Short-term UAV Path-Planning with Monocular-Inertial SLAM in the Loop

Conference Paper

Author(s):
Alzugaray, Ignacio; Teixeira, Lucas; Chli, Margarita

Publication date:
2017-07-21

Permanent link:
https://doi.org/10.3929/ethz-a-010862339

Rights / license:
In Copyright - Non-Commercial Use Permitted

Originally published in:
https://doi.org/10.1109/ICRA.2017.7989319

Funding acknowledgement:
644128 - Collaborative Aerial Robotic Workers (SBFI)

Short-term UAV Path-Planning with Monocular-Inertial SLAM in the Loop

Ignacio Alzugaray, Lucas Teixeira and Margarita Chli
Vision for Robotics Lab, ETH Zurich, Switzerland

***Abstract*— Small Unmanned Aerial Vehicles (UAVs) are some of the most promising robotic platforms in a variety of applications due to their high mobility. Their restricted computational and payload capabilities, however, translate into significant challenges in automating their navigation. With Simultaneous Localization And Mapping (SLAM) systems recently demonstrated to be employable onboard UAVs, the focus fall on path-planning on the quest of achieving autonomous navigation. With the vast body of path-planning literature often assuming perfect maps or maps known *a priori*, the biggest challenge lies in dealing with the robustness and accuracy limitations of onboard SLAM in real missions. In this spirit, this paper proposes a path-planning algorithm designed to work in the loop of the SLAM estimation of a monocular-inertial system. This point-to-point planner is demonstrated to navigate in an unknown environment using the incrementally generated SLAM map, while dictating the navigation strategy for preferable acquisition of sensor data for better estimations within SLAM. A thorough evaluation testbed of both simulated and real data is presented, demonstrating the robustness of the proposed pipeline against the state-of-the-art and its dramatically lower computational complexity, revealing its suitability to UAV navigation.**

Video—`https://youtu.be/Izn_vVb_M-E`

I. INTRODUCTION

With the potential of revolutionizing tasks, such as search-and-rescue, 3D modeling and industrial inspection, small Unmanned Aerial Vehicles (UAVs) have recently drawn great attention from both academia and industry. Multi-rotor UAVs are particularly interesting due to their high mobility and light weight, promising swift access to remote areas without requiring any particular infrastructure (e.g. paved roads). Nonetheless, their properties come at the cost of limited payload, restricting the number, total weight and power consumption of the sensors and processors that can be carried onboard and thus, limiting the onboard computational capacity and sensing capabilities. Some of the most promising sensors for UAVs for these robotics platforms are cameras due to their relatively low weight, small size and low cost for the rich information that they provide about their surroundings.

With seminal advancements in vision-based Simultaneous Localization and Mapping (SLAM) techniques (e.g. [1]) demonstrating that ego-motion scene estimation are possible using visual cues, their employment onboard UAVs (e.g. [2]) has been key in the research towards automating UAV navigation. With path-planning constituting the last piece of the puzzle in completing the loop of robot navigation, the community has recently been turning to the vast body of theoretical path-planning works to devise computationally feasible approaches, albeit, traditionally assuming the availability of a known or pre-acquired map. While some powerful systems have emerged, it is the restricting computational complexity onboard a UAV and the perceptual uncertainties arising in a real mission that pose the greatest challenges prohibiting the practical employment of these methods in reality. Exploiting sensing cues processed in a SLAM system in real-time for the purpose of developing a feasible and preferable navigation strategy, as advocated in this paper, opens up a great envelope of potential benefits in both robustifying UAV navigation as well as pushing the platform's capabilities towards greater navigation autonomy.

In this paper, we employ a nominal keyframe-based SLAM system that fuses sensing cues from a monocular camera and an Inertial Measurement Unit (IMU) featuring onboard the UAV's sensor-suite. Since the SLAM map is generated incrementally throughout the flight, we develop a path-planner that can handle changes in the plan on the fly as new unexplored areas of the environment are discovered. Exhibiting such replanning capabilities is particularly important

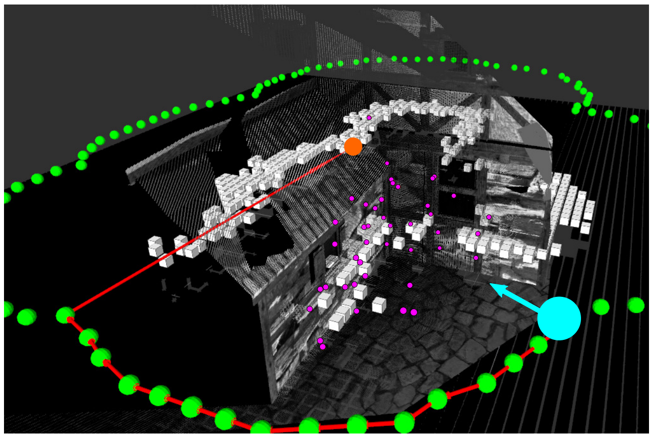


Fig. 1. The proposed pipeline on the task of navigating a UAV around a building. The map is estimated on the fly using monocular-inertial SLAM resulting to sparse 3D points (in magenta), in contrast to the dense grey point cloud on the building's facades, which would be obtained from a depth sensor. The SLAM map provides information about obstacles (white boxes), which is used to generate potential UAV pose samples (green spheres). The planner computes a path (red line) through these samples from the current pose (cyan sphere) to an arbitrary goal (in orange, here set inside the building). The obtained path is collision-free according to the current map and, in our approach, it can be efficiently re-planned in cases of map changes as new areas are explored.

This research was supported by the Swiss National Science Foundation (SNSF, Agreement no. PP00P2_157585) and EC's Horizon 2020 Programme under grant agreement n. 644128 (AEROWORKS).

in real missions, as one can never assume that a even a pre-acquired map of the environment will not change. As a result, this property of the proposed system is key in ensuring the applicability and the robustness of this framework. While the proposed approach is agnostic to the particular Monocular-Inertial SLAM (MIS) system used, in this paper, we present experiments using the open source[1] keyframe-based SLAM system of [3] and [4]. A snapshot of the proposed pipeline in action is visible in Fig. 1.

Traditionally, SLAM approaches rely on tracking features across the robot's trajectory, resulting in the construction sparse SLAM maps of the robot's workspace. When using a single camera as the only exteroceptive sensor, this SLAM map is considerably noisier when using additional sensors, such as stereo cameras, as the depth estimation of features becomes particularly sensitive to the parallax across consecutive images. As a result, the trajectory of the camera, as well as the distribution of these features in space (e.g. to sufficiently cover an obstacle) have a great impact on the fidelity of the acquired SLAM map. While these factors pose great challenges in both SLAM and thus, in any path-planner that aims to use the SLAM map in real-time, the proposed approach explicitly exploits potentially uneven distribution of features around obstacles to guide the path-planning decisions, while favouring beneficial camera motions for robust SLAM performance.

A. *UAV Path-Planning for real missions*

One of the most important aspects of planning without a map known *a priori*, and thus planning on a map that is constructed on the fly, is to consider a plan to follow a point-to-point path may become outdated as new areas of the environment are being explored and new obstacles are discovered. Therefore, for an algorithm aiming to incorporate such a map providing only partial knowledge of the immediate surroundings of the UAV, the ability to re-plan the UAV's path according to map changes within a reasonable amount of time (onboard CPU present on small UAVs, such as the AscTec Neo[2]), is imperative. One of the most used path-planning algorithms with re-planning capabilities is ADA* [5], successfully employed for UAV navigation in [6]. As ADA* relies on densely connected graphs, usually generated by a cell-grid decomposition of the map, its scalability is severely limited by the considered map size and the grid resolution.

Alternative to grid based methods are random sampling approaches, which have become increasingly popular in the Robotics community, especially since the development of Rapidly-exploring Random Trees (RRT) [7]. The recently developed RRTX approach [8], allows the planned path to be adapted to changes in the map, while guaranteeing asymptomatic optimality with respect to a cost function, as it is based on the successful RRT* [9] method. The quick re-planning capability of this algorithm is mainly achieved by keeping record of alternative connections between different samples of the tree, which are used in case a newly perceived obstacle triggers the modification of the planned path. One major drawback of this approach in the scenario targeted in this paper, is that the tree generated by RRTX is rooted to a fixed goal position, meaning that if the mission goal is modified during the execution of the mission it is necessary to re-run the complete pipeline from scratch. Additionally, although it might be interesting to embed some valuable information for the MIS in the cost function optimized by the RRTX, the inclusion of any such metric in the cost function would increase significantly the already high computational cost of this algorithm making it infeasible to be executed onboard the UAV's CPU.

It is only very recently that approaches explicitly integrating SLAM information into path-planning started emerging, leading to only a handful of relevant works in the literature. In [10] the proposed approach was applied to UAV navigation, using visual cues and Rapidly-exploring Random Belief Trees (RRBT) [11], which consider the uncertainty of the robot along the generated paths predicted by means of robot and measurement models. These information-rich paths are planned according to an already known map, in order to predict future sensor measurements along the estimated path and, as a result, the selected plan can never be adapted in the case of map changes. Another major drawback of [10] is the high computational complexity of the algorithm, rendering it computationally feasible only on a powerful workstation.

Most relevant to this paper, is probably the most recently proposed receding horizon Next-Best-View Planner (NBVP) [12]. Designed for exploration and inspection tasks, this approach based on RRT considers inside the planning loop the visual information retrieved from a depth sensor in each of the nodes of the tree. Adopting a receding horizon methodology, according to which, in each planning iteration only a small segment of the trajectory towards the next best-view is executed at a time, allowing it to adapt the plan between iterations as new parts of the map are explored. However, the evaluation of the view in order to obtain the best-view from the RRT is a costly step applied to all the nodes in the RRT and therefore, it may poorly scale with the tree size. Inspired by the re-planning capabilities of NBVP, here, we employ a receding horizon methodology, while employing the far more applicable, albeit more noise-prone monocular visual-inertial sensor-setup. Despite this, our experimental results, demonstrate superior performance of the proposed methodology to NBVP in an inspection task.

II. PROPOSED APPROACH

Let us define Ξ as the set of all possible state configurations $\xi \in \Xi$ of the navigating robot. The robot's i-th state defined as $\xi_i = \{\mathbf{x}_i, \theta_i\}$, where $\mathbf{x}_i \in \mathbb{R}^3$ is its position vector and $\theta_i \in \mathbb{R}$ its heading angle or yaw. The aim here is to generate a point-to-point collision-free path Γ connecting a sequence of n discrete states from the current robot state ξ_0 to the goal state ξ_n. Let us define the application $\sigma : \mathbb{R} \rightarrow \xi$ so that $\sigma_i^j(s)$, $s \in [0, 1]$ is the linearly interpolated trajectory

[1]https://github.com/ethz-asl/okvis
[2]http://www.asctec.de/en/uav-uas-drones-rpas-roav/asctec-neo/

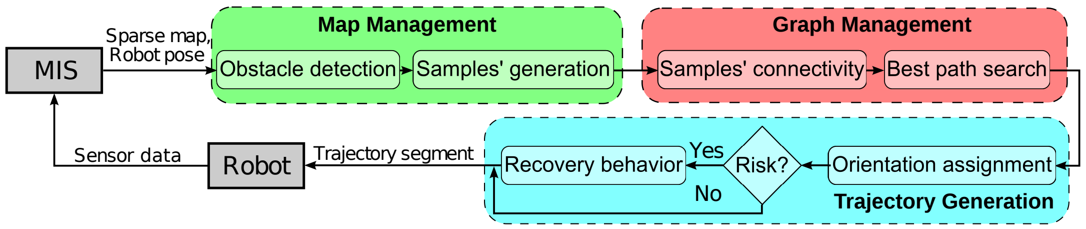


Fig. 2. The algorithm pipeline is fed with the sparse and noisy map generated by a nominal Monocular-Inertial SLAM (MIS) system. This MIS map is converted into processable obstacles and later used in the generation of samples distributed around them. The position of these samples are relevant since we assume most of the visual features that can be used by the MIS are located in the map obstacles. A graph is built connecting these samples as well as the current and goal positions. The graph is then searched for the global plan towards the goal. Lastly, we use the positions right after the current state to assign the best orientation of the next state and compute the next trajectory segment. Before the algorithm executes the next trajectory segment, it detects if it entails any collision risk. If so, a new trajectory segment to a recovery state is computed and executed, instead.

that connects the states ξ_i and ξ_j, i.e. $\sigma_i^j(0) = \xi_i$ and $\sigma_i^j(1) = \xi_j$. Adopting a receding horizon methodology as in [12], we compute the next best move for our robot in each planning iteration, which corresponds to the first trajectory segment σ_0^1 obtained from the complete plan Γ. After the execution of the first trajectory segment, a new planning iteration is called until the robot reaches the goal state. The steps of the proposed pipeline are illustrated in Fig. 2. Although the algorithm presented here is agnostic to the robotic platform employed, it has been designed with high-mobility robots in mind that have severely limited onboard computational capacity, such as UAVs.

III. Map management

Assuming that a nominal MIS system is running in parallel to the proposed pipeline, the input to our system is the sparse and noisy map generated by MIS, in the form of a collection of 3D points detected along the executed path of the camera. In order to convert this input into obstacles that we can process, we also run a probabilistic 3D occupancy map representation [13] in parallel, so that the noisy measurements get filtered out as they are acquired. The following subsections we describe how this map data is pre-processed to generate new position samples around the obstacles perceived in the MIS map.

A. Obstacle detection

At the beginning of the planning iteration, we process the current probabilistic occupancy grid at that moment to retrieve the probability of all the cells being occupied. The cells with a probability higher than a threshold will be transferred into a regular occupancy grid map and considered as obstacle cells later on in our algorithm. This occupancy grid is maintained during different planning iterations and only updated according to changes in the probabilistic occupancy map.

When transferring the occupied cells considered, we can also apply different strategies to boost the performance of the algorithm. For instance, we can use a different resolutions for the probabilistic occupancy map and the actual occupancy grid map we employ in the algorithm, allowing us to subsample the environment.

While the proposed approach can perform in 3D space using holonomic robots, multi-rotors UAVs are usually re-

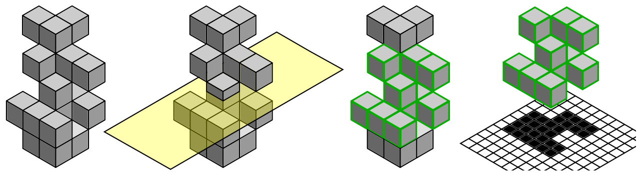


Fig. 3. Map segmentation for 2D navigation. The occupied cells forming the probabilistic occupancy map (grey boxes) are segmented according to a fixed navigation altitude (yellow plane). The relevant cells (green boxes) that may collide with the robot are then projected onto a 2D occupancy grid representing the map obstacles considered in the algorithm.

stricted to a single rotational degree of freedom in the space during normal navigation, i.e. changes in roll and pitch angles are usually employed to generate motion, whereas the yaw angle determines the robot orientation. Therefore, in this paper we focus on the 2D navigation planning at given altitude. In this case, the we only consider the obstacles that may lie in the robot's path when navigating at this altitude and project them onton a 2D occupancy grid as illustrated in Fig. 3.

B. Samples generation

The obstacles represented by the occupied cells of the occupancy grid are used to generate position samples around them. These obstacles are sources of visual features in a MIS setup and therefore, navigating close to them can lead to a more robust overall performance.

The generation of samples around the obstacles is done using part of the Sparse Tangential Network (SPARTAN) approach [14]. Inspired by the low-complexity of SPARTAN, here we adopt an implementation of its methodology for sampling. Based on an occupancy grid map representing the obstacles of the environment, the distance from each cell to the closest occupied cell is computed, up to a maximum distance d_{max}. Samples are generated at a given clearance distance ρ from the closest obstacle, but no closer than a distance d_{min} to another sample.

The main motivation behind employing this method is that the samples are generated using a so-called wavefront propagation, triggered when a cell of the occupancy grid representing the obstacles changes its status from occupied to free or vice versa. Since the maximum number of cells, whose status may change between planning iterations is limited according to the number of image frames retrieved by the visual sensor in the meantime, the computation time of the sample generation step is also bounded in time.

Additionally, sampling around the obstacles at a given constant distance ρ away from them, also improves the MIS performance as most of the visual features are tracked using similar scale during the navigation.

IV. Graph management

Based on the position of the samples computed in the previous section we build then a graph structure and maintain it during the subsequent planning iterations. Lastly, in each iteration we search in the graph for the best sequence of n positions that connects the current robot position $\mathbf{x}_0$ to the final goal position $\mathbf{x}_n$.

A. Graph generation

Let $\mathcal{G}$ denote an undirected graph defined as $\mathcal{G} = \{V, E\}$, where V is the set of vertices $v \in V$ and E is the set of edges $e \in E$. Each vertex v_i essentially represents a sample in free space located in position $\mathbf{x}_i$, whereas each edge $e_{v_i v_j}$ represents the collision free connection between the vertices v_i and v_j, i.e. the positions $\mathbf{x}_i$ and $\mathbf{x}_j$.

We establish edges connecting different vertices within a sphere of radius d_{conn}, if there exists a direct collision-free connection between them in the map. Formally, the set of edges E is defined as follows:

$$E = \left\{ \begin{array}{c|c} e_{v_i v_j} & \|\mathbf{x}_i - \mathbf{x}_j\| \le d_{conn} \\ \forall\, v_i, v_j \in V,\, i \neq j & \wedge \\ & f_{conn}(\mathbf{x}_i, \mathbf{x}_j) = \text{Free} \end{array} \right\}, \quad (1)$$

where $f_{conn}(\mathbf{x}_i, \mathbf{x}_j)$ is the function that evaluates whether the direct connection between the positions $\mathbf{x}_i$ and $\mathbf{x}_j$ is collision-free according to the current map. This function may take into consideration other parameters for collision detection, such as the dimensions of the robot.

Note that, in our approach, the set vertices V in $\mathcal{G}$ matches the set of samples generated in the previous section. Since the samples are only distributed around map obstacles, the sparsity and density of connections are lower than using, for instance, random sampling. This has a direct impact in the computation time of the several operations, such as graph search or graph mutation.

In addition to the samples generated around obstacles, the current robot position $\mathbf{x}_0$ and goal position $\mathbf{x}_n$ are also included in the set of vertices V; these are the vertices v_0 and v_n, respectively. The connectivity of v_0 is set according to (1), whereas the goal vertex v_n is connected ignoring the distance limitation d_{conn}. Since the area surrounding the goal position may be initially unexplored, there may not be samples nearby to be connected if its connections were limited to d_{conn}. Instead, we use the current set of vertices to advance towards the goal assuming that new samples will be generated as the exploration advances.

B. Graph update

The generation of a new graph from scratch, and particularly the collision detection in the evaluation of possible edges, is a costly operation. In our approach, the method described in Section III-B only modifies the sample set according to local map changes, while the other samples remain in the same position. Therefore, we can reuse the same graph between different planning iterations, if we modify the set of vertices V according these changes.

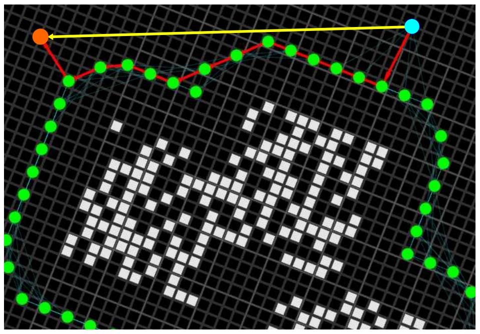


Fig. 5. Best path search connecting the current robot position (cyan) and goal position (orange) in the graph using Euclidean distance (yellow) and quadratic Euclidean distance (red). The vertices of the graph are depicted as spheres (green) and the edges as lines (light blue).

Note that, using this methodology, the connectivity between vertices added in previous planning iterations is not updated according to map changes as their connections have been already established. Since keeping the connectivity up-to-date in each iteration would be a costly operation, we delay the evaluation of each vertex's connectivity until it is required to be executed by the robot. This is explained in detail in the following section.

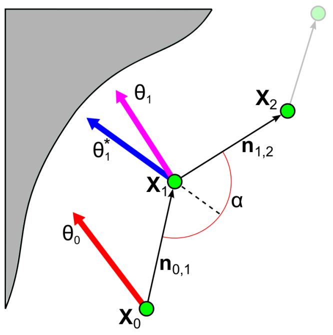
C. Best path search

In this step, we search the graph $\mathcal{G}$ for the best sequence of vertices, and therefore positions, that connect the starting vertex v_0 to the goal vertex v_n. We search in the graph using the Dijkstra's algorithm [15] that computes the minimum accumulated cost to all vertices of the graph from v_n. The lowest cost path from v_0 to v_n is then retrieved as the best path. We define the cost c of the edge $e_{v_i v_j}$ with the following expression:

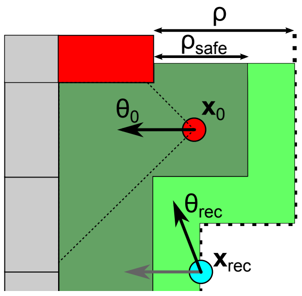
$$c(e_{v_i v_j}) = \|\mathbf{x}_i - \mathbf{x}_j\|^{\gamma}, \quad \gamma \ge 2\ , \quad (2)$$

where we use $\gamma = 2$ here in order to simplify the approach. The use of the quadratic Euclidean distance (i.e. $\gamma = 2$) instead of the usual Euclidean distance (i.e. $\gamma = 1$) is motivated by the type of path we want to generate; using the Euclidean distance, the resulting path is the shortest one, whereas using the quadratic version of it, the sum of small edges is preferred to long edges, as depicted in Fig. 5. As the vertices are evenly distributed around the map obstacles, we bias the path of the robot to be close to them. These positions are particularly relevant as the obstacles represent the main source of visual features used in MIS.

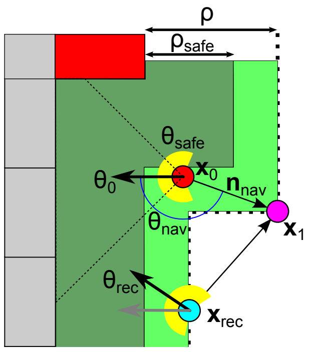
As mentioned in Section IV-B, the connectivity of the edges in the best path have to be evaluated again with the f_{conn} function. If a potential collision is detected, the relevant edge is deleted and the graph search is restarted until a collision-free best path is obtained. The best path,



(a) Next state orientation assignment



(b) Recovery behaviour: safe zone



(c) Recovery behaviour: safe direction

Fig. 4. Next trajectory segment generation. The next state orientation θ_1 computation in (a) using the current position $\mathbf{x}_0$ and orientation θ_0, following positions $\mathbf{x}_1$ and $\mathbf{x}_2$ and next orientation reference θ_1^* obtained from the angle α. The safe zone and the safe direction recovery behaviours in (b) and (c) are applied when the robot moves from the previous state (blue marker and grey arrow) to the current state $\xi_0 = \{\mathbf{x}_0, \theta_0\}$ (red marker and black arrow) and generate a new recover state $\xi_{rec} = \{\mathbf{x}_{rec}, \theta_{rec}\}$ (blue marker and black arrow) if a newly acquired obstacle (red box) is added to the current map (grey boxes) and endangers the navigation. The nominal navigation clearance ρ from obstacles modifies the safe zone defined by ρ_{safe} (light green area) when the new obstacle is added. When we are detecting if the next position $\mathbf{x}_1$ (magenta marker) is safely reachable following the direction $\mathbf{n}_{nav}$, we check if the angle navigation θ_{nav} lie within the θ_{safe} margin (yellow arc).

obtained as a sequence of vertices, is then processed to obtain a sequence of the equivalent 3D positions $\mathcal{X} = \{\mathbf{x}_0, \mathbf{x}_1, \ldots, \mathbf{x}_n\}$.

V. Trajectory Generation

Even though we obtain the complete sequence $\mathcal{X}$, we only execute the first segment in each iteration, such that we can adapt the path as new unexplored areas are discovered, changing the map. In this last step, the next state and the next trajectory segment are computed. Finally, we also describe some manoeuvres that allow the robot to overcome some situations, where it is unsafe to execute the planned trajectory segment.

A. *Next state orientation assignment*

In order to compute the next trajectory segment σ_0^1 we first need to define ξ_1. As we can obtain the next position $\mathbf{x}_1$ from $\mathcal{X}$, we only need to assign the next state orientation θ_1. For this purpose, we consider up to the two following positions along the best path sequence $\mathcal{X}$; i.e. $\mathbf{x}_0$, $\mathbf{x}_1$ and $\mathbf{x}_2$. The process to obtain the next state orientation θ_1 is depicted in Fig. 4a.

Let us define the direction vectors $\mathbf{n}_{0,1} = \mathbf{x}_1 - \mathbf{x}_0$ and $\mathbf{n}_{1,2} = \mathbf{x}_2 - \mathbf{x}_1$. Let α denote the smallest angle between $\mathbf{n}_{1,0} = -\mathbf{n}_{0,1}$ and $\mathbf{n}_{1,2}$. Then the next state orientation reference θ_1^* can be obtained by geometrically bisecting the angle α and heading towards the closest obstacle.

The assignment of the next state orientation θ_1 with next state orientation reference θ_1^* may result in an excessive turn according to the travelled distance. This is a pseudo-stationary rotation, leading to poor estimation of the depth of the visual features when using a MIS, which can severely affect the performance or even lead to a critical failure of the complete system. To prevent this situation, we limit the variation of the orientation according to a maximum turn-distance ratio $\hat{\omega}$. The final next state orientation θ_1 is then obtained with the following expression:

$$\theta_1 = \theta_0 + \left(\min \left\{ \frac{(\theta_1^* + \theta_{ext}) - \theta_0}{d}, \hat{\omega} \right\} \right) d \ , \qquad (3)$$

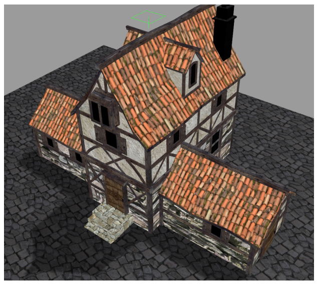
where d is defined as the distance $d = \|\mathbf{n}_{0,1}\|$ and θ_{ext} is a small angle increment in the navigation direction $\mathbf{n}_{0,1}$. If θ_{ext} is set to zero, the robot navigates heading perpendicularly to the surface of the obstacles, which improves the performance of the MIS as the robot moves laterally to the scene, increasing the parallax of the visual features tracked. However, moving completely laterally entails additional risks during the navigation as detailed in the following section.

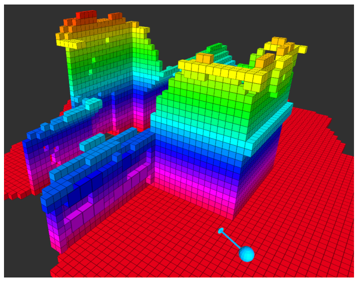
Finally, the next state is defined as $\xi_1 = \{\mathbf{x}_1, \theta_1\}$ and the next trajectory segment σ_0^1 from the current state ξ_0 to the next best state ξ_1 can be obtained and executed by the robot.

B. *Recovery behaviours*

Due to the limited view frustum of the camera onboard the robot, the planned path aiming to keep the closest obstacle in the field of view instead of heading towards the direction of motion, may lead to dangerous flights close to other, yet unexplored obstacles. Although this effect is minimized by re-orienting the robot's heading towards the navigation direction with θ_{ext}, significant changes in the environmental structure could still endanger the robot. If the robot detects that is facing such dangerous situations, a so-called recovery behaviour is triggered in order to overcome it.

When a recovery behaviour is triggered, the planned trajectory segment σ_0^1 is not executed. Instead, a new trajectory segment σ_0^{rec} is defined to reach a recovery state ξ_{rec}. Let us define $\mathbf{x}_{-1}$ as the position of the robot in the previous iteration and then set the position of recovery state to $\mathbf{x}_{rec} = \mathbf{x}_{-1}$. Since we have executed the trajectory from $\mathbf{x}_{-1}$ to $\mathbf{x}_0$ in the previous iteration, we assume it is also safe to execute it in the opposite direction. The orientation of the recovery state θ_{rec} will be computed according to the type of dangerous situation that has triggered the recovery behavior.


(a) Simulated scenario


(b) Dense SLAM map

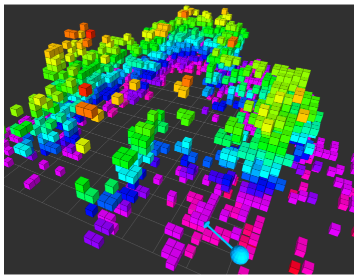

(c) Sparse SLAM map

Fig. 6. The model of the building used for facade inspection in the simulated environment is shown in (a). Figure (b) shows the map retrieved using a depth sensor as in the NBVP, whereas figure (c) shows the sparse and noisy map obtained from the MIS system as in the MISP. The latter, represents a far more challenging scenario for robot navigation.

1) Safe Zone: The robot navigation is meant to occur at a nominal clearance distance ρ, following the samples around the obstacles. However, due to changes in the map in the vicinity of the robot, it may happen that the robot is unsafely close to the map obstacles, triggering this recovery behaviour.

Let $d_{obs}(\mathbf{x})$ denote the distance to the closest obstacle in the map from the position $\mathbf{x}$. This recovery behaviour is triggered if $d_{obs}(\mathbf{x}_0) < \rho_{safe}$, where distance ρ_{safe} is defined, so that $\rho_{safe} < \rho$. The parameter ρ_{safe} denotes a threshold in which the robot can handle map irregularities without triggering the recovery behaviour. The orientation θ_{rec} is computed so that the robot heads towards the new obstacles that caused this unsafe situation (See Fig. 4b).

2) Safe Direction: Since we are navigating heading towards the closest obstacles in order to improve the performance of the MIS instead of heading towards the navigation direction, we cannot assure that the trajectory to reach the next best state is collision-free. Instead, we rely on the assumption that map is structured and presents a continuous surface and thus we can anticipate upcoming the obstacles following the surface. Nonetheless, this assumption is not valid when navigating, for instance, backwards as we can still collide with unexplored obstacles.

We obtain the navigation direction vector $\mathbf{n}_{nav} = \mathbf{x}_1 - \mathbf{x}_0$ and compute the navigation angle θ_{nav} from the difference between the current robot orientation θ_0 and the orientation of the navigation direction $\mathbf{n}_{nav}$. This recovery behavior is triggered if $|\theta_{nav}| > \theta_{safe}$, where θ_{safe} is a threshold within the range $\pi/2 \leq \theta_{safe} \leq \pi$. If it is triggered, the orientation of the recovery state θ_{rec} is computed so that the execution of the future trajectory segment from ξ_{rec} to ξ_1 does not trigger again this recovery behaviour (See Fig. 4c).

VI. EXPERIMENTS

In order to evaluate the performance of the proposed algorithm, we present two experiments: the first experiment simulates facade inspection from monocular and inertial data captured from a UAV, while in the second experiment real data is recorded using a monocular-inertial sensor in an urban environment. For both experiments, OKVIS [3][4] is used for the MIS system of the proposed pipeline.

Parameter		Value (Simulation)	Value (Real data)
Nominal Clearance	ρ	3.25m	6m
Max. explored distance	d_{max}	3.75m	6.5m
Min. dist between samples	d_{min}	0.5m	1.5m
Graph connection range	d_{conn}	2m	2m
Extra heading angle	θ_{ext}	20°	-
Turn-ratio limit	$\hat{\omega}$	45°/m	-
Safe zone threshold	ρ_{safe}	1.95m	-
Safe direction threshold	θ_{safe}	135°	-

TABLE I
TUNING PARAMETERS OF THE MISP.

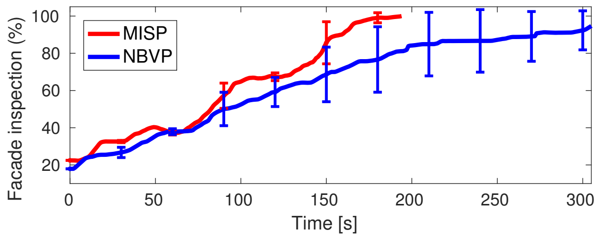


Fig. 7. The percentage of completion of the facade inspection task (mean and st. devation) with respect to time required for the MISP and the NBVP over 10 runs, sampled every $30s$. The NBVP spends $291s{\pm}107s$ on average to complete the task, whereas the MISP spends an equivalent of $170s{\pm}12s$. Note that MISP's rate of the task's completion is faster, as it explicitly exploits the scene's structure to guide the navigation.

A. Facade inspection in a simulation environment

Employing the Gazebo-based simulation RotorS [16] framework, we use an AscTec Firefly hexacopter UAV model equipped with a Visual-Inertial Sensor[3] (VI-Sensor), providing monocular-inertial information for the inspection of the facades of a building (see Fig. 6a). For this purpose, we limit the functionality of the proposed pipeline to 2D planning at the constant UAV altitude of $2m$. To complete the inspection task with our planner, and select the goal position of the planner inside the building. In this setup, the proposed Monocular-Inertial SLAM based Planner (MISP) will inspect the facades of the building, while trying to find an opening.

In order to compare the MISP with the NBVP [12], we adapt the NBVP to plan in 2D. We compute the gain of each randomly sampled state as the amount of unexplored facade area that is retrieved from such robot pose and thus focusing the algorithm in the completion of the inspection

[3]http://wiki.ros.org/vi_sensor

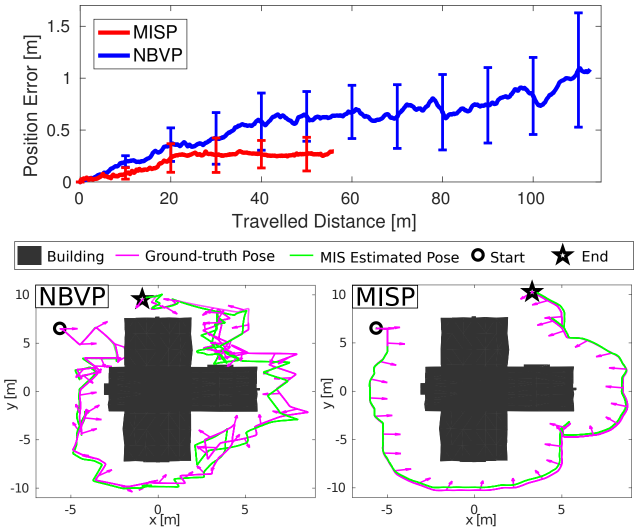


Fig. 8. The top figure shows the position error (i.e. global drift) of the robot's pose against the travelled distance, as estimated by the nominal monocular-inertial SLAM system used, which is the same for both MISP and NBVP. Results are averaged over 10 runs. The global drift at the end of the mission is $1.09m \pm 0.58m$ for NBVP and $0.30m \pm 0.22m$ for MISP. The bottom figures illustrate the complete path travelled to complete the mission. Although the NBVP is able to generate more excitation of the IMU, as evident in the growth of the global drift, MISP's more structured exploration results to smaller estimation errors.

task. Such feature is already considered in the original NBVP implementation[4] by providing the mesh model of the surface to inspect.

The simulated environment enclosing the building is an area of $16 \times 27\,m^2$. Limiting the UAV's maximum linear and angular velocities to $v_{max} = 0.3\,[m/s]$ and $\dot{\psi}_{max} = 0.75\,[\text{rad}/s]$, respectively, we set the dimensions of the UAV to $0.9m \times 0.9m \times 0.3m$. The NBVP parameters are tuned, such that the minimum and maximum iterations per planner-call are set to 15 and 5000, respectively and the maximum length of the RRT edges is set to $2m$. The maximum distance of the UAV to the perceived scene-depth is set to $5.75m$ and the map resolution is set to $0.25m$. These two parameters are tuned to the same values in the MISP for the sake of fairness, while the rest of the tuning parameters of the MISP are in Table I.

In the first part of this experiment, the effectiveness of both methods in completion of the inspection task is evaluated, employing a dense sensor in the NBVP, while using OKVIS in the MISP to retrieve the map. The difference in the quality of the map is notable, as illustrated in Figures 6b and 6c. The different sensor setup is mainly motivated by the original conception of the NBVP, which requires to plan in known free space and therefore relies on dense sensor to retrieve this volumetric information from the robot's surroundings. Using a MIS setup, the retrieval of such free space depends on the visual scene as well as the relative camera motion. Therefore, the MISP plans in unknown space, exploiting the structure of the environment and relying on the recovery behaviours to prevent collisions with unexpected obstacles. Fig. 7 shows

[4]https://github.com/ethz-asl/nbvplanner

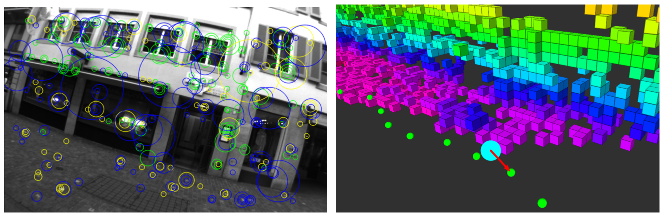


Fig. 9. The left figure shows an image frame of the real urban scene dataset and the visual features (circles) tracked by the MIS and represented by an octomap in the right figure. The MIS system processes the visual and inertial data estimating the sensor's position (cyan sphere), while generating a noisy and sparse map. This map is then used in the MISP to compute the best next trajectory segment (red arrow).

the time that each planner takes to complete the task on the same standard laptop (CPU only), while Fig. 1 illustrates the information inside the MISP at one of the final iterations of the facade inspection task.

When employing the MISP, the mean computation time per iteration is $24ms$ (st. deviation of $14ms$), whereas the NBVP needs $179ms$ on average (st. deviation of $69ms$). The dramatic reduction in the computation time of MISP, translates to a far better reaction time of response to changes in the environment. To evaluate the performance of the estimation accuracy of the MIS system during the mission, we employ OKVIS in combination with both the MISP and the NBVP. For the sake of fairness, we adapt the NBVP to permit planning in unknown space, as the MISP does. Note that we employ the MIS system only for map retrieval, while we use the ground-truth to localize the robot. The results, as illustrated in Fig. 8, illustrate that the MISP outperforms the NBVP in maintaining lower global drift, as expected, by design.

B. Real urban scene captured from a hand-held sensor

In this experiment, we evaluate the MISP using real data obtained from a VI-sensor in an urban environment. For safety reasons, the data is captured from a hand-held setup instead of using a UAV to navigate autonomously in this space. Therefore, the purpose here is to evaluate the algorithm on processing real input data, as opposed to the simulation experiment above. The human carrier of the VI-sensor moves laterally to the scene for $35m$ along a street (illustrated in Fig. 9) for about $30s$, keeping a constant clearance to obstacles. This trajectory is bound to be different from the computed planned path, but also rather similar, as the MISP is designed to generates paths that can are suitable to be processed by a MIS system. In reality, a planned path can anyway not always be followed by the UAV accurately, due to wind gusts and drift in the estimation of the internal sensors (e.g. measuring the rotational speed of the propellers).

The map resolution during this experiment is set to $0.5m$ and the other MISP parameters are set to the values illustrated in Table I. As we are not actively navigating by executing the best trajectory segment at each instant, we do not need to set the parameters related to the trajectory generation. The planner is called every $0.25s$ and compute

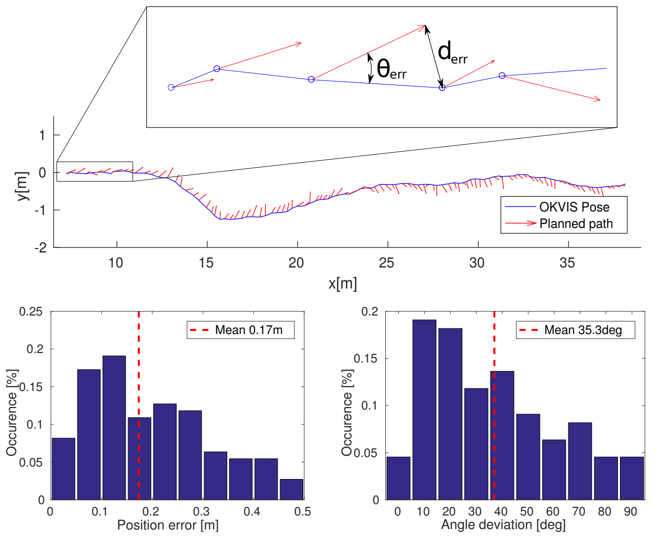


Fig. 10. The top figure depicts the path estimated by OKVIS (blue) and the predicted trajectory segments according to the proposed algorithm (red) and illustrates the definition of the pose error d_{err} as well as the angle deviation θ_{err}. The distribution of d_{err} and θ_{err} along the experiment are represented by histograms in the bot left and the bot right figures, respectively.

the best trajectory segment from the current sensor position, so that the robot would advance in the street for the case that the navigation was controllable. Note that, in contrast to the previous experiment, we obtain the pose of the robot directly from the OKVIS system as ground-truth is not available.

In order to evaluate our approach, we compare the planned action that the MISP would command to robot, with actual movement of the sensor. At each planning iteration, the next position based on the planned trajectory segment is computed using the MISP and adapted to the actual speed of the sensor. Then we obtain the position error of the planning iteration, as the distance between the planned next position and the actual position estimated by OKVIS. We also compute the angular deviation between the planned to the actual (i.e. measured by OKVIS) trajectory direction. These results, illustrated in Fig. 10, also embed other sources of error such as the MIS system accuracy or the deviation of the walker from a nominal clearance to the obstacles.

While in simulation we demonstrated the ability of the planner to plan and navigate around obstacles to complete a facade scanning task, the evaluations in this experiment demonstrate that the ability of the proposed pipeline to cope with real visual and inertial data, that give rise to a noisy SLAM map, attesting to the potential of this method in real navigation tasks.

VII. CONCLUSIONS

Following the relative maturity of SLAM pipelines and their application to UAV navigation, in this paper we propose a system to put monocular-inertial SLAM in the loop of navigation to plan a trajectory for a small UAV in real-time in a previously unknown environment. Dealing with the traditionally noisy and sparse SLAM maps, the proposed approach generates point-to-point paths, actively improving the SLAM estimates of the scene and the motion of the UAV, by biasing paths towards feature-rich areas and beneficial camera motions. Demonstrated to outperform the state of the art in computing collision-free paths on the fly onboard computationally constraint processors available onboard small UAVs, the proposed pipeline is put to the test in both simulation and real experiments.

Future directions will focus on the tighter integration of SLAM into the path planning strategy, which would serve as a basis for an integrated active SLAM framework.

REFERENCES

[1] G. Klein and D. W. Murray, “Parallel tracking and mapping for small AR workspaces,” in *Proceedings of the International Symposium on Mixed and Augmented Reality (ISMAR)*, 2007.

[2] S. Weiss, M. W. Achtelik, S. Lynen, M. C. Achtelik, L. Kneip, M. Chli, and R. Siegwart, “Monocular Vision for Long-term MAV Navigation: A Compendium,” *Journal of Field Robotics (JFR)*, vol. 30, pp. 803–831, 2013.

[3] S. Leutenegger, P. Furgale, V. Rabaud, M. Chli, and R. Siegwart, “Keyframe-based Visual-Inertial SLAM using Nonlinear Optimization,” in *Proceedings of Robotics: Science and Systems (RSS)*, 2013.

[4] S. Leutenegger, S. Lynen, M. Bosse, R. Siegwart, and P. Furgale, “Keyframe-based visual-inertial odometry using nonlinear optimization,” *International Journal of Robotics Research (IJRR)*, vol. 34, no. 3, pp. 314–334, 2015.

[5] M. Likhachev, D. I. Ferguson, G. J. Gordon, A. Stentz, and S. Thrun, “Anytime Dynamic A*: An Anytime, Replanning Algorithm.” in *International Conference on Automated Planning and Scheduling (ICAPS)*, 2005.

[6] L. Heng, L. Meier, P. Tanskanen, F. Fraundorfer, and M. Pollefeys, “Autonomous obstacle avoidance and maneuvering on a vision-guided mav using on-board processing,” in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2011.

[7] S. M. LaValle, “Rapidly-exploring random trees: A new tool for path planning,” Computer Science Department, Iowa State University, Tech. Rep. 98-11, 1998.

[8] M. Otte and E. Frazzoli, “RRT X: Real-time motion planning/replanning for environments with unpredictable obstacles,” in *Workshop on the Algorithmic Foundations of Robotics XI*, 2015.

[9] S. Karaman and E. Frazzoli, “Incremental Sampling-based Algorithms for Optimal Motion Planning,” in *Proceedings of Robotics: Science and Systems (RSS)*, 2010.

[10] M. W. Achtelik, S. Lynen, S. Weiss, M. Chli, and R. Siegwart, “Motion and Uncertainty Aware Path Planning for Micro Aerial Vehicles,” *Journal of Field Robotics (JFR)*, vol. 31, no. 4, pp. 676–698, 2014.

[11] A. Bry and N. Roy, “Rapidly-exploring random belief trees for motion planning under uncertainty,” in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2011.

[12] A. Bircher, M. Kamel, K. Alexis, H. Oleynikova, and R. Siegwart, “Receding horizon ”next-best-view” planner for 3D exploration,” in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2016.

[13] A. Hornung, K. M. Wurm, M. Bennewitz, C. Stachniss, and W. Burgard, “OctoMap+: An efficient probabilistic 3d mapping framework based on octrees,” *Autonomous Robots*, vol. 34, no. 3, pp. 189–206, 2013.

[14] H. Cover, S. Choudhury, S. Scherer, and S. Singh, “Sparse tangential network (SPARTAN): Motion planning for micro aerial vehicles,” in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2013.

[15] E. W. Dijkstra, “A note on two problems in connexion with graphs,” *Numerische mathematik*, vol. 1, no. 1, pp. 269–271, 1959.

[16] F. Furrer, B. Burri, M. Achtelik, and S. Siegwart, “RotorS – a modular gazebo MAV simulator framework,” *Studies in Computational Intelligence*, vol. 625, pp. 595–625, 2016.