A Complete System for Vision-Based Micro-Aerial Vehicle Mapping, Planning, and Flight in Cluttered Environments

Helen Oleynikova, Zachary Taylor, Alexander Millane, Roland Siegwart, and Juan Nieto
Autonomous Systems Lab, ETH Zürich

***Abstract*— We present a complete system for micro-aerial vehicle autonomous navigation from vision-based sensing. We focus specifically on mapping using only on-board sensing and processing, and how this map information is best exploited for planning, especially when using narrow field of view sensors in very cluttered environments. In addition, details about other necessary parts of the system and special considerations are presented. We compare multiple global planning and path smoothing methods on real maps made in realistic search and rescue and industrial inspection scenarios.**

I. Introduction

Autonomous navigation from on-board sensing is essential for Micro-Aerial Vehicles (MAVs) in many applications. Specifically, we want to create MAVs that can assist human operators in difficult inspection tasks in search and rescue (S&R) and industrial applications. To address both of these needs, we do not use GPS, and focus on online mapping and planning from only vision-based sensors.

This paper aims to present a complete system capable of performing repeated inspections of the same scene. Our previous work proposes mapping [1] and online re-planning [2], [3] methods for safe navigation of previously-unexplored spaces. We focus specifically on explicitly mapping free space in very cluttered environments, and exploring planning strategies that are inherently conservative: that is, they only allow traversal of space that is confirmed to be free. Here we aim to extend the local planning work to also cover global planning scenarios, where a map is already available (either on the return route of the current mission or from a previous mission).

We compare various global planning methods, including improving on our previously-proposed topological graphs built from Euclidean Signed Distance Fields (ESDFs) [4]. We also extend our local replanner to be used as a path-smoothing method for converting lists of waypoints from global planners to dynamically-feasible timed trajectories. All planning methods are evaluated on three realistic scenarios, two from a search and rescue training area, and one from an industrial environment with large machinery, recorded with the MAV system described in this paper.

The aim of this work is to serve as a reference for the *complete system* needed for these applications, requiring no off-board processing or external sensing. This set-up allows the robot to be robust to loss of communication (which is typical in real S&R scenarios), and behave intelligently and safely even when not under direct control of a human. We discuss the control, state estimation, sensor, and hardware concerns and requirements for such systems, and make the software for all parts available open-source.

Fig. 1: The platform used for collecting the test datasets, a custom-built drone using the DJI FlameWheel F550 frame, an Intel NUC for on-board processing, Pixhawk for flight control, VI Sensor for monocular state estimation and stereo depth, and an Intel RealSense D415 for RGB-D input.

The contributions of this work are as follows:

- We present a complete open-source system for autonomous GPS-denied navigation,
- A thorough treatment of considerations in 3D mapping for planning applications, expanding on our previous work [1] with regards to unknown space and generation methods,
- We extend our previous work on topological sparse graph generation [4] to create more useful graphs, faster,
- A discussion and comparison of various global planning methods,
- We extend our previous work in local planning [2], [3] to also be usable for path smoothing and compare to several competing methods.

II. Related Work

We will give a very abbreviated overview of related work, as more thorough discussions of all parts are available in our previous work [2], [1], [3], [4].

We aim to show a complete system for mapping and planning on-board an autonomous UAV, using vision-based sensing. Lin *et al.* [5] presented a similar complete system, spanning visual-inertial state estimation, local re-planning, and control. However, there are a few key differences between the frameworks proposed: ours focuses strongly on

the map representation we use and exploiting all the information within, while their uses a standard occupancy map. More importantly, our planning is *conservative*, meaning we will only traverse known free space, while theirs assumes unknown space is free. Therefore, we must make more considerations about the contents of our map with these restrictive assumptions. We also offer an evaluation of global and path smoothing planning methods.

Mohta *et al.* [6] also propose an autonomous system for fast UAV flight through cluttered environments. There are a few key differences with their work, especially on mapping and planning. They use a LIDAR as the main sensor, which gives 360° field of view for collision detection, removing many of the issues with narrow field of view sensors which we attempt to address in this work. They also only keep a small local 3D map, and use a global 2D map to escape local minima, whereas we use a full global 3D approach at comparable computation speeds. For how the mapping is used, they attempt to break the world into overlapping convex free-space regions, which grows in complexity and is increasingly more limited as the space gets more complex, while we always plan directly in the map space. They also make no considerations for how drift will affect the map other than to keep only a local 3D map.

Finally, the system we propose is conceptually similar to the original system in our previous work [7]. The core differences are that we improved every individual component, designed and evaluated a custom mapping system, and proposed a way to do local re-planning as well (whereas the previous work was only global planning). This makes the system proposed in this work much more robust and able to deal with changes in the environment.

III. System Overview

We describe a complete MAV hardware and software system capable of supporting autonomous flight with only on-board vision-based sensing. All of the software described in the system has available open-source with provided links.

A. Overall Architecture

We show an overview of the complete system in Fig. 2, focusing on the data flow between mapping and planning processes. Stereo and depth images can be used interchangeably for the mapping, combined with a pose estimate from visual-inertial monocular odometry. The odometry is then fused with the body IMU of the MAV to create a high-rate pose estimate used for control. Position control runs on the on-board computer, and gives roll, pitch, and yaw-rate commands to the attitude controller on the flight controller. The output of the planning stack are timed trajectories, sampled at the position controller rate (typically 100 Hz). Since we use a model-predictive controller (described in more detail below), this allows us to have a much higher trajectory tracking accuracy due to the long control horizon.

The diagram shows operation set up over two stages: the first is online flight through completely or partially unexplored space, in white, and in purple global planning in

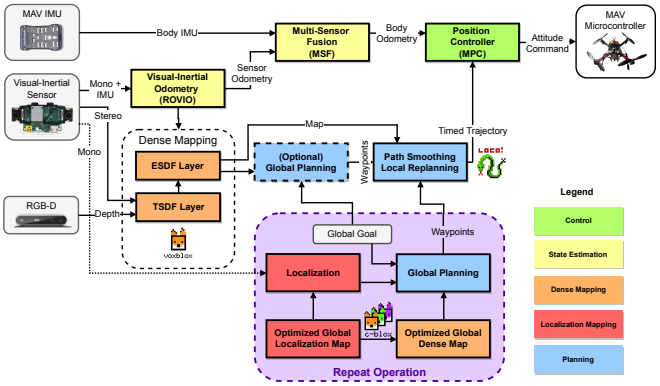


Fig. 2: Overall system architecture, showing most key components of the system, and the data flow between mapping and planning. A second use-cases is shown in purple, where a previously-optimized global map is available for a mission.

previously-built maps. This system allows us to do an initial flight, optimize a map using off-line tools, then perform repeat inspections in the same area. Using a local planner on the output of the global planners guarantees that even if the environment changes between missions, we are able to safely navigate through it by replanning locally. Likewise, we can use the same tools without stopping to optimize a map – for instance, using a global planner to return home at the end of a mission.

B. Hardware

Fig. 1 shows the MAV used to collect the evaluation datasets and test the overall system. It is built around a DJI FlameWheel F550, with 6 motors for actuation. The low-level control is performed with a pixhawk[1], using custom firmware that accepts attitude and yaw-rate commands [2]. This is necessary as we do not use GPS and fly indoors and in other environments where magnetometer readings are unreliable, therefore giving an absolute yaw reference is both undesirable and meaningless in our local coordinate frame.

On-board processing, which runs everything shown in Fig. 2, is performed on an Intel NUC. The main sensor is a custom-made visual-inertial sensor [8], with two monochromatic cameras in a stereo configuration, hardware-synced to an ADIS448 IMU. It is used for mono visual-inertial state estimation, and also for stereo depth for mapping. Optionally, we also use an Intel RealSense D415 for an additional source of RGB-D depth.

Though a dedicated visual-inertial sensor is a nice-to-have for such platforms, there is currently not one available off the shelf that is suitable for MAV flight. Instead, we recommend using a USB machine vision camera, and hardware time-synchronizing it to a flight controller. We make a sample driver implementing this for the FLIR Blackfly or Chameleon 3[3] and the pixhawk available[4]. This set-up is also extend-

[1] pixhawk.org
[2] github.com/ethz-asl/ethzasl_mav_px4
[3] ptgrey.com/blackfly-usb3-vision-cameras
[4] github.com/ethz-asl/flir_camera_driver

able to multi-camera systems, as multiple cameras can be triggered from the same pulse.

C. Control

We use a cascaded control architecture, with an inner loop that controls attitude and runs at a minimum of 100 Hz on the MAV autopilot, and an outer position control loop that runs on the on-board computer at 100 Hz. For the outer loop, we use a non-linear Model Predictive Control (MPC), proposed by Kamel *et al.* [9] and available open-source[5].

The MPC takes in the body odometry estimates from MSF (described in Section III-D) and a timed full state trajectory to track. We exploit the properties of the flat state for MAVs to only need to specify position, yaw, and their derivatives [10].

One of the advantages of using an MPC over a PID loop for trajectory tracking is that the MPC is able to look ahead at future trajectory points, and minimize tracking error over the complete horizon. This means that overall trajectory tracking performance is improved significantly, and there are advantages to planning high-fidelity, dynamically-feasible trajectories, as they will be executed almost perfectly.

The non-linear MPC also has a *very* long horizon of 3 seconds, or 300 timesteps. While this is very convenient for executing long complex global trajectories, special care must be taken when using it for online replanning. Namely, we need to timestamp our entire trajectory to be monotonically increasing, and trajectory updates must be inserted into the correct place in the MPC queue. The queue is cleared if a trajectory with a time before the current execution time is sent, and the new trajectory replaces the complete queue. Fig. 3 shows an example of a replan cycle, happening for the purposes of the illustration at 50 Hz. The MPC queue is initialized with a starting trajectory. The local replanning "locks" the beginning of the initial trajectory, including the first 20 ms which is when the controller will receive the updated trajectory, and also another 30 ms look-ahead so that the reference does not change too quickly, then replans starting at 50 ms. A 3 second chunk, starting at the 50 ms, is then sent to the controller queue, which inserts the updated trajectory at the correct time, even though it has only executed up to 20 ms.

This queueing scheme allows us to replan at any given rate, while making sure that the controller always has a reasonable trajectory within its time horizon.

D. State Estimation

All state estimation is done on-board and not using external sensing (i.e., no vicon or GPS). This gives our system flexibility to be used in complex GPS-denied environments. Our main state estimator is Rovio[6], which is a robust visual-inertial odometry framework [11], [12]. Rovio is a filter-based estimator, which uses direct photometric error on a small number of patch features in the image. For our application, this design has a few distinct advantages over more traditional, keypoint-based methods like Okvis [13]: a filter with a small state-space (using only 25 features) is fast to compute, even on the on-board CPU, and using direct photometric error makes the method resistant to motion blur. In our experience, Rovio is comparably accurate to other methods such as Okvis and VINS-mono [14], but more robust under real conditions.

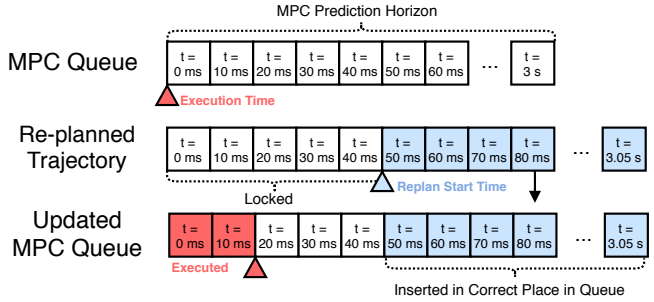


Fig. 3: Diagram showing the MPC queue, and how it is updated when a section of the trajectory is replanned. Note that slightly more of the trajectory is locked down than is executed during the planning time. This allows the MPC to not have very abrupt changes in reference.

However, Rovio only gives us the odometry in the *sensor* frame, which for our system is usually the IMU that is part of the visual-inertial (VI) sensor. Rovio also only outputs updated poses at the camera frame rate. Depending on the hardware set-up, we have two solutions to receive body-IMU-frame odometry at 100 Hz.

If using a separate VI sensor, then the odometry estimate from Rovio must be transformed into the body frame and fused with the body IMU. This is done using Multi-Sensor Fusion (MSF) [15], which is a highly-configurable filter capable of taking multiple sensors and pose sources[7].

In another configuration, where the only IMU on the system is hardware time-synchronized to a camera, no transformation or fusion needs to be done. We only need to use the estimated Rovio biases to propagate the odometry estimate using incoming IMU measurements. This is done using a package called `odom_predictor`, which queues incoming IMU measurements and re-applies them when new (delayed) estimates are available from Rovio. It is also available open-source[8].

E. Localization

This paper aims to address issues with creating dynamically-feasible global plans. However, global planning requires *global localization*. Since all visual- and visual-inertial odometry frameworks drift, no matter how little, long-term operation or operation in previously-explored environments requires the ability to perform loop-closures.

To localize against a global map, we use maplab [16], an open-source[9] framework for creating, storing, optimizing,

[5] `github.com/ethz-asl/mav_control_rw`
[6] `github.com/ethz-asl/rovio`
[7] `github.com/ethz-asl/ethzasl_msf`
[8] `github.com/ethz-asl/odom_predictor`
[9] `github.com/ethz-asl/maplab`

and localizing in visual-inertial maps.

To create global maps, we first generate a sparse pose-graph of landmarks in the observed scene. This is done by using Rovioli [16], a front-end for Rovio that also does feature tracking and extraction (independently of Rovio's 25 tracked patch features).

This sparse graph can then be loop-closed and optimized using bundle adjustment in an offline process, giving optimized, globally-consistent poses for all keyframes. To generate the global map, we can then replay all pointclouds from the initial flight and integrate them into a dense map using optimized poses. Localization in the matching sparse map will then line up correctly with the optimized dense map.

If using ORB-SLAM [17] as the SLAM system, a better approach is to build up a dense map using submaps, and only fuse them when the covariance between their relative poses is small enough, as presented in our previous work [18]. This allows us to get a globally-optimal dense map in a single step, without having to run a recorded dataset through an offline framework. However, since ORB-SLAM does not support localization against a previous map, this limits us to global localization within a single mission.

IV. Dense Mapping

Dense mapping is key to planning performance, as a plan can only be as good as the map. We use a flexible mapping framework called *voxblox*[10], introduced in our previous work [1]. The framework is centered around using Signed Distance Fields (SDFs), or voxel grids of distance values to surfaces. We use two different types of SDFs: Truncated Signed Distance Fields (TSDFs), based on Curless and Levoy [19] and KinectFusion [20] for integrating point cloud data, in a method that gives a more accurate surface estimate than occupancy-based methods, but uses projective distances and truncates the value to a small band around surface crossings. The second type of field is a Euclidean Signed Distance Field (ESDF), which store Euclidean, rather than projective distances to each obstacle, and are not truncated to a specific range. These ESDFs are then the representation we use for planning, as they contain collision information for the entire map, and can also be used to quickly get obstacle gradients, which will be essential for some of the planning methods below. A system diagram showing inputs, outputs, and data flow is shown in Fig. 4.

A. Euclidean Signed Distance Fields

This section will discuss how an ESDF is computed from a TSDF. A more thorough analysis, including upper bounds on error introduced by various assumptions and a comparison with occupancy-based methods is offered in our previous work [1].

While it might seem unintuitive that there is a non-trivial process to convert from a TSDF to an ESDF, this is because the distances in both representations are computed differently. TSDFs use projective distance, or distance along the ray cast from the sensor to the surface. These distances are fairly accurate near surface crossings, but quickly accumulate large errors [21]. In contrast, an ESDF needs true Euclidean distances, which can only be calculated in a global fashion. Luckily, incremental algorithms exist for computing ESDFs from occupancy maps [22], and our work extends these methods to also work from TSDFs.

[10] github.com/ethz-asl/voxblox

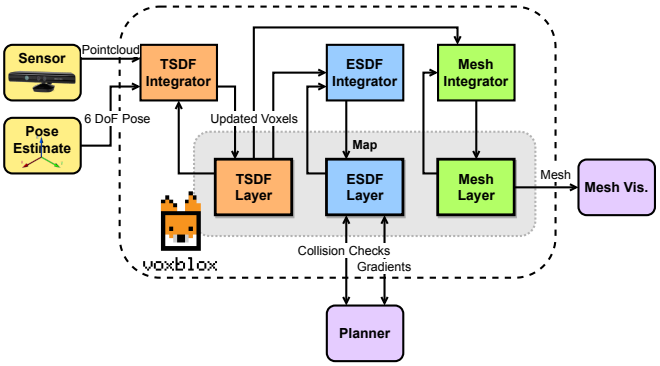
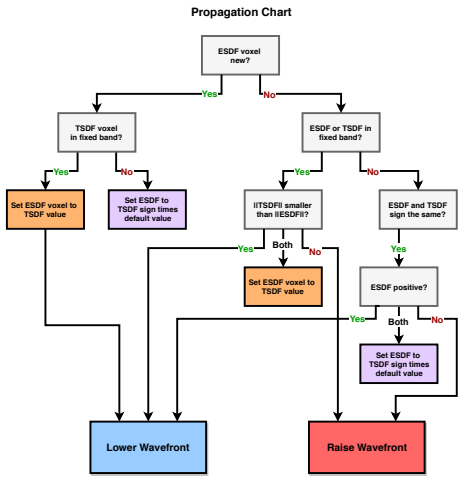


Fig. 4: Voxblox system diagram, showing how the TSDF and ESDF layers are interconnected through integrators.

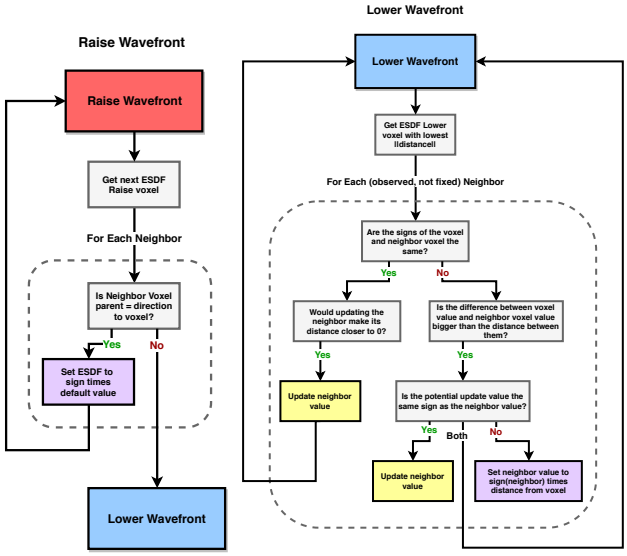
Generating the ESDF is done in three stages, detailed in Fig. 5: propagation (a), raise (b), and lower (c). The first, and most different from occupancy-based methods such as [22], is the TSDF to ESDF propagation. Due to the inaccuracy of TSDF distance estimates, we define a radius called a "fixed band" around the surface, which must be at least one voxel size and at most equal to the truncation distance. TSDF values that fall within this band are considered fixed, copied into the ESDF, and can not be altered in the ESDF update. Next, updated voxels may simply retain their value, or be put into the "lower" wavefront (when their updated distance values are closer to the surface than before) or the "raise" wavefront (when their values become farther from the surface). If performing a batch update (i.e., the entire layer at once), all voxels will go to the lower wavefront.

After propagating all updated values from the TSDF into the ESDF, we then process the raise wavefront. This consists of simply invalidating all voxels in the wavefront and their children. Since each voxel stores its "parent" (if the voxel is in the fixed band from the TSDF, it is its own parent), this is then an incremental brushfire operation. All voxels cleared from the raise wavefront have their still-valid neighbors added to the lower wavefront, to guarantee that their values get updated.

The lower wavefront behaves similarly to the occupancy case: iterate over all voxels in the lower wavefront, if their neighbor's distance to the surface can be lowered through the current voxel, then update the neighbor and add it to the lower wavefront. Special distinctions must be made when implicit zero-crossings exist: i.e., two voxels are neighbors with opposite signs, and neither is fixed. This case is further explained in Fig. 5c.



(a) TSDF to ESDF Propagation



(b) ESDF Raise (c) ESDF Lower

Fig. 5: Diagrams explaining how the ESDF is constructed in three stages.

B. Unknown Space

A key problem with mapping for collision avoidance is deciding how unknown space is handled. There are two options: treating all unknown space as free (optimistic), and treating all unknown space as occupied (pessimistic or conservative).

While many local collision avoidance works treat unknown space as free and have a high replan rate to avoid collisions [23], this is inherently unsafe. While it works well in uncluttered environments, where most unknown space *is* free, this assumption gets progressively worse as obstacle density in the environment increases and sensor field of view decreases.

Since our work aims to deal with the worst possible case, which is very obstacle-dense environments and a narrow field-of-view sensor, we cannot adapt the optimistic strategy. In fact, we always plan to stop in known free space, to guarantee safety in partially-unexplored environments.

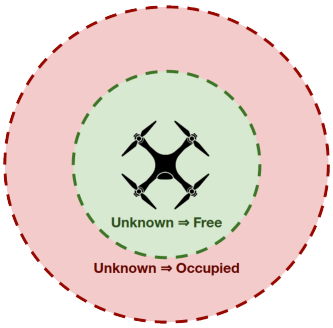


Fig. 6: Diagram showing the two radiuses around the current robot pose: a small clear sphere, which should be only slightly larger than the robot, and an occupied sphere, which should be roughly the size of the planning radius.

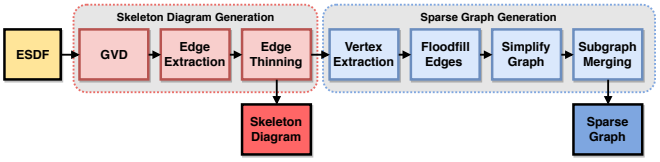


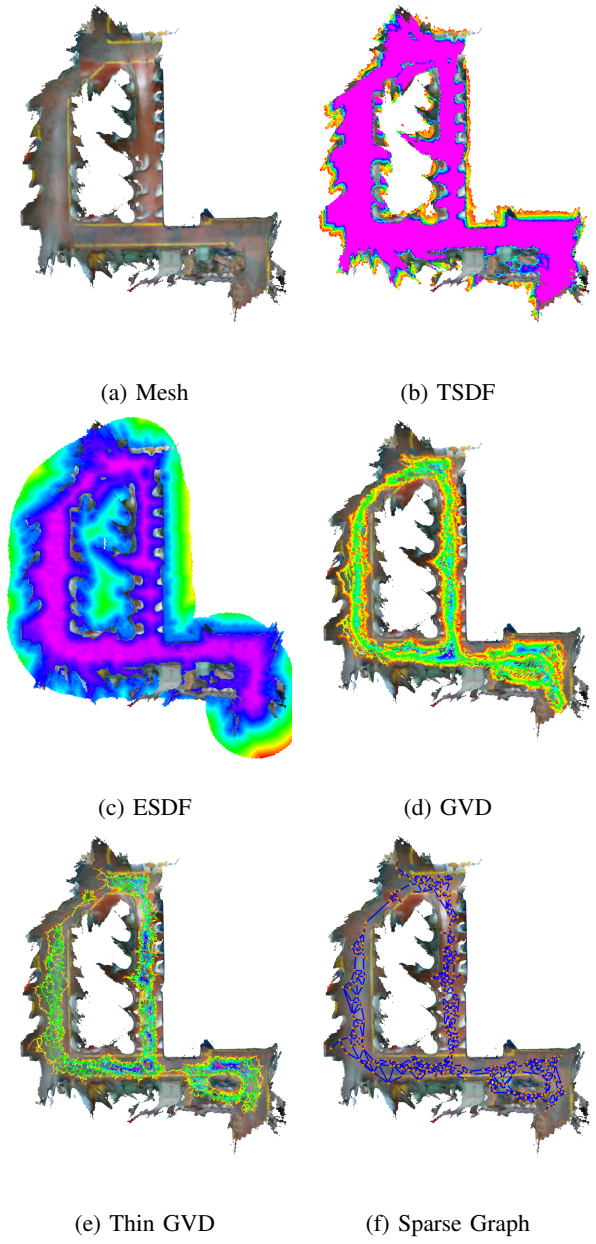
Fig. 7: Steps in generating a sparse topological graph, also referred to as a skeleton graph.

However, it is a challenge to encode unknown space information in the ESDF, as the ESDF integration requires TSDF distances to build on, and those are either positive or negative. There is the additional problem that the MAV has no knowledge of the state of its current position at start-up or take-off, as it has never observed the space it occupies at the start. Furthermore, if the sensor has a very narrow field of view, the space it perceives in front of the sensor may not be wide enough to fit the entire robot body, essentially paralyzing the robot to never move. Finally, it is not clear how to correctly treat voxels bordering unknown space in ESDF computation.

We propose a simple strategy to resolve these issues, originally proposed in our previous work [3]. The idea is to have two overlapping "spheres" centered around the current robot pose that are applied in the ESDF, shown in Fig. 6. The inner sphere is small and only slightly larger than the robot radius and is called the "clear sphere", which sets unknown space within it to free in the ESDF. The outer sphere affects all points not within the clear sphere and sets all unknown voxels to occupied. Any voxels that receive distances from this operation are marked as hallucinated in the map, so that as soon as real distance measurements are available from the TSDF, their values are overwritten.

V. Sparse Topology

In this section, we extend our work on generating sparse topological skeletons from [4]. We describe a complete method to extract a sparse graph of the traversable free space in an ESDF from only ESDF map data. This sparse graph is then used for very fast global planning in later sections of



(a) Mesh (b) TSDF

(c) ESDF (d) GVD

(e) Thin GVD (f) Sparse Graph

Fig. 8: Different stages of sparse topology skeleton computation, shown on the machine hall RealSense dataset. Colors represent distances from obstacles.

this work.

Our main contributions over our previous work in this section include:

- Switching to a simpler definition of the Generalized Voronoi Diagram (GVD), using 26-connectivity.
- Extending the method to work for both full and quasi-Euclidean distance.
- Speeding up sparse graph construction through use of flood-fill operations on the edges.
- Proposing a new sparse graph simplification and subgraph re-connection methods, which produce more usable graphs significantly faster.

Fig. 7 shows the stages in the process, illustrated at key points with Fig. 8.

First, the generalized voronoi diagram (GVD) is constructed. This is done by iterating all voxels in the ESDF to find "ridges" or "basis points". A point is considered a basis point if its neighbors have parent vectors that are at least some angle apart. The angle differs depending on whether quasi-Euclidean or full Euclidean distance is employed, as quasi-Euclidean has a lower resolution in parent directions. For full Euclidean distance, the separation angle is $45°$, and for quasi-Euclidean it is $90°$.

We use a different definition of what belongs on the GVD from our previous work, making the definition simpler and more physically meaningful. Before we used 6-connectivity when evaluating belonging on the GVD, but now we use full 26-connectivity for all parts of the process. A point is considered a GVD face if it has 9 or more neighbors that are basis points, an edge if it has 12 or more, and a vertex at 16 or more. For the purposes of the remaining method, we do not consider faces, as the goal is to build as sparse of a diagram as possible.

Finally, to create a sparse diagram (which is simply a voxel layer containing number of basis point neighbors and whether the point is an edge or vertex or not), we apply a series of thinning operations described in more detail in [4]. Fig. 8d and e show the difference before and after thinning: after the operation is applied, all that is left is a one voxel-thick skeleton diagram.

To be more useful for planning, we want to further sparsify this diagram into a graph, removing the notion of discrete voxel sizes. We propose a different method of generating the sparse graph from the skeleton diagram here, which does not follow the underlying diagram exactly (as our previous work did) but greatly simplifies it. The downside to this is that edges no longer follow the maximum-clearance edges, but may now pass through intraversable space. However, they will always be *very near* traversable space, therefore as long as a flexible smoothing method is employed afterwards, a feasible path will be found.

We take all vertices in the skeleton diagram, assign them unique vertex IDs, and perform a flood-fill in all directions that contain edges, labelling the edges with the two nearest vertex IDs. This is a speed improvement and simplification over our previous "edge-following" algorithm, and does not suffer from cases where edge connections are missed.

To simplify this graph, we attempt to remove all vertices that are not adding information to the graph – essentially, vertices that are on straight lines or nearly-straight lines between other vertices. The filter consists of removing vertices that have exactly two edges, and whose removal will not displace the edges more than 2 voxels.

As a final step, we attempt to find any way to reconnect disconnected subgraphs. We label each self-contained subgraph in the sparse graph with another flood-fill operation, and assign subgraph IDs. We then iteratively search for connections from all subgraphs to all other subgraphs: if

a connection is found, one of the subgraphs is relabelled. To see if two sub-graphs can be connected, we first search along the skeleton diagram using A*: if a connection exists, we insert a new edge between the two closest vertices. If no connection exists in the diagram (which very occasionally happens due to discretization error), we search in the traversable space of the ESDF, again using A*. If a path exists there, then we attempt to verify that ESDF path is valid and close to straight-line.

Both of these methods are much faster and more reliable at producing usable sparse planning graphs than our previous approach.

VI. Global Planning

In this section we will discuss different global planning strategies and their advantages and disadvantages. It is assumed that a complete global map is available for these methods. We plan only in position space, assuming the MAV is a sphere for collision-checking purposes (as this makes it rotationally-invariant), and the output of all the global planners is a list of position waypoints from start to goal. All methods described here are available open-source[11].

A. Sampling-based Methods

The first class of methods we will consider are sampling-based methods. They are very well suited for large 3D problems, as they do not suffer from the same scaling problem as search-based methods. One of the most commonly-used classes of methods is Rapidly-exploring Random Trees (RRT) [24], where random points are sampled in the planning space (in our case, just 3D position space) and iteratively connected to a tree. Once the goal point is sampled, there exists a path from the start point (the root of the tree) to the goal.

One key advantage of this class of methods is that all that is required is a way to determine if a randomly-sampled state is valid or not, and a way to determine whether connections between states are valid. We propose two different methods to do this: one in the ESDF, which is pessimistic (assuming unknown space is occupied) and on the TSDF, which is optimisic (assuming unknown space is free). As long as they are coupled with a conservative/pessimistic local planner, either method can be used.

The look-up in the ESDF requires only a single point per pose (as the ESDF already stores the distance to the nearest obstacle, and therefore as long as that distance is larger than the robot radius, the point is feasible), while the TSDF look-up requires looking up an entire sphere and every voxel within. Additionally, for determining if motions between two states are valid, we use a ray-cast operation to not miss any potentially occupied voxels.

1) RRT-Connect: RRT-Connect is a very fast variant of the original RRT algorithm, which does a bi-directional search: growing a tree from both the start and the goal points [25]. One the downside, it does not optimize the paths (the algorithm terminates once a path is found), so they are often much longer than necessary in complex environments. For the purposes of our benchmarks, we treat this as the "first solution" time and solution length for RRT-based algorithms.

2) RRT:* RRT* combines the best of both A* (which is an optional planning algorithm) and random sampling to create a probabilistically-optimal planning solution [26] . This method samples new points and rewires the graph for shorter solutions, up until a time-limit is reached. There are other variants, such as Informed RRT* [27], which iteratively shrink the sampling space after an initial solution is found, similar to how the admissible heuristic is used in A*.

In general, this is the class of methods we prefer to use, as they give short, nearly-optimal paths, and it is possible to decide how to trade-off computation time versus optimality depending on application.

3) PRM and PRM:* For planning in a changing map, RRT-based methods are a very powerful tool, as they do not store any information from iteration to iteration. However, for global planning in a static map, this discards and replicates a large amount of sampling effort. Therefore, Probabilistic Roadmaps (PRMs) are suitable for many applications where the map remains fixed.

These approaches are similar to the topological graphs described below, in that they usually consist of two stages: building the roadmap, and then searching the roadmap for a solution. However, they suffer the same drawbacks as all probabilistically-optimal methods: it is not clear how much sampling is "enough" sampling, so it can only be decided by heuristics, and small openings or narrow corridors pose huge problems for PRM-based methods (as the chance of samples landing within them are small).

B. Topological Graphs

Our proposed method is a search through the sparse topological graph generated in Section V. Unlike PRM-based methods, ours is deterministic and has a fixed computation time. Additionally, since the graph is based on the structure of the ESDF, which already encodes the geometry of the scene, it does not suffer from narrow corridor openings. The downside to this fixed execution and search time is that while the graph will contain all topologically-distinct homotopies of the space, it is not guaranteed that the path length in the graph is the same as the shortest path length through the space. We perform graph shortening (described below) to attempt to overcome this issue, but it is possible that an incorrect homotope will be selected (though this can also be a problem in PRMs, depending on how the point distributions are sampled).

The method works as a two-stage search, starting from the sparse graph from Section V. We first find the nearest sparse graph vertex to the start and goal by using a pre-computed k-D tree of vertices. Then we find a path through the graph using A*. Due to the small size of the graph, this is a very fast operation, so it is possible to solve the problem for multiple start and goal vertices from the k-D tree to attempt to find a better solution. Finally, we plan from the start pose

[11] github.com/ethz-asl/mav_voxblox_planning

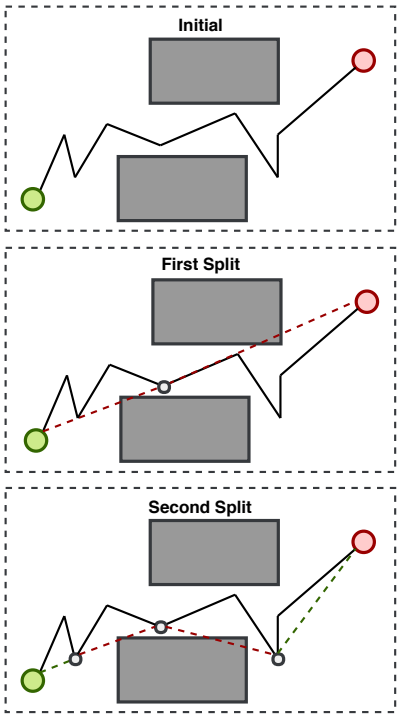


Fig. 9: First three steps of binary-search graph shortening. The path is iteratively split in half, until sub-paths are able to be shortened or no more splits are possible. Green shows successful shortened paths, while red shows invalid shorten attempts.

to the start vertex using A* in the ESDF, and likewise for the goal. Since these distances are always short, this is also not an expensive process.

1) Graph Shortening: While planning through the sparse topological graph is extremely fast, the waypoints it produces aim to maximize clearance, not necessarily minimize absolute path length (path length on the *graph* is minimized) through all traversable space. For instance, even a straight-line path from A to B would zig-zag along the maximum clearance lines in the graph.

To overcome this, we use iterative path shortening in the ESDF. We attempt to short-cut between pairs of waypoints on the initial graph path in a binary search manner, checking for traversability in the ESDF map, shown in Fig. 9.

We first try to shortcut directly from start to goal; if the straight-line path is not traversable, we then split the waypoint list into two halves: front to middle and middle to back. Each half is then iteratively checked, whether the intermediate vertices can simply be removed; if not, it is further split into two halves.

We perform this full splitting procedure multiple times to ensure that no further shortening is possible. This is similar to what the OMPL library does with the RRT-planned paths, with the important distinction that our method is deterministic, while theirs randomly tries to connect pairs of waypoints. This means that ours does not need heuristics to know when it is terminated: once no more changes are made, the waypoint list is as shortened as possible. The randomized approach requires options such as maximum steps and maximum empty steps (steps that do not shorten) before terminating, which means that it may still be possible to shorten the graph at termination.

VII. Path Smoothing

Path smoothing deals with taking a set of waypoints and converting them to a smooth, dynamically-feasible path. We present three methods we compare for these purposes: velocity ramp, polynomial, and our approach named Loco. We enforce dynamic constraints in the form of maximum velocity and acceleration limits.

A. Velocity Ramp

The simplest method is velocity ramp. A straight-line path is drawn between consecutive pairs of waypoints, and maximum acceleration is applied until the velocity limit is reached, at which point the acceleration is zero. The same principle is applied on decceleratng toward the next point.

The total time between two waypoints is described as:

$$t = \frac{v_{max}}{a_{max}} + \frac{\|\mathbf{x}_{goal} - \mathbf{x}_{start}\|}{v_{max}} \tag{1}$$

where t is the time in seconds, v_{max} and a_{max} are the velocity and acceleration constraints, respectively, and $\mathbf{x}_{start}$ and $\mathbf{x}_{goal}$ are the 3 DoF positions of the start and goal.

B. Polynomial

High-degree polynomial splines are a common representation for MAV trajectories, as they are easy to compute, can be smooth and continuous up to high derivatives, and are shown to be dynamically feasible as long as velocity and acceleration constraints are met [10], [28].

We implement a path smoothing method from Richter *et al.* [28], where a polynomial spline is fit to the waypoints and then iteratively split at collisions.

First, we discuss the optimization problem. We formulate it to minimize a high derivative such as jerk or snap, as shown to be desirable by Mellinger *et al.* [10].

We will consider a polynomial spline in K dimensions, with S segments, and each segment of order N. Each segment has K dimensions, each of which is described by an Nth order polynomial:

$$f_k(t) = a_0 + a_1 t + a_2 t^2 + a_3 t^3 \dots a_N t^N \tag{2}$$

with the polynomial coefficients:

$$\mathbf{p}_k = \begin{bmatrix} a_0 & a_1 & a_2 & \dots & a_N \end{bmatrix}^\top. \tag{3}$$

Rather than optimizing over the polynomial coefficients directly, which has numerical issues at high Ns, we instead optimize over the end-derivatives of segments within the spline [28]. We distinguish between fixed derivatives $\mathbf{d}_F$ (such as end-constraints) and free derivatives $\mathbf{d}_P$ (such as intermediate spline connections):

$$\mathbf{p} = \mathbf{A}^{-1}\mathbf{M}\begin{bmatrix}\mathbf{d}_F \\ \mathbf{d}_P\end{bmatrix}. \tag{4}$$

Where $\mathbf{A}$ is a mapping matrix from polynomial coefficients to end-derivatives, and $\mathbf{M}$ is a reordering matrix to separate $\mathbf{d}_F$ and $\mathbf{d}_P$.

We aim to minimize the derivative cost, J_d, which represents a certain derivative (often jerk or snap) of the position [10], with $\mathbf{R}$ as the augmented cost matrix.

$$\begin{aligned} J_d &= \mathbf{d}_F^\top \mathbf{R}_{FF}\mathbf{d}_F + \mathbf{d}_F^\top \mathbf{R}_{FP}\mathbf{d}_P + \\ &\quad \mathbf{d}_P^\top \mathbf{R}_{PF}\mathbf{d}_F + \mathbf{d}_P^\top \mathbf{R}_{PP}\mathbf{d}_P \end{aligned} \tag{5}$$

Finding the $\mathbf{d}_P^\star$ that minimized J_d is possible to do in closed-form [28]:

$$\mathbf{d}_P^\star = -\mathbf{R}_{PP}^{-1}\mathbf{R}_{FP}^\top \mathbf{d}_F \tag{6}$$

This method allows us to fit a smooth polynomial spline to a series of waypoints, by using the positions of the waypoints as vertex constraints.

However, since waypoint trajectories are planned such that the *straight-line* path (visibility graph) between them is collision-free, the smoothed path often runs into collision. To remedy this, any time a collision is detected, this method adds a new waypoint on the visibility graph closest to its projection onto the straight-line path, and the optimization is re-run.

While this is fast and easy to implement, this method suffers from occasionally not being able to escape collisions, and in difficult cases, creates many extra waypoints. The optimization problem does not scale well numerically when there are many waypoints, especially close together in time, as the segment times get very short (and must be raised to high powers). Furthermore, adding these additional waypoints often perturbs the trajectory in unexpected ways, causing the robot to take large detours.

C. Local Continuous Optimization (Loco)

To overcome these issues, we propose our own method, Local Continuous Optimization method, Loco[2]. Rather than iteratively collision-checking and splitting the trajectories, we introduce the collisions as soft costs in the optimization, following the general structure that Ratliff *et al.* [29] proposed in their CHOMP method.

Introducing this soft cost leads to the following optimization problem, where w terms are constant weights:

$$\mathbf{d}_P^* = \underset{\mathbf{d}_P}{\operatorname{argmin}}\ w_d J_d + w_c J_c \tag{7}$$

J_d remains as in (5) above, and we introduce a new term, J_c represents a soft collision cost:

$$J_c = \sum_{t=0}^{t_m} c(\mathbf{f}(t))\, \|\mathbf{v}(t)\|\, \Delta t \tag{8}$$

which approximates the line integral of costs along the path, where $c(\mathbf{x})$ is the collision cost from the map, $\mathbf{f}(t)$ is the position along the trajectory at time t, and $\mathbf{v}(t)$ is the velocity at time t.

For the collision cost in the map, we use a smooth gradually decreasing function proposed in CHOMP[29], where ϵ is a tuning value for how far outside the robot radius we care about collisions, $\mathbf{x}$ is a position in the map, and $d(\mathbf{x})$ is the ESDF distance at that point:

$$c(\mathbf{x}) = \begin{cases} -d(\mathbf{x}) + \frac{1}{2}\epsilon & \text{if } d(\mathbf{x}) < 0 \\ \frac{1}{2\epsilon}(d(\mathbf{x}) - \epsilon)^2 & \text{if } 0 \le d(\mathbf{x}) \le \epsilon \\ 0 & \text{otherwise} \end{cases} \tag{9}$$

In practice our robot radius is rarely 0, so we subtract the robot radius r from $d(\mathbf{x})$.

Now that we have a method of locally optimizing trajectories to be collision-free, evaluated at length in our previous work[2], the question is how to best make it fit through a series of waypoints.

We can use the starting method described above, where each waypoint simply becomes a control point/vertex in the spline. This means that for every waypoint, we have another segment in the spline. This has various downsides, most notably that for long, complex trajectories, the problem gets significantly slower computationally, and can have numerical scaling issues.

We found in practice that the optimization works best with a small (3-5) number of segments, therefore we explored methods to fit a visibility graph to these segments. The first solution was to generate an initial polynomial solution passing through all waypoints, and then re-sample it down to S segments, by selecting $S-1$ evenly-spaced times to sample the trajectory at. These then become the new waypoints. We will refer to this strategy as "polynomial resampling" in the results in Section VIII-C.

The second method is to instead sample directly on the visibility graph. Rather than sampling evenly-spaced ts on a polynomial trajectory, we instead fit a velocity ramp straight-line trajectory to a visibility graph, and sample the ts on this trajectory. This method is referred to as "visibility resampling" in the results.

As shown in Section VIII-C, both of these methods create better, higher-quality paths faster than simple waypoint fitting.

VIII. Evaluations

We attempt to validate our complete system, especially focusing on global planning and path smoothing on real data.

The local replanning is separately validated on both synthetic test-cases and in the real world in our previous work [2], [3].

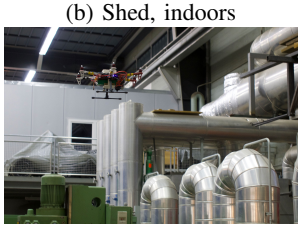
A. Evaluation Datasets

We focus our evaluations on real datasets, collected in typical scenarios for search and rescue and industrial inspection. We performed three inspection flights, two at a military search and rescue training ground at Wangen an der Aare, and one in the ETH Zürich Machine Hall. Photos of the three areas, named "shed", "rubble", and "Machine Hall" respectively, are shown in Fig. 10 and described in Table I. All datasets were collected with the MAV and sensor setup described in Section III-B, using both stereo and RGB-D (Intel RealSense D415) sensors.

(a) Shed, outdoors

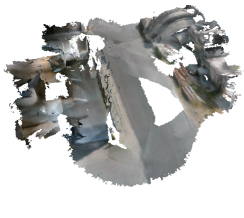
(b) Shed, indoors

(c) Rubble

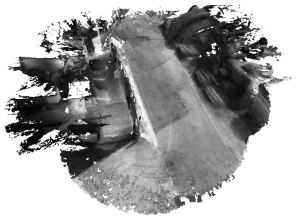


(d) Machine Hall

Fig. 10: Photos of the three scenarios from the benchmark dataset. (a) and (b) outdoor and indoor parts of the shed scene. (c): rubble scene. (d): machine hall scene.

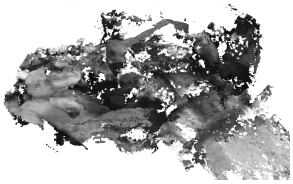


(a) Shed (RS)



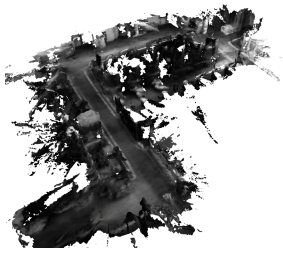
(b) Shed (Stereo)

(c) Rubble (RS)



(d) Rubble (Stereo)

(e) Machine Hall (RS)



(f) Machine Hall (Stereo)

Fig. 11: Colored meshes of the RealSense (RS) and stereo versions of the three maps used for benchmarking. Our stereo system is based on grayscale cameras, so no color data is available.

We make six global maps available: two per dataset, one using the stereo cameras and one using the RGB-D sensor (RS). All the maps are available for download[12].

The provided maps were generated with 10 cm voxels, 1 meter clear and 4 meter occupied spheres (described in Section IV-B), 8 meter maximum ray distance for TSDF construction, and 4 meter maximum ESDF computation distance.

We provide both a stereo and an RS map of all environments due to the narrow field of view (FoV) of the RealSense camera, but superior depth measurements (and color information). Fig. 12 shows the difference in traversability between the stereo and the RGB-D version of the shed dataset, assuming an 0.5 meter robot radius. For other datasets, such as machine hall, this is less important because the structure is much closer and therefore the narrow FoV makes little difference.

B. Sparse Topology Generation

To show that our sparse topology graph (or skeleton) is a feasible global planning strategy, we analyze the amount of time it takes to generate the sparse graph from an ESDF for each dataset. The results are shown in Fig. 13.

There is a large difference in the generation time between stereo and RealSense for the machine hall and rubble datasets, due to how much more space is traversable in the stereo datasets. However, most datasets are generated in 2 seconds or less, and the worst-case is 10 seconds. We consider this very feasible for a pre-processing step for global planning, as the actual planning times are orders of magnitude faster than other methods.

C. Loco

We analyze the effect of different waypoint fitting methods for Loco, as described in Section VII-C. We compare three methods: waypoint fitting (where each pair of waypoints has a segment between them), polynomial resampling (where an initial polynomial trajectory is fit with one segment through each waypoint, then resampled to a fixed number of waypoints), and visbility graph resampling (where intermediate waypoints are sampled directly off the visibility graph). The results are shown in Fig. 14, evaluated on the stereo shed dataset.

The visibility graph resampling has the highest success rate, and we use that variant as 'Loco' for the remaining evaluations.

D. Global Planning Benchmarks

To demonstrate the differences between different global planning methods, and how choice of path smoother affects the final result, we run 100 trials on each provided dataset. Each trial starts and ends at a random location, a minimum of 2 meters apart. The robot radius is 0.5 meters for all planners.

We use multiple global planning methods, summarized below:

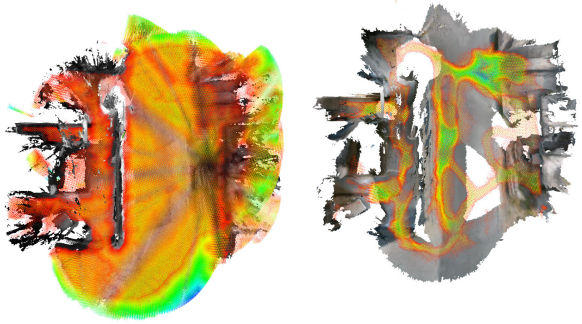
None Straight-line path between start and goal, no planning, meant to give an estimate of how many of the test cases have trivial solutions.

RRT Conn. RRT-Connect, which grows a bi-directional tree from and toward the goal. Very fast, run with an

[12] github.com/ethz-asl/mav_voxblox_planning

Dataset Name	Location	Flight Length	Volume	Contents
Shed	Wangen a. A, BE, CH	217 sec	38 m x 35 m x 12 m	Mixed indoor and outdoor dataset with narrow openings
Rubble	Wangen a. A, BE, CH	159 sec	28 m x 27 m x 12 m	Outdoor dataset, over earth-quake damaged buildings
Machine Hall	ETH Zürich, ZH, CH	251 sec	24 m x 30 m x 8 m	Indoor area, with large industrial machinery

TABLE I: Dataset statistics and descriptions.



(a) Shed (Stereo) (b) Shed (RS)

Fig. 12: Traversability differences between stereo datasets and RealSense datasets. Traversable space (given an 0.5 meter robot radius) is shown as colored points. As can be seen, much more space is considered traversable with stereo data.

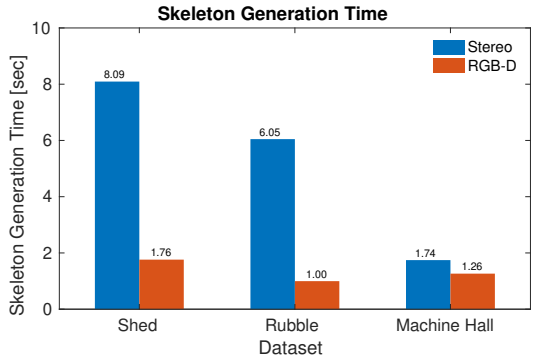


Fig. 13: Sparse topological (skeleton) graph generation timings for the test datasets. The stereo datasets generally take longer because there is more traversable space.

upper time bound of 1.0 sec (though terminates when first solution is found).

RRT* Probabilistically-optimal random planner, run for 2.0 seconds.

Skeleton Our sparse topological planner, using path shortening on the output path.

PRM Probabilistic roadmap, aimed to compare versus our skeleton-based method. The pre-planning stage is run for 2.0 seconds (to mirror the average dataset processing time of the sparse topology), and each planning query is given an additional 0.1 seconds.

Likewise, we test a variety of path smoothing methods:

No Smoothing Not an actual path smoothing method, just an indicator showing whether the global planner succeeded or not.

Velocity Ramp Velocity ramp method, always applying maximum or no acceleration. Follows straight-line paths between waypoints.

Polynomial The polynomial splitting approach of [28], described in sections above.

Loco Our local continuous trajectory optimizaton algorithm, run with visibility waypoint re-sampling, as determined from the previous sections to be the best.

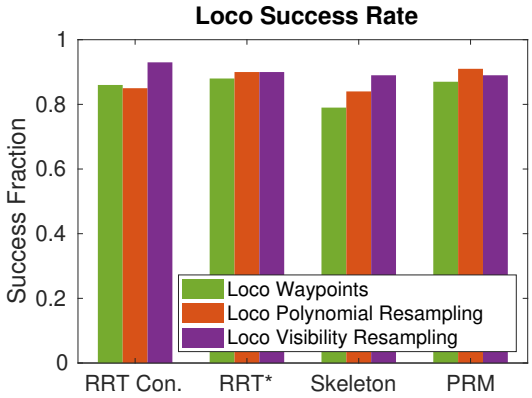


Fig. 14: A comparison of different waypoint fitting methods for Loco, and their success rate on the shed stereo dataset given different global planning methods. In general, visibility resampling has the highest success rate.

Fig. 15 shows a comparison of all described methods on the Machine Hall RealSense dataset. There are a number of take-aways from these results. When not using a global planner (i.e., attempting to draw a straight-line path between start and goal), only 13% of the test cases have trivial solutions, but Loco is able to solve 56% of problems with no global plan. In general, the success rate of Loco is also slightly higher, as it is able to better utilize the information in the map.

All the global planners are able to solve all of the planning problems. One key point to note is that the velocity ramp method does not work very well with the topological skeleton planner: this is because our graph simplification method contains some edges that do not lie perfectly on the straight-line. However, the Loco planner has a comparable success rate with the skeleton planner as other planners, again because it can follow gradient information in the map and slightly perturb the waypoints to produce a collision-free path.

The timings for a single typical trajectory on the Shed

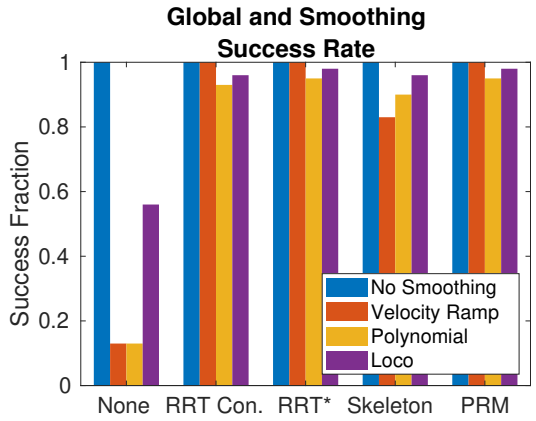


Fig. 15: Success rate of various global planner and path smoothing methods on the Machine Hall RealSense dataset, showing our method (Loco) is able to give smooth dynamically-feasible paths for more tests cases than competing methods.

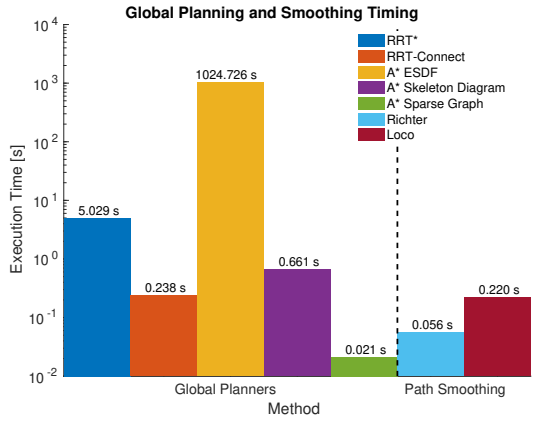


Fig. 16: Timings for various global and local planning methods. Note the log scale. As can be seen, skeleton planning is at least one order of magnitude faster than other global planning methods, and while loco timing is slightly slower than polynomial, it is still within bounds for fast global planning applications.

stereo dataset are shown in Fig. 16 (note the log scale). The topological skeleton planning method is 10x faster than even RRT Connect (and produces much shorter path lengths). Both the polynomial and loco smoothing methods are acceptable for fast global planning, though the polynomial method is approximately 2x faster.

IX. Conclusions

This paper presents a complete system for performing global planning, path smoothing, and local replanning. We extend on our previous work by improving a global planning sparse topology generation algorithm, suggest methods in which our local re-planning algorithm can also function for path smoothing, and benchmark a variety of global and path smoothing methods. Most importantly, we describe not only our mapping and planning approaches, but considerations that must be taken in other parts of the system for this approach to work, such as state estimation and controls, and make all of our code available online and open-source.

References

[1] H. Oleynikova, Z. Taylor, M. Fehr, R. Siegwart, and J. Nieto, "Voxblox: Incremental 3d euclidean signed distance fields for on-board mav planning," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, 2017.

[2] H. Oleynikova, M. Burri, Z. Taylor, J. Nieto, R. Siegwart, and E. Galceran, "Continuous-time trajectory optimization for online uav replanning," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, 2016.

[3] H. Oleynikova, Z. Taylor, R. Siegwart, and J. Nieto, "Safe local exploration for replanning in cluttered unknown environments for micro-aerial vehicles," *IEEE Robotics and Automation Letters*, 2018.

[4] H. Oleynikova, Z. Taylor, R. Siegwart, and J. Nieto, "Sparse 3d topological graphs for micro-aerial vehicle planning," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, 2018.

[5] Y. Lin, F. Gao, T. Qin, W. Gao, T. Liu, W. Wu, Z. Yang, and S. Shen, "Autonomous aerial navigation using monocular visual-inertial fusion," *Journal of Field Robotics (JFR)*, 2017.

[6] K. Mohta, M. Watterson, Y. Mulgaonkar, S. Liu, C. Qu, A. Makineni, K. Saulnier, K. Sun, A. Zhu, J. Delmerico, *et al.*, "Fast, autonomous flight in gps-denied and cluttered environments," *Journal of Field Robotics (JFR)*, vol. 35, no. 1, pp. 101–120, 2018.

[7] M. Burri, H. Oleynikova, M. W. Achtelik, and R. Siegwart, "Real-time visual-inertial mapping, re-localization and planning onboard mavs in unknown environments," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Sept 2015.

[8] J. Nikolic, J. Rehder, M. Burri, P. Gohl, S. Leutenegger, P. T. Furgale, and R. Siegwart, "A synchronized visual-inertial sensor system with fpga pre-processing for accurate real-time slam," in *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 431–437, IEEE, 2014.

[9] M. Kamel, J. Alonso-Mora, R. Siegwart, and J. Nieto, "Nonlinear model predictive control for multi-micro aerial vehicle robust collision avoidance," *arXiv preprint arXiv:1703.01164*, 2017.

[10] D. Mellinger and V. Kumar, "Minimum snap trajectory generation and control for quadrotors," in *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 2520–2525, IEEE, 2011.

[11] M. Bloesch, S. Omari, M. Hutter, and R. Siegwart, "Robust visual inertial odometry using a direct ekf-based approach," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 298–304, IEEE, 2015.

[12] M. Bloesch, M. Burri, S. Omari, M. Hutter, and R. Siegwart, "Iterated extended kalman filter based visual-inertial odometry using direct photometric feedback," *The International Journal of Robotics Research (IJRR)*, vol. 36, no. 10, pp. 1053–1072, 2017.

[13] S. Leutenegger, S. Lynen, M. Bosse, R. Siegwart, and P. Furgale, "Keyframe-based visual–inertial odometry using nonlinear optimization," *The International Journal of Robotics Research (IJRR)*, 2015.

[14] T. Qin, P. Li, and S. Shen, "Vins-mono: A robust and versatile monocular visual-inertial state estimator," *IEEE Transactions on Robotics*, vol. 34, no. 4, pp. 1004–1020, 2018.

[15] S. Lynen, M. W. Achtelik, S. Weiss, M. Chli, and R. Siegwart, "A robust and modular multi-sensor fusion approach applied to mav navigation," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, 2013.

[16] T. Schneider, M. Dymczyk, M. Fehr, K. Egger, S. Lynen, I. Gilitschenski, and R. Siegwart, "maplab: An open framework for research in visual-inertial mapping and localization," *IEEE Robotics and Automation Letters*, vol. 3, no. 3, pp. 1418–1425, 2018.

[17] R. Mur-Artal, J. M. M. Montiel, and J. D. Tardos, "Orb-slam: a versatile and accurate monocular slam system," *IEEE Transactions on Robotics*, vol. 31, no. 5, pp. 1147–1163, 2015.

[18] A. Millane, Z. Taylor, H. Oleynikova, J. Nieto, R. Siegwart, and C. Cadena, "C-blox: A scalable and consistent tsdf-based dense mapping approach," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, 2018.

[19] B. Curless and M. Levoy, "A volumetric method for building complex models from range images," in *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pp. 303–312, ACM, 1996.

[20] R. A. Newcombe, S. Izadi, O. Hilliges, D. Molyneaux, D. Kim, A. J. Davison, P. Kohi, J. Shotton, S. Hodges, and A. Fitzgibbon, "Kinectfusion: Real-time dense surface mapping and tracking," in *Mixed and augmented reality (ISMAR), 2011 10th IEEE international symposium on*, pp. 127–136, IEEE, 2011.

[21] H. Oleynikova, A. Millane, Z. Taylor, E. Galceran, J. Nieto, and R. Siegwart, "Signed distance fields: A natural representation for both mapping and planning," in *RSS Workshop on Geometry and Beyond*, 2016.

[22] B. Lau, C. Sprunk, and W. Burgard, "Improved updating of euclidean distance maps and voronoi diagrams," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, 2010.

[23] J. Chen, T. Liu, and S. Shen, "Online generation of collision-free trajectories for quadrotor flight in unknown cluttered environments," in *International Conference on Robotics and Automation (ICRA)*, IEEE, 2016.

[24] S. M. Lavalle, J. J. Kuffner, and Jr., "Rapidly-exploring random trees: Progress and prospects," in *Algorithmic and Computational Robotics: New Directions*, pp. 293–308, 2000.

[25] J. J. Kuffner and S. M. LaValle, "Rrt-connect: An efficient approach to single-query path planning," in *IEEE International Conference on Robotics and Automation (ICRA)*, vol. 2, pp. 995–1001, IEEE, 2000.

[26] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *The International Journal of Robotics Research (IJRR)*, vol. 30, no. 7, pp. 846–894, 2011.

[27] J. D. Gammell, S. S. Srinivasa, and T. D. Barfoot, "Informed rrt*: Optimal sampling-based path planning focused via direct sampling of an admissible ellipsoidal heuristic," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, 2014.

[28] C. Richter, A. Bry, and N. Roy, "Polynomial trajectory planning for aggressive quadrotor flight in dense indoor environments," in *Proceedings of the International Symposium on Robotics Research (ISRR)*, 2013.

[29] N. Ratliff, M. Zucker, J. A. Bagnell, and S. Srinivasa, "Chomp: Gradient optimization techniques for efficient motion planning," in *IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2009.