environment for visual tracking. Since the drone could not be equipped with a wide-baseline stereo camera, the map was generated using a photometric model built with aerial data from a drone Mavic 3E. This procedure replaces the key-frame selection step of the mapping workflow, providing an opportunity to evaluate the system's ability to exploit other sources of information as reference. Drone operators frequently generate 3D models using similar tools for purposes such as visualization, inspection, or environmental modeling, consistent with the methods applied in this experiment. This test demonstrates DRONE-SLAM's ability to effectively utilize pre-generated models, enabling rapid deployment even on drones with limited payload capacity.

The real dataset was not evaluated with ORB-SLAM as the data collected was not sufficient for a fair comparison. The Mavic 3E cannot be equipped with a wide-baseline camera essential for long-range depth estimation during mapping,. In consequence, the map generated in mapping mode was insufficient to build a significant map, specially when the interest regions lack of details.

As outlined in the proposed workflow, the scanning trajectory and reference data were generated to provide inputs to the tracking thread for pose estimation during the scanning mission. The tracking test used a video recorded by the drone's default wide-lens monocular camera, flying at a distance of 2.5 meters from the wall. The test surface was a plain white wall recorded on a cloudy day (see Figure 8).

The software environment is composed of the following elements:

- Airsim as the simulation environment [23]
- ROS for communication between different software components.
- A Python-based offline registration node for map generation.
- A direct odometry algorithm [21] implemented the OpenROX library (developed by the ACENTAURI team).
- A mission control module to load the map, generate the virtual frames and control the drone.

C. Benchmark conditions

As done in [1], ORB-SLAM3[13] is refereed as comparison method. The principal argument to compare with this method is that this system is relatively popular in robotics since is fast and precise enough for mobile robots deployment. For each scenario, we perform almost the same operation with both ORB3 and DRONE. First a mapping flight is performed with a stereo camera in full SLAM mode, far from the structure. A second flight is performed using the generated map from the first session. In this step ORB3 is configured to run in localization mode, which implies that no new information will be added to the map. That said, is worth noting that the scanning flight with ORB3 is performed still with the stereo camera since with the monocular camera the system struggles to converge. For comparison we take the closest flight to the surface where ORB3 is able to perform the whole flight. The test allows ORB3 to get lost but stops when the error gets above 5 meters without recovery.

D. Experimental results

As presented in [1], the system's performance is evaluated across three aspects: mapping precision, rendered image quality, and localization accuracy. This organization is retained to clearly illustrate the improvements introduced by this method

1) Mapping precision: The first aspect is computed as the distance of the center of the ellipsoids in the map to the reference model. For this we computed a KD-tree to obtain the distances of each surface to the model. To avoid an unfair comparison, points in the ground plane were ignored so the distance represents only the quality of the structure reconstruction. To better appreciate the quality of the map, the distance is expressed in two formats: the quantity of ellipsoid centers that fall into three constraints of precision and the mean distance error of all the points.

TABLE I. MAPPING PRECISION METRICS.

Difficulty	**Lab**	**Cliff**
Hard ($<$ 10 cm)	89.33 %	49.04 %
Medium ($<$ 20 cm)	10.19 %	37.10 %
Light ($<$ 30 cm)	0.46 %	0.13 %
Mean Error Distance	5.17 cm	12.40 cm

2) Rendering: DRONE-SLAM implements improvements to the rasterization method presented in [1], focusing on enhanced rendering accuracy and noise reduction. In [1], a parameter was introduced to define the shape of the weight table, addressing the overlap of Gaussian distributions. In this work, the ellipsoid formulation has been redefined to provide a more accurate representation of the environment. The new implementation uses a splatting method for surface initialization and updates, effectively reducing noise in ellipsoid positioning, enabling a

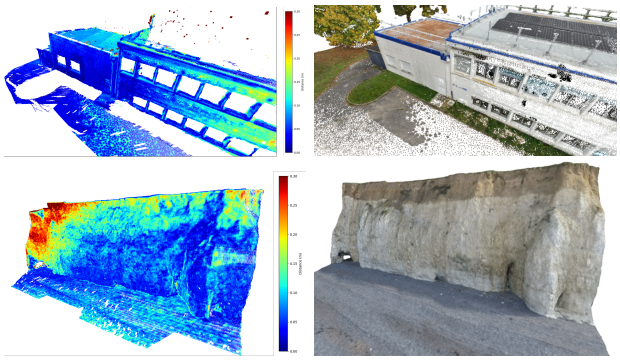

Figure 9. 3D renderings of maps generated by the mapping module (left) with corresponding ground truth models (right). Point colors in the generated maps indicate the distance from the ground truth models. The reference models were generated using geo-referenced images captured during a photometric data-acquisition flight with an average accuracy of approximately 5cm, derived from the GPS-RTK accuracy.

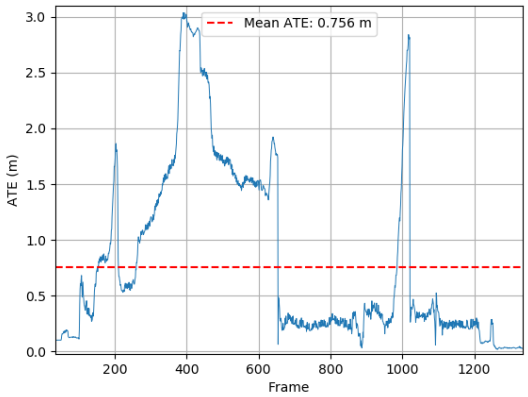

Figure 10. ORB-SLAM3 absolute translation error (ATE) in meters during the scan path with stereo camera. Following the 2 step workflow, ORB-SLAM3 manages to approach up to 5 m close to the cliff surface.

smoother and more consistent data integration. As a result, the mapping method more effectively captures subtle surface details and improves the alignment of data clouds. DRONE-SLAM continues to address the challenges of reduced and uneven point density during close-range observations. However, the need for a custom weight distribution to avoid blurring has been significantly reduced, thanks to improvements in point cloud merging strategies.

3) Localization: simulations: The system's performance is evaluated based on its localization precision during close-range flights to a structure. For this, DRONE-SLAM is compared to ORB-SLAM3 as the baseline in the first simulated scenario. During the scan flight, ORB-SLAM3 initially shows an increase in localization error as the drone approaches the cliff (Figure 10). The system nearly loses tracking capability in the proximity phase. However, as the images contain more informative regions, ORB-SLAM3 recovers. Almost at the end of the flight, a new error peak appears, consequence of a low texture region with oblique geometry that is quickly occluded as the drone follows its path.

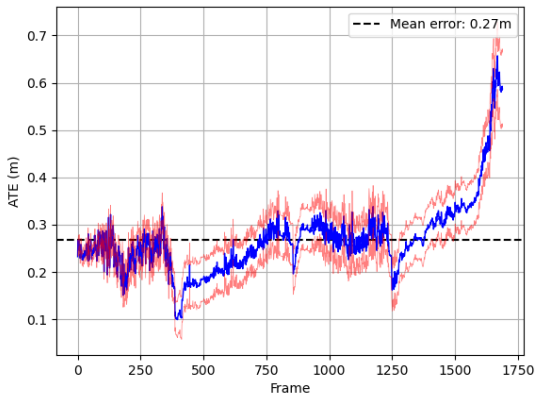

Figure 11. DRONE-SLAM absolute translation error (ATE) in meters during the scan path with monocular camera. The red lines represent the standard deviation cumulated over the time as a from to show the precision of the system. This flight was performed 2 meters away from the cliff.

In contrast, we observe a stable behavior from DRONE-SLAM. The localization error maintains over the different regions of the map. The peak of the end is due to a poor performance of the masked SSIM metric to trigger new nodes when small but relevant regions are discovered. Compared to the performance presented in [1], DRONE SLAM shows more precision since the variations of its estimations remains coherent with the movement of the drone during the scan. Moreover, the peak of error that affected both methods is no longer observed in the case of DRONE (see Figure 11).

4) Localization: real data: Real data tests confirmed that increased resolution, achieved by generating the photometric model at a lower altitude than in the cliff scenario, enhances system performance. In this scenario, the performance comments will be done wrt to the GPS-RTK data synchronized with the video images. As seen at the top of the Figure 12, the pose estimation closely follows the GPS-RTK position estimation. A continuous offset is observed, potentially caused by delays in GPS updates. This observation emerges from the shape of the GPS path that sometimes seems slightly off with respect to the movement observed in the camera. Moreover, the error metric of the odometry seems to increase when these abrupt changes occur. That said, further study on the test conditions has to be done but for the moment, the GPS RTK will be considered as the groundthruth data.

The ATE metric shows a mean error of 17 cm (Figure 13), with stable performance even in low-texture regions. The system performs slightly better during lateral movements compared to vertical movements. As stated before, the lack of geometry features contributes to the ambiguity of the solutions, thus undermining precision. Another factor affecting precision is the use of masks for reference data. Since the reference data consists only of valid map renderings, background elements are often masked and ignored, reducing available visual cues for localization. As a result, the DVO module may observe only planar regions with minimal texture variation, such as

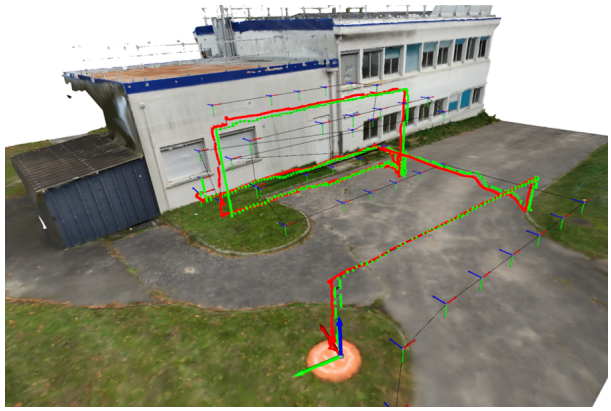

Figure 12. Path followed during the lab flight. GPS-RTK positions are showed in green and DRONE estimations in red. The drone first made a flight at 4 meters from the wall and then another one at around 2m from the wall. The second part of the flight was not completed for KF graph issues.

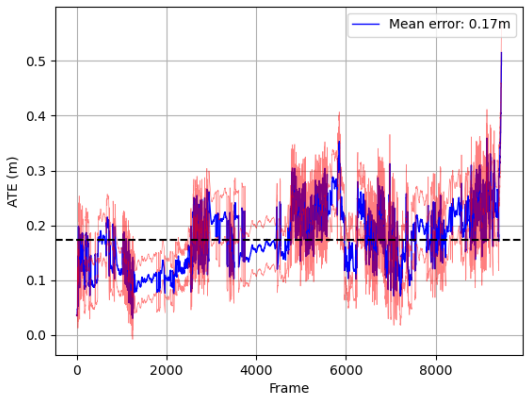

Figure 13. DRONE-SLAM ATE in meters during wall inspection. The system showed better behavior in this structured scenario. The higher performance of the depth estimation module seems to improve the capacity of the mapping module to register subtile details in the texture, which is complicate to perform with relatively noisy estimations like StereoBM.

a uniform wall surface, instead of building edges. These conditions demand high precision to capture subtle texture changes; otherwise, the risk of non-convergence significantly increases.

The system's accuracy heavily depends on the availability of virtual keyframes during the scan step, as these provide essential reference points for pose estimation. This is because the DVO module can diverge if the distance between the reference and the input image are too far. To address this, the system is designed to generate new reference frames when needed, but the mission planner must trigger keyframe generation opportunely. At the end of the sequence, during the flight at 2 meters from the wall, the system failed due to an inadequate keyframe generation strategy in previously unexplored regions. The system performs smoothly in regions where the path planner has pre-generated virtual keyframes, but its performance degrades in unplanned areas. The current implementation relies on inter-keyframe distance criteria to mitigate these issues. However, there are some cases that need a more robust keyframe generation criterion to ensure stability.

V. Conclusion

This paper presented a custom mapping and localization framework, designed for close-range inspection scenarios. The evaluation metrics show that DRONE-SLAM can successfully generate a high resolution dense model of the environment tailored for this use case. Thanks to this dense map and the rendering technique, the system can achieve accurate localization with simple equipment such as monocular cameras. Additionally, the rendering method compensates the perspective narrowing problems while preserving robustness face to low-texture data.

The system's performance was measured across three aspects: First, the quality of the map reconstruction. Secondly, the quality of the custom render method and its compatibility with the DVO module. Finally, the localization accuracy in different scenarios.

While the performance of the system shows an improvement in the quality of estimation in close-range scenarios, some challenges remain. First, the framework must achieve real-time execution to meet the practical requirements of field deployment. Analysis of the execution time profile reveals that the main issues lie in the implementation language and its lack of direct memory access.

Second, the mapping method can improve its robustness in outdoors scenarios. While naive color fusion may work under optimal conditions, the system must reliably handle varying weather. Ongoing work focuses on surface luminance mapping to enable localization with non-uniform lighting. Likewise, this can also improve the multi-session performance of the system as it will allow the use of the map in different conditions and even simulate specific lighting scenarios.

Finally, further research is needed to improve the criteria for generating virtual reference frames. Progress in this aspect will both enhance system stability and extend the utility of dense maps for navigation and mission planning. Simulations show that DRONE-SLAM is more accurate at proximity to the surface with reduced equipment. The test with realistic data demonstrate that this performance is maintained even in extreme low texture conditions. In summary, DRONE-SLAM delivers pose estimation accuracy comparable to GPS-RTK without its inherent limitations, such as dependency on signal availability and sensibility to obstacles.

References

[1] D. Navarro Tellez, E. Malis, R. Antoine, and P. Martinet, "Hybrid visual slam for multi-session precise localization: Application to a coastal cliff in Normandy", in *The Twenty-First International Conference on Autonomic and Autonomous Systems*, Mar. 2025, pp. 35–40, ISBN: 978-1-68558-241-8.

[2] D. Chabot, "Trends in drone research and applications as the journal of unmanned vehicle systems turns five", *Journal of Unmanned Vehicle Systems*, vol. 6, no. 1, pp. vi–xv, 2018.

[3] J. Fan and M. A. Saadeghvaziri, "Applications of drones in infrastructures: Challenges and opportunities", *International Journal of Mechanical and Mechatronics Engineering*, vol. 13, no. 10, pp. 649–655, 2019.

[4] R. Antoine *et al.*, "Geoscientists in the sky: Unmanned aerial vehicles responding to geohazards", *Surveys in Geophysics*, vol. 41, no. 6, pp. 1285–1321, 2020.

[5] A. Gupta and X. Fernando, "Simultaneous localization and mapping (slam) and data fusion in unmanned aerial vehicles: Recent advances and challenges", *Drones*, vol. 6, no. 4, p. 85, 2022.

[6] S. Halder and K. Afsari, "Robots in inspection and monitoring of buildings and infrastructure: A systematic review", *Applied Sciences*, vol. 13, no. 4, p. 2304, Jan. 2023, Number: 4 Publisher: Multidisciplinary Digital Publishing Institute, ISSN: 2076-3417. DOI: 10.3390/app13042304.

[7] Y. Ham, K. K. Han, J. J. Lin, and M. Golparvar-Fard, "Visual monitoring of civil infrastructure systems via camera-equipped Unmanned Aerial Vehicles (UAVs): A review of related works", *Visualization in Engineering*, vol. 4, no. 1, p. 1, Jan. 2016, ISSN: 2213-7459. DOI: 10.1186/s40327-015-0029-z.

[8] J. Jia and Y. Li, "Deep Learning for Structural Health Monitoring: Data, Algorithms, Applications, Challenges, and Trends", *Sensors*, vol. 23, no. 21, p. 8824, Jan. 2023, Publisher: Multidisciplinary Digital Publishing Institute, ISSN: 1424-8220. DOI: 10.3390/s23218824.

[9] G. Esposito, A. Salari, I. Catapano, D. Erricolo, and F. Soldovieri, "UAV-based GPR prototype for structural monitoring of bridges: Preliminary results and perspectives", *ce/papers*, vol. 6, no. 5, pp. 930–933, 2023, _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/cepa.2074, ISSN: 2509-7075. DOI: 10.1002/cepa.2074.

[10] A. Massaro *et al.*, "Thermal IR and GPR UAV and Vehicle Embedded Sensor Non-Invasive Systems for Road and Bridge Inspections", in *2021 IEEE International Workshop on Metrology for Industry 4.0 & IoT (MetroInd4.0&IoT)*, Jun. 2021, pp. 248–253. DOI: 10.1109/MetroInd4.0IoT51437.2021.9488483.

[11] A. Couturier and M. A. Akhloufi, "A review on absolute visual localization for UAV", *Robotics and Autonomous Systems*, vol. 135, p. 103 666, Jan. 2021, ISSN: 09218890. DOI: 10.1016/j.robot.2020.103666.

[12] R. Mur-Artal and J. D. Tardós, "ORB-SLAM2: An Open-Source SLAM System for Monocular, Stereo, and RGB-D Cameras", *IEEE Transactions on Robotics*, vol. 33, no. 5, pp. 1255–1262, Oct. 2016. DOI: 10.1109/tro.2017.2705103.

[13] C. Campos, R. Elvira, J. J. G. Rodríguez, J. M. M. Montiel, and J. D. Tardós, "ORB-SLAM3: An Accurate Open-Source Library for Visual, Visual–Inertial, and Multimap SLAM", *IEEE Transactions on Robotics*, vol. 37, no. 6, pp. 1874–1890, Dec. 2021, IEEE Transactions on Robotics, ISSN: 1941-0468. DOI: 10.1109/TRO.2021.3075644.

[14] R. A. Newcombe, S. J. Lovegrove, and A. J. Davison, "DTAM: Dense tracking and mapping in real-time", in *2011 International Conference on Computer Vision*, Nov. 2011, pp. 2320–2327. DOI: 10.1109/ICCV.2011.6126513.

[15] J. Engel, T. Schöps, and D. Cremers, "LSD-SLAM: Large-Scale Direct Monocular SLAM", in *Computer Vision – ECCV 2014*, D. Fleet, T. Pajdla, B. Schiele, and T. Tuytelaars, Eds., ser. Lecture Notes in Computer Science, Cham: Springer International Publishing, 2014, pp. 834–849, ISBN: 978-3-319-10605-2. DOI: 10.1007/978-3-319-10605-2_54.

[16] B. Zhang and D. Zhu, "A Stereo SLAM System With Dense Mapping", *IEEE Access*, vol. 9, pp. 151 888–151 896, 2021, ISSN: 2169-3536. DOI: 10.1109/ACCESS.2021.3126837.

[17] B. Kerbl, G. Kopanas, T. Leimkühler, and G. Drettakis, "3d gaussian splatting for real-time radiance field rendering", *ACM Transactions on Graphics (SIGGRAPH Conference Proceedings)*, vol. 42, no. 4, Jul. 2023.

[18] R. A. Hamzah, R. A. Rahim, and Z. M. Noh, "Sum of Absolute Differences algorithm in stereo correspondence problem for stereo matching in computer vision application", in *2010 3rd International Conference on Computer Science and Information Technology*, vol. 1, Jul. 2010, pp. 652–657. DOI: 10.1109/ICCSIT.2010.5565062.

[19] J. Li *et al.*, "Practical stereo matching via cascaded recurrent network with adaptive correlation", in *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, ISSN: 2575-7075, Jun. 2022, pp. 16 242–16 251. DOI: 10.1109/CVPR52688.2022.01578.

[20] P. S. Heckbert, "Fundamentals of Texture Mapping and Image Warping", University of California at Berkeley, USA, Technical Report, 1989.

[21] A. I. Comport, E. Malis, and P. Rives, "Real-time Quadrifocal Visual Odometry", *The International Journal of Robotics Research*, vol. 29, no. 2, pp. 245–266, 2010. DOI: 10.1177/0278364909356601.

[22] T. Junique *et al.*, "Investigation of the geological and hydrogeological structure of chalk cliffs with visible, thermal infrared and electrical resistivity imaging", *Journal of Hydrology*, vol. 630, p. 130 642, 2024.

[23] S. Shah, D. Dey, C. Lovett, and A. Kapoor, "Airsim: High-fidelity visual and physical simulation for autonomous vehicles", in *Field and Service Robotics*, 2017. eprint: arXiv:1705.05065.