# Autonomous Exploration and Inspection Path Planning for Aerial Robots Using the Robot Operating System

**Chapter** *in* Studies in Computational Intelligence · January 2019

DOI: 10.1007/978-3-319-91590-6_3

CITATIONS
42

READS
539

**10 authors**, including:

Christos Papachristos
University of Nevada, Reno
**76** PUBLICATIONS **1,275** CITATIONS
SEE PROFILE

Mina Samir Kamel
ETH Zurich
**44** PUBLICATIONS **1,585** CITATIONS
SEE PROFILE

Marija Popovic
Imperial College London
**39** PUBLICATIONS **570** CITATIONS
SEE PROFILE

Shehryar Khattak
ETH Zurich
**34** PUBLICATIONS **504** CITATIONS
SEE PROFILE

# Autonomous Exploration and Inspection Path Planning for Aerial Robots Using the Robot Operating System


**Christos Papachristos, Mina Kamel, Marija Popović, Shehryar Khattak, Andreas Bircher, Helen Oleynikova, Tung Dang, Frank Mascarich, Kostas Alexis and Roland Siegwart**

**Abstract** This use case chapter presents a set of algorithms for the problems of autonomous exploration, terrain monitoring and optimized inspection path planning using aerial robots. The autonomous exploration algorithms described employ a receding horizon structure to iteratively derive the action that the robot should take to optimally explore its environment when no prior map is available, with the extension to localization uncertainty–aware planning. Terrain monitoring is tackled

C. Papachristos · S. Khattak · T. Dang · F. Mascarich · K. Alexis (✉)
Autonomous Robots Lab, University of Nevada, 1664 N. Virginia St.,
Reno, NV 89557, USA
e-mail: kalexis@unr.edu

C. Papachristos
e-mail: cpapachristos@unr.edu

S. Khattak
e-mail: shehryar.khattak@nevada.unr.edu

T. Dang
e-mail: tung.dang@nevada.unr.edu

F. Mascarich
e-mail: fmascarich@nevada.unr.edu

M. Kamel · M. Popović · A. Bircher · H. Oleynikova · R. Siegwart
Autonomous System Lab, ETH Zurich, Leonhardstrasse 21,
8092 Zurich, Switzerland
e-mail: fmina@ethz.ch

M. Popović
e-mail: mpopovic@ethz.ch

A. Bircher
e-mail: abircher@ethz.ch; andreas.bircher@mavt.ethz.ch

H. Oleynikova
e-mail: holeynikova@ethz.ch; elena.oleynikova@mavt.ethz.ch

R. Siegwart
e-mail: rsiegwart@ethz.ch



A. Koubaa (ed.), *Robot Operating System (ROS)*, Studies in Computational Intelligence 778, https://doi.org/10.1007/978-3-319-91590-6_3

by a finite–horizon informative planning algorithm that further respects time budget limitations. For the problem of optimized inspection with a model of the environment known a priori, an offline path planning algorithm is proposed. All methods proposed are characterized by computational efficiency and have been tested thoroughly via multiple experiments. The Robot Operating System corresponds to the common middleware for the outlined family of methods. By the end of this chapter, the reader should be able to use the open–source contributions of the algorithms presented, implement them from scratch, or modify them to further fit the needs of a particular autonomous exploration, terrain monitoring, or structural inspection mission using aerial robots. Four different open–source ROS packages (compatible with ROS Indigo, Jade and Kinetic) are released, while the repository https://github.com/unr-arl/informative-planning stands as a single point of reference for all of them.

## 1 Introduction

Autonomous inspection, exploration, and mapping correspond to main and critical use cases for aerial robots. In a multitude of application domains, including infrastructure and industrial inspection [1], precision agriculture [2, 3], search and rescue [4], and security surveillance [5], such capabilities are enabling wide utilization of autonomous aerial systems. Their use leads to enhanced safety for human personnel, major cost savings, improved operation of the inspected facilities, enhanced security, and more. Recent advances in onboard sensing and processing, alongside a set of algorithmic contributions relating to the problems of Simultaneous Localization And Mapping (SLAM) and path planning, pave the way for aerial robots to conduct the aforementioned tasks autonomously, reliably, and efficiently.

In this chapter, we present a family of algorithmic contributions that address the problems of:

- autonomous area exploration and mapping when no prior knowledge of the environment is available,
- informative path planning for terrain monitoring,
- full coverage inspection planning provided a prior model of the environment.

For the problem of autonomous unknown area exploration, iterative online receding horizon algorithms are discussed that further account for the localization uncertainty of the robot. The terrain monitoring problem is addressed through a finite–horizon planner that optimizes for information gain subject to endurance constraints. For the problem of optimized coverage given a model of the environment, efficient derivation of an inspection path offline is presented. For all three cases, the methods account for the vehicle constraints and motion model.

Each method is presented in regards to the algorithmic concepts, key implementation details, user interface and developer information for the associated open–source contributions by our team, alongside relevant experimental data. By the end of the chapter, the reader should be able to implement and test these algorithms onboard real

aerial robots or in simulation. The autonomous exploration, terrain monitoring, and inspection algorithms are open–sourced as Robot Operating System (ROS) packages and documented to enable and accelerate further development. For all algorithms, various implementation hints and practical suggestions are provided in this chapter. The methods have been experimentally verified and field demonstrated. The presented methods have enabled: (a) the autonomous exploration of railway and city tunnels in visually–degraded conditions (darkness, haze), (b) the mapping of a national park, (c) the exploration of several office room-like environments, (d) the monitoring of a mock–up relevant to precision agriculture tasks, and more. It is critical to highlight that ROS is not only the environment in which the planners are implemented but also the enabling factor for the presented experiments. The core loops of localization and mapping, as well as position control in all the described experiments are also ROS-based.

Section 2 presents the exact problem statements for each planning scenario above. Sections 3, 4 and 5 detail the algorithms for autonomous exploration, terrain monitoring, and inspection respectively and present indicative evaluation results. Finally, Section 6 details the open–source code contributions, followed by conclusions in Sect. 7.

## 2 Problem Definition

This section outlines the exact definitions for the three informative planning scenarios discussed in this chapter: unknown area exploration, terrain monitoring, and inspection planning. In total four algorithms are presented, namely the "Receding Horizon Next-Best-View Planner" (`nbvplanner`), "Localization Uncertainty–aware Receding Horizon Exploration and Mapping Planner" (`rhemplanner`), "Terrain Monitoring Planner" (`tmplanner`) and "Structural Inspection Planner" (`siplanner`). The associated problem statements are provided below.

### *2.1 Exploration Path Planning*

The autonomous unknown area exploration problem considered in this chapter is that of exploring a bounded 3D volume $V^E \subset \mathbb{R}^3$ while possibly aiming to minimize the localization and mapping uncertainty as evaluated through a metric over the probabilistic belief of robot's pose and landmark positions. The exploration problem may be casted globally as that of starting from an initial collision free configuration and deriving viewpoints that cover the a priori unknown volume by determining which parts of the initially unexplored space $V^E_{une} \overset{init.}{=} V^E$ are free $V^E_{free} \subseteq V^E$ or occupied $V^E_{occ} \subseteq V^E$. For this process, the volume is discretized in an occupancy map $\mathcal{M}$ consisting of cubical voxels $m \in \mathcal{M}$ with edge length $r$. The operation is

subject to vehicle dynamic constraints and limitations of the sensing system. As the perceptive capabilities of most sensors stop at surfaces, hollow spaces or narrow pockets can sometimes remain unexplored, leading to a residual volume:

**Definition 1** (*Residual Volume*) Let $\Xi$ be the simply connected set of collision free configurations and $\bar{\mathcal{V}}_m^E \subseteq \Xi$ the set of all configurations from which the voxel $m$ can be perceived. Then the residual volume is given by $V_{res}^E = \bigcup_{m \in \mathcal{M}} (m | \, \bar{\mathcal{V}}_m^E = \emptyset)$.

The exploration problem is globally defined as:

**Problem 1** (*Volumetric Exploration Problem*) Given a bounded volume $V^E$, find a collision free path $\sigma$ starting at an initial configuration $\xi_{init} \in \Xi$ that leads to identifying the free and occupied parts $V_{free}^E$ and $V_{occ}^E$ when being executed, such that there does not exist any collision free configuration from which any piece of $V^E \setminus \{V_{free}^E, V_{occ}^E\}$ could be perceived. Thus, $V_{free}^E \cup V_{occ}^E = V^E \setminus V_{res}^E$.

While the exploration problem is globally defined, the optional sub–problem of localization and mapping uncertainty–aware planning in a priori unknown environments can only be casted locally as new features are tracked from the perception system. At every planning step, let $V^M \subset V^E$ be a local volume enclosing the current pose configuration and the next exploration viewpoint. The problem is that of minimizing the covariance of the propagated robot pose and tracked landmarks belief as the vehicle moves from its current configuration to the next exploration viewpoint. By minimizing the robot belief uncertainty, a good prior for high–quality mapping is achieved. More formally:

**Problem 2** (*Belief Uncertainty–aware Planning*) Given a $V^M \subset V^E$, find a collision free path $\sigma^M$ starting at an initial configuration $\xi_0 \in \Xi$ and ending in a configuration $\xi_{final}$ that aims to maximize the robot's localization and mapping confidence by following paths of its optimized pose and covariance of tracked landmarks.

## 2.2 Path Planning for Terrain Monitoring

This chapter considers the problem of informative path planning for monitoring a flat terrain using an aerial robot equipped with an altitude-dependent sensor. The environment where measurements are to be taken is represented as a 2D plane $\mathcal{E} \subset \mathbb{R}^2$. The objective is to maximize the informative data collected within $\mathcal{E}$ while respecting a time budget. Thus, we seek a continuous trajectory $\psi$ in the 3D space of all trajectories $\Psi$ above the terrain for maximum gain in some information-theoretic measure:

$$\psi^* = \underset{\psi \in \Psi}{\operatorname{argmax}} \frac{I[\text{MEASURE}(\psi)]}{\text{TIME}(\psi)}, \quad \text{s.t. TIME}(\psi) \leq B, \tag{1}$$

where $B$ denotes a time budget and $I[\cdot]$ defines the utility function quantifying the informative objective. The function MEASURE$(\cdot)$ obtains a finite set of measurements along the trajectory $\psi$ and TIME$(\cdot)$ provides the corresponding travel time.

In Eq. 1, $I[\cdot]$ is formulated generically and can be defined to capture situation-specific interests with respect to the underlying terrain map representation (Sect. 4.3.3), e.g. entropy minimization. In this chapter, we study two different mapping scenarios involving (i) discrete and (ii) continuous target variables, as relevant for practical applications.

### *2.3 Inspection Path Planning*

The problem of structural inspection path planning is considered in this chapter as that of finding a high quality path that guarantees the complete coverage of a given 3D structure $\mathcal{S}$ subject to dynamic constraints of the vehicle and limitations of its on–board sensor system. The 3D structure to be inspected may be represented in a computationally efficient way, such as a triangular mesh or voxel-based octomap, and is embedded in a bounded environment that could contain obstacle regions. The problem setup is to be such that that for each triangle in the mesh, there exists an *admissible* viewpoint configuration – that is, a viewpoint from which the triangle is visible for a specific sensor model. Then, for the given environment and with respect to the operational constraints, the aim is to find a path connecting all viewpoints which guarantees complete inspection of the 3D structure. Quality measures for paths are situation-specific, depending on the system and mission objectives, e.g. short time or distance.

## 3 Autonomous Unknown Area Exploration

This section overviews and details a set of sampling–based receding horizon algorithms enabling the autonomous exploration of unknown environments. Both methodologies, the "Receding Horizon Next-Best-View planner" (`nbvplanner`) and "Localization Uncertainty–aware Receding Horizon Exploration and Mapping planner" (`rhemplanner`), exploit a mechanism that identifies the next–best–view in terms of exploration gain along a finite–depth trajectory. The first viewpoint of this trajectory is then visited, while the whole process is repeated iteratively. The distinctive feature of the second algorithm is that it identifies the best path to go to this new viewpoint, such that the on–board localization uncertainty of the robot is minimized.

### 3.1 Receding Horizon Next–Best–View Planner

Considering the exploration of unknown space, the presented `nbvplanner` employs a sampling–based receding horizon path planning paradigm to generate paths that cover the whole, initially unknown but bounded, space $V^E$. A sensing system that perceives a subset of the environment and delivers depth/volumetric data (e.g. stereo camera, a LiDAR or a time-of-flight 3D camera) is employed to provide feedback on the explored space. All acquired information is combined in a map $\mathcal{M}$ representing an estimate of the current world state. This map is used for both collision–free navigation and determination of the exploration progress.

The representation of the environment is an occupancy map [6] dividing space $V^E$ in cubical volumes $m \in \mathcal{M}$, that can either be marked as free, occupied or unmapped. For every voxel $m$ in this set lies in the unexplored area $V^E_{unm}$, the direct Line of Sight (LoS) does not cross occupied voxels and complies with the sensor model. Within this work, the camera sensor is mounted with a fixed pitch, has a Field of View (FoV) described by vertical and horizontal opening angles $[a_v, a_h]$, and a maximum effective sensing distance. While the sensor limitation is $d^{\text{sensor}}_{\max}$, the planner uses a value $d^{\text{planner}}_{\max} < d^{\text{sensor}}_{\max}$. The resulting array of voxels is saved in an octomap, enabling computationally efficient access for operations like integrating new sensor data or checking for occupancy. Paths are only planned through known free space $V_{free}$, thus inherently providing collision–free navigation. While a collision–free vehicle configuration is denoted by $\xi \in \Xi$, a path is given by $\sigma^E : \mathbb{R} \to \xi$, specifically from $\xi_{k-1}$ to $\xi_k$ by $\sigma^k_{k-1}(s)$, $s \in [0, 1]$ with $\sigma^k_{k-1}(0) = \xi_{k-1}$ and $\sigma^k_{k-1}(1) = \xi_k$. The path has to be such that it is collision–free and can be tracked by the vehicle, considering its dynamic and kinematic constraints. The corresponding path cost (distance) is $c(\sigma^k_{k-1})$. For a given occupancy map representing the world $\mathcal{M}$, the set of visible and unmapped voxels from configuration $\xi$ is denoted as $\textbf{Visible}(\mathcal{M}, \xi)$. Every voxel $m \in \textbf{Visible}(\mathcal{M}, \xi)$ lies in the unmapped exploration area $V_{unm}$, the direct LoS does not cross occupied voxels and complies with the sensor model (in terms of FoV, maximum effective perception, and depth sensing range). Starting from the current vehicle configuration $\xi_0$, a geometric random tree (RRT [7]) $\mathbb{T}^E$ is incrementally built in the configuration space. The resulting tree contains $N^E_{\mathbb{T}}$ nodes $n^E$ and its edges are given by collision–free paths $\sigma^E$, connecting the configurations of the two nodes. The quality - or collected information gain - of a node $\textbf{Gain}(n)$ is the summation of the unmapped volume that can be explored at the nodes along the branch, e.g. for node $k$, according to

$$\textbf{ExplorationGain}(n^E_k) = \textbf{ExplorationGain}(n^E_{k-1}) + \textbf{VisibleVolume}(\mathcal{M}, \xi_k) \exp(-\lambda c(\sigma^E_{k-1,k}))$$

with the tuning factor $\lambda$ penalizing high path costs and therefore indirectly giving preference to shorter paths as long as sufficient information–gathering is achieved [8].

After every replanning iteration, the robot executes the first segment of the branch to the best node as evaluated by $\textbf{ExtractBestPathSegment}(n_{best})$, where $n_{best}$ is the node with the highest gain. Generally, tree construction is stopped when $N_{\mathbb{T}^E} = N^E_{\max}$, but if the best gain $g_{best}$ of $n_{best}$ remains zero, this process continues until $g_{best} > 0$.

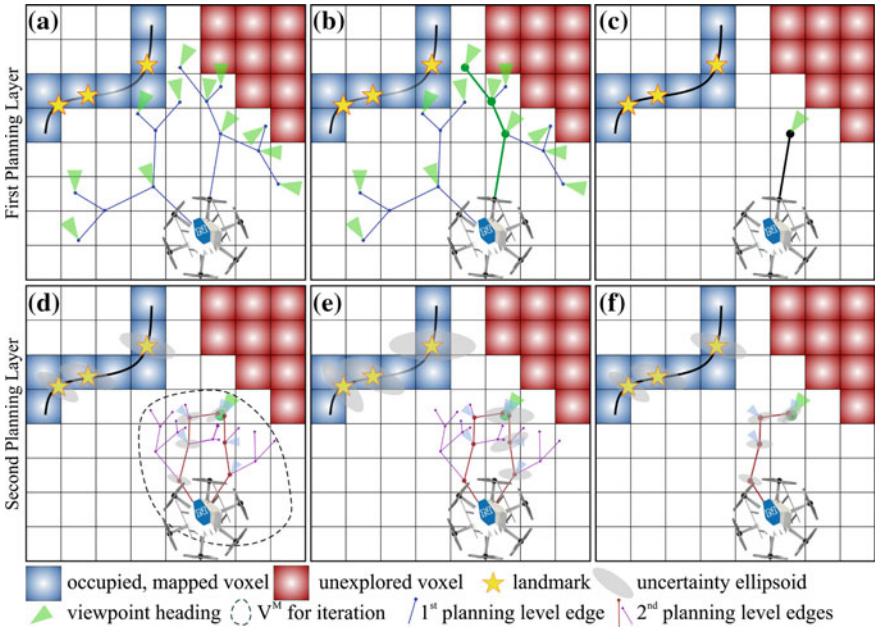


**Fig. 1** 2D representation of the two–step uncertainty–aware exploration and mapping planner. The first planning layer samples the path with the maximum information gain in terms of space to be explored. This also corresponds to a single iteration of the `nbvplanner`. The viewpoint configuration of the first vertex of this path becomes the reference to the second planning layer. This step samples admissible paths that arrive to this configuration and selects the one that provides minimum uncertainty of the robot belief about its pose and the tracked landmarks. To achieve this, the belief is propagated along the candidate edges. It is noted that the second step is only executed by the uncertainty–aware planner

For practical purposes, a tolerance value $N_{TOL}$ is chosen significantly higher than $N_{\max}$ and the exploration step is considered solved when $N_{\mathbb{T}}^{E} = N_{TOL}^{E}$ is reached while $g_{best}$ remains zero. Algorithm 1 and Fig. 1 summarize and illustrate the planning procedure.

### 3.1.1 Indicative Evaluation Study

A simulation–based evaluation study is presented to demonstrate intermediate steps of the exploration process commanded by the `nbvplanner`. Since the planner works in a closed loop with the robot's perception, estimation, and control loops, a detailed and realistic simulation is required. The Gazebo–based simulation environment RotorS has been used along with the provided model of the AscTec Firefly hexacopter MAV. It employs a stereo camera that gives real time depth images of the simulated environment. Its specifications are a FoV of $[60, 90]^{\circ}$ in vertical and horizontal direction, respectively, while it is mounted with a downward pitch angle of

15°. For all simulations, the size of the box model for collision detection is assumed as $0.5 \times 0.5 \times 0.3$ m.

**Algorithm 1** `nbvplanner` - Iterative Step

$\xi_0 \leftarrow$ current vehicle configuration
Initialize $\mathbb{T}^E$ with $\xi_0$ and, unless first planner call, also previous best branch
$g_{best} \leftarrow 0$ ▷ Set best gain to zero.
$n_{best} \leftarrow n_0^E(\xi_0)$ ▷ Set best node to root.
$N_{\mathbb{T}}^E \leftarrow$ Number of initial nodes in $\mathbb{T}^E$
**while** $N_{\mathbb{T}}^E < N_{\max}^E$ **or** $g_{best} = 0$ **do**
  Incrementally build $\mathbb{T}^E$ by adding $n_{new}^E(\xi_{new})$
  $N_{\mathbb{T}}^E \leftarrow N_{\mathbb{T}}^E + 1$
  **if ExplorationGain**$(n_{new}^E) > g_{best}$ **then**
    $n_{best} \leftarrow n_{new}^E$
    $g_{best} \leftarrow$ **ExplorationGain**$(n_{new}^E)$
  **end if**
  **if** $N_{\mathbb{T}}^E > N_{TOL}^E$ **then**
    Terminate exploration
  **end if**
**end while**
$\sigma^E \leftarrow$ **ExtractBestPathSegment**$(n_{best})$
Delete $\mathbb{T}^E$
**return** $\sigma^E$

The simulation scenario considers an indoor room–like environment involving significant geometric complexity and, specifically, a room with a very narrow corridor entrance. Figure 2 visualizes the environment and instances of the exploration mapping based on the reconstructed occupancy map.

The open–sourced ROS package of `nbvplanner` further supports multi–robot systems. In Fig. 3, an additional result is presented to demonstrate the significant improvement in terms of exploration time when appropriately pre–distributed multiple agents are used. Note that the code release currently assumes that each vehicle explores independently with only local knowledge about its environment, and can be improved by integrating more complex collaborative strategies.

A set of experimental studies are presented in [9–11] and relate to experimental evaluation in indoor–like environments of varying geometric complexity. The video in https://youtu.be/D6uVejyMea4 is indicative of the planner experimental operation.

### *3.2 Localization Uncertainty–Aware Receding Horizon Exploration and Mapping Planner*

The abovementioned planning paradigm ensures efficient exploration of unknown volume but neglects the fact that robots often operate subject to uncertain localization and mapping. In fact, especially when GPS–denied operation is considered, the view-

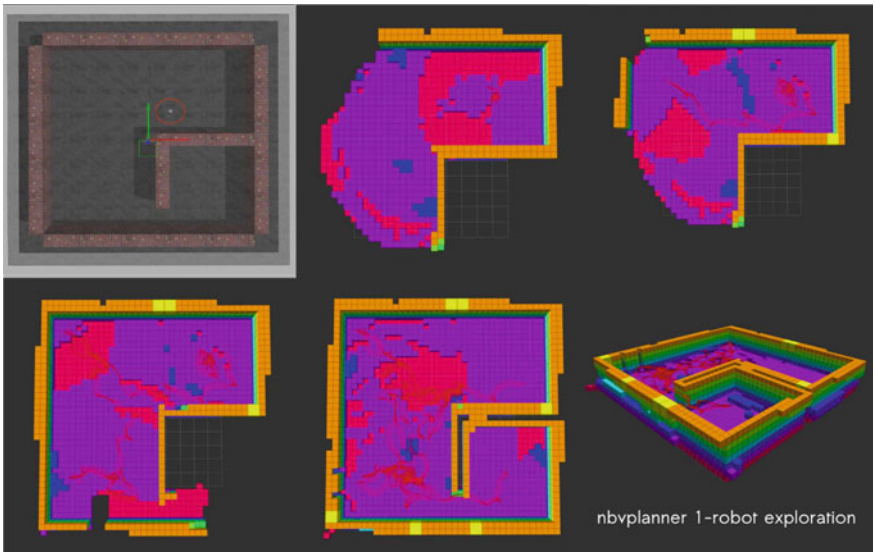


**Fig. 2** Single robot exploration of a room–like environment with challenging geometry (narrow corridor to secondary room) using the `nbvplanner`. Incremental steps of exploration are presented while the identical simulation scenario (in terms of environment and robot configuration - paths will be different due to the randomized nature of the solution) can be executed using the open–source ROS package for this planner

point and trajectory selection mechanisms not only influence the exploration process but also the accuracy of the robot pose estimation and map quality. To address this problem, the receding horizon exploration paradigm was extended into a second algorithm accounting for the propagated robot belief regarding its localization uncertainty. This new algorithm, called `rhemplanner`, employs a two–step sampling–based receding horizon planning paradigm to iteratively generate paths that can cover the whole space $V^E$, while locally aiming to minimize the localization uncertainty of the robot in terms of the covariance of its pose and tracked landmarks. The first planning step replicates the strategy of the `nbvplanner` and identifies a finite–steps path that maximally explores new, previously unmapped space. Subsequently, the first new viewpoint configuration of this path becomes the goal point of a second, nested, planning step that computes a new path to go to this viewpoint while ensuring that low localization uncertainty belief is maintained. The robot follows this path and the entire process is iterated. A visualization of this procedure is shown in Fig. 1.

A sensor, e.g. a camera or a stereo visual–inertial unit, as in the presented experimental studies, is used to enable robot localization and map building. More specifically, estimates for robot pose and tracked landmarks as well as the associated covariance matrix are provided, along with a volumetric map of the environment derived using the stereo depth sensing and the estimated robot poses. Any other sensing system that provides landmarks–based localization and a depth estimate can

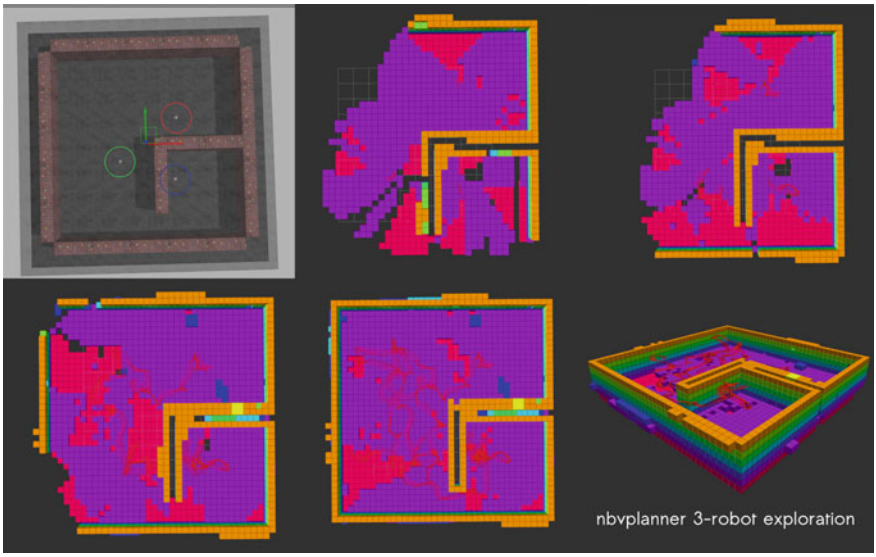


**Fig. 3** Multi robot exploration of a room–like environment with challenging geometry (narrow corridor to secondary room) using the `nbvplanner`. Incremental steps of exploration are presented while the identical simulation scenario (in terms of environment and robot configuration - paths will be different due to the randomized nature of the solution) can be executed using the open–source ROS package for this planner. Although the support for for multi–agent systems in the code release aims to support the community of multi–robot collaboration, it also demonstrates the significant savings in time if multiple and appropriately pre–distributed robotic systems are employed for the same environment

also be used without any change in the planner functionality. The robot belief and the volumetric map are used for collision–free navigation and exploration path planning.

The same volumetric representation of an occupancy map $\mathcal{M}$ [6] (implemented in our system with the support of ROS) dividing the space $V^E$ in cubical volumes $m \in \mathcal{M}$ is employed. Each voxel is marked as free or occupied with a probability $\mathcal{P}(m)$ or unmapped. Similarly, for planning purposes, a robot configuration is defined by the state $\xi = [x, y, z, \psi]^T$ with roll ($\phi$) and pitch ($\theta$) considered to be close to zero. The following details the role of each planning step.

### *3.3 Exploration Planning*

At each iteration, for the map representing the world $\mathcal{M}$, the set of visible but unmapped voxels given a robot configuration $\xi$ is denoted as **VisibleVolume**$(\mathcal{M}, \xi)$. Beyond the volumetric exploration gain as in the case of the `nbvplanner`, this extended planner also considers the probability of a voxel being occupied $\mathcal{P}(m) < p_{thres}$ and calculates the **ReobservationGain**$(\mathcal{M}, \mathcal{P}, \xi)$ which refers to

the voxels' volume weighted by $(1 - \mathcal{P}(m))$. Starting from the current configuration $\xi_0$, a RRT $\mathbb{T}^E$ with maximum edge length $\ell_E$ is incrementally built. The resulting tree contains $N_{\mathbb{T}}^E$ nodes $n^E$ and its edges are given by collision–free paths $\sigma^E$. A minimum $N_{\max}^E$ and maximum $N_{TOT}^E$ threshold on the amount of nodes sampled are set. The collected information gain of a node **ExplorationGain**$(n^E)$ is the cumulative unmapped volume that can be explored from the nodes along the branch, added to the **ReobservationGain**$(\mathcal{M}, \mathcal{P}, \xi)$. For node $k$:

$$\begin{aligned} \textbf{ExplorationGain}(n_k^E) = \textbf{ExplorationGain}(n_{k-1}^E) + \\ \textbf{VisibleVolume}(\mathcal{M}, \xi_k) \exp(-\lambda c(\sigma_{k-1,k}^E)) + \\ \textbf{ReobservationGain}(\mathcal{M}, \mathcal{P}, \xi_k) \exp(-\lambda c(\sigma_{k-1,k}^E)) \end{aligned} \tag{2}$$

At the end of this planning level iteration, the first segment $\sigma_{RH}^E$ of the branch to the best node **ExtractBestPathSegment**$(n_{best}^E)$, the vertex at the end of this segment $n_{RH}^E$, and the associated pose configuration $\xi_{RH}$ are extracted. If no positive gain can be found, the exploration process is terminated.

### 3.4 Localization Uncertainty–Aware Planning

The second, nested, planning step serves to minimize the localization uncertainty of the robot and tracked landmarks uncertainty given that the vehicle is equipped with a relevant sensor system and algorithms. At every call of this nested planning level, a new RRT $\mathbb{T}^M$ with maximum edge length $\ell_M$ is spanned within a continuous local volume $V^M \subset V^E$ that includes the current robot pose and $\xi_{RH}$. A total set of $N_{\mathbb{T}}^M$ vertices are sampled with $N_{\mathbb{T}}^M < N_{\max}^M$. This allows the derivation of admissible paths (with $\sigma_{RH}^E$ treated as one of them) that (a) start from the current robot configuration and arrive in a local set $\mathbb{S}_{\xi_{RH}}$, around $\xi_{RH}$ provided from the first exploration planning layer, and (b) have an overall length $c \leq \delta c(\sigma_{RH}^E)$ where $\delta \geq 1$ a tuning factor. Since an admissible path is one that arrives into the local set $\mathbb{S}_{\xi_{RH}}$ $(\mathbb{V}(\mathbb{T}^M) \cap \mathbb{S}_{\xi_{RH}} \neq \emptyset)$ and not necessarily on $\xi_{RH}$ an additional connection takes place to guarantee that the robot arrives exactly on the configuration sampled by the previous planning layer. For each of the $N_{\mathbb{T}}^M - 1$ edges of the tree (including the additional connections to $n_{RH}$), belief propagation takes place to estimate the expected robot belief about its state and the tracked landmarks along the sampled paths. Given the updated estimates of the state and landmarks covariance, the admissible tree branches are evaluated regarding their "belief gain" measured using the D-optimality [12] operation over the robot pose and tracked landmarks covariance matrix at the end–vertex of each admissible path. The admissible branch with the best belief gain is then selected to be executed by the robot. Algorithm 2 details the steps of the complete `nbvplanner` process, while Sects. 3.4.1–3.4.3 detail the localization framework employed from a generalized perspective, the belief propagation, and the belief gain calculations, respectively.

### 3.4.1 Visual–Inertial Localization

For the purposes of GPS–denied robot navigation, a visual–inertial odometry framework is employed due to the high robustness and accuracy provided by these methods. In particular, the open–source and ROS-based Robust Visual Inertial Odometry (ROVIO) is utilized [13]. Below, a necessarily very brief summary will be provided as its formulation is also used for belief propagation. It is important that the par-

**Algorithm 2** `rhemplanner` - Iterative Step

$\xi_0 \leftarrow$ current vehicle configuration
Initialize $\mathbb{T}^E$ with $\xi_0$
$g^E_{best} \leftarrow 0$ ▷ Set best exploration gain to zero.
$n^E_{best} \leftarrow n_0(\xi_0)$ ▷ Set best exploration node to root.
$N^E_{\mathbb{T}} \leftarrow$ Number of initial nodes in $\mathbb{T}^E$
**while** $N^E_{\mathbb{T}} < N^E_{\max}$ **or** $g^E_{best} = 0$ **do**
  Incrementally build $\mathbb{T}^E$ by adding $n^E_{new}(\xi_{new})$
  $N^E_{\mathbb{T}} \leftarrow N^E_{\mathbb{T}} + 1$
  **if ExplorationGain**$(n^E_{new}) > g^E_{best}$ **then**
    $n^E_{best} \leftarrow n^E_{new}$
    $g^E_{best} \leftarrow$ **ExplorationGain**$(n^E_{new})$
  **end if**
  **if** $N^E_{\mathbb{T}} > N^E_{TOL}$ **then**
    Terminate planning
  **end if**
**end while**
$\sigma^E_{RH}, n^E_{RH}, \xi_{RH} \leftarrow$ **ExtractBestPathSegment**$(n^E_{best})$
$\mathbb{S}_{\xi_{RH}} \leftarrow$ **LocalSet**$(\xi_{RH})$
Propagate robot belief along $\sigma^E_{RH}$
$\alpha \leftarrow 1$ ▷ number of admissible paths
$g^M_\alpha \leftarrow$ **BeliefGain**$(\sigma^E_{RH})$
$g^M_{best} \leftarrow g^M_\alpha$ ▷ straight path belief gain
$\sigma^M_{best} \leftarrow \sigma^M_{RH}$ ▷ Set best belief path.
**while** $N^M_{\mathbb{T}} < N^M_{\max}$ **or** $\mathbb{V}(\mathbb{T}^M) \cap \mathbb{S}_{\xi_{RH}} = \emptyset$ **do**
  Incrementally build $\mathbb{T}^M$ by adding $n^M_{new}(\xi_{new})$
  Propagate robot belief from current to planned vertex
  **if** $\xi_{new} \in \mathbb{S}_{\xi_{RH}}$ **then**
    Add new vertex $n^M_{new}$ at $\xi_{RH}$ and connect
    $\alpha \leftarrow \alpha + 1$
    $\sigma^M_\alpha \leftarrow$ **ExtractBranch**$(n^M_{new})$
    $g^M_\alpha \leftarrow$ **BeliefGain**$(\sigma^M_\alpha)$
    **if** $g^M_\alpha < g^M_{best}$ **then**
      $\sigma^M \leftarrow \sigma^M_\alpha$
      $g^M_{best} \leftarrow g^M_\alpha$
    **end if**
  **end if**
**end while**
**return** $\sigma^M$

ticular visual–inertial system does not have to be utilized for the actual localization of the robot: as long as a landmarks–based localization system is considered and the expected robot pose and the covariance of its pose and tracked landmarks are calculated, then those can be provided to the algorithm without any modification requirement. ROVIO tracks referringmultilevel image patches. The estimated landmarks are decomposed into a bearing vector, as well as a depth parametrization. The Inertial Measurement Unit (IMU) fixed coordinate frame ($\mathcal{B}$), the camera fixed frame ($\mathcal{V}$) and the inertial frame ($\mathcal{W}$) are considered and the employed state vector with dimension $l$ and associated covariance matrix $\Sigma_l$ is:

$$\mathbf{x} = [\overbrace{\mathbf{r}\ \mathbf{q}}^{\text{pose},}\ l_p \boldsymbol{\upsilon}\ \mathbf{b}_f\ \mathbf{b}_\omega\ \mathbf{c}\ \mathbf{z}\ \underbrace{\Big|\ \boldsymbol{\mu}_0,\ \cdots\ \boldsymbol{\mu}_J\ \rho_0\ \cdots\ \rho_J}_{\text{features states},l_f}]^T \tag{3}$$

where $l_p, l_s, l_f$ are dimensions, $\mathbf{r}$ is the robocentric position of the IMU expressed in $\mathcal{B}$, $\boldsymbol{\upsilon}$ represents the robocentric velocity of the IMU expressed in $\mathcal{B}$, $\mathbf{q}$ is the IMU attitude represented as a map from $\mathcal{B} \rightarrow \mathcal{W}$, $\mathbf{b}_f$ represents the additive accelerometer bias expressed in $\mathcal{B}$, $\mathbf{b}_\omega$ stands for the additive gyroscope bias expressed in $\mathcal{B}$, $\mathbf{c}$ is the translational part of the IMU–cameras extrinsics expressed in $\mathcal{B}$, $\mathbf{z}$ represents the rotational part of the IMU–cameras extrinsics and is a map from $\mathcal{B} \rightarrow \mathcal{V}$, while $\boldsymbol{\mu}_j$ is the bearing vector to feature $j$ expressed in $\mathcal{V}$ and $\rho_j$ is the depth parameter of the $j$th feature such that the feature distance $d_j$ is $d(\rho_j) = 1/\rho_j$. The relevant state propagation and update steps are briefly summarized in Table 1, and are detailed in [13].

### 3.4.2 Belief Propagation

To enable localization uncertainty–aware planning at the second layer of the presented `rhemplanner`, belief propagation takes place such that the sampled paths contain the expected values and covariance estimates of both the robot state and the landmarks corresponding to the latest tracked features. This is achieved through the following: First, for every path segment (an edge $\sigma_{k-1,k}$ of $\mathbb{T}^M$), the expected IMU trajectories are derived by simulating the closed–loop robot dynamics $\dot{\boldsymbol{\chi}} = f(\boldsymbol{\chi}, \boldsymbol{\chi}^r)$ ($\boldsymbol{\chi}$ being the robot pose $[\mathbf{r}\ \mathbf{q}]^T$ expressed in $\mathcal{W}$ and $\boldsymbol{\chi}^r$ its reference) sampled every $T_s$. The closed–loop dynamics can be derived through first–principles methods or (as in this work) grey–box system identification. A second–order system was considered for the purposes of grey–box system identification. Using these IMU trajectories, prediction of the robot belief takes place by running the state propagation step in the EKF–fashion of ROVIO shown in Table 1, Eq. 3. In order to then conduct the filter update step, the landmarks corresponding to the features tracked at the initialization of the planner are projected in world coordinates $\mathcal{V} \rightarrow \mathcal{W}$ using the estimated feature distance, as well as the extrinsic and intrinsic camera parameters. For feature $\boldsymbol{\mu}_j$ this takes the form:

**Table 1** ROVIO state propagation and filter update steps

| State propagation step - Eq. 3 |
|---|
| $\dot{\mathbf{r}} = -\hat{\boldsymbol{\omega}}^{\times}\mathbf{r} + \boldsymbol{v} + \mathbf{w}_r$ |
| $\dot{\boldsymbol{v}} = -\hat{\boldsymbol{\omega}}^{\times}\boldsymbol{v} + \hat{\mathbf{f}} + \mathbf{q}^{-1}(\mathbf{g})$ |
| $\dot{\mathbf{q}} = -\mathbf{q}(\hat{\boldsymbol{\omega}})$ |
| $\dot{\mathbf{b}}_f = \mathbf{w}_{bf}$ |
| $\dot{\mathbf{b}}_\omega = \mathbf{w}_{bw}$ |
| $\dot{\mathbf{c}} = \mathbf{w}_c$ |
| $\dot{\mathbf{z}} = \mathbf{w}_z$ |
| $\dot{\boldsymbol{\mu}}_j = \mathbf{N}^T(\boldsymbol{\mu}_j)\hat{\boldsymbol{\omega}}_{\mathcal{V}} - \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}\mathbf{N}^T(\boldsymbol{\mu}_j)\frac{\hat{\boldsymbol{v}}_{\mathcal{V}}}{d(\rho_j)} + \mathbf{w}_{\mu,j}$ |
| $\dot{\rho}_j = -\boldsymbol{\mu}_j^T\hat{\boldsymbol{v}}_{\mathcal{V}}/d'(\rho_j) + w_{\rho,j}$ |
| $\hat{\mathbf{f}} = \tilde{\mathbf{f}} - \mathbf{b}_f - \mathbf{w}_f$ |
| $\hat{\boldsymbol{\omega}} = \tilde{\boldsymbol{\omega}} - \mathbf{b}_\omega - \mathbf{w}_\omega$ |
| $\hat{\boldsymbol{v}}_{\mathcal{V}} = \mathbf{z}(\boldsymbol{v} + \hat{\boldsymbol{\omega}}^{\times}\mathbf{c})$ |
| $\hat{\boldsymbol{\omega}}_{\mathcal{V}} = \mathbf{z}(\hat{\boldsymbol{\omega}})$ |
| Filter update step - Eq. 4 |
| $\mathbf{y}_j = \mathbf{b}_j(\boldsymbol{\pi}(\hat{\boldsymbol{\mu}}_j)) + \mathbf{n}_j$ |
| $\mathbf{H}_j = \mathbf{A}_j(\boldsymbol{\pi}(\hat{\boldsymbol{\mu}}_j))\frac{d\boldsymbol{\pi}}{d\boldsymbol{\mu}}(\hat{\boldsymbol{\mu}}_j)$ |
| By stacking the above terms for all visible features, standard EKF update step is directly conducted to derive the new estimate of the robot belief for its state and tracked features. |
| Notation |
| $^{\times} \rightarrow$ skew symmetric matrix of a vector, $\tilde{\mathbf{f}} \rightarrow$ proper acceleration measurement, $\tilde{\boldsymbol{\omega}} \rightarrow$ rotational rate measurement, $\hat{\mathbf{f}} \rightarrow$ biased corrected acceleration, $\hat{\boldsymbol{\omega}} \rightarrow$ bias corrected rotational rate, $\mathbf{N}^T(\boldsymbol{\mu}) \rightarrow$ projection of a 3D vector onto the 2D tangent space around the bearing vector, $\mathbf{g} \rightarrow$ gravity vector, $\mathbf{w}_\star \rightarrow$ white Gaussian noise processes, $\boldsymbol{\pi}(\boldsymbol{\mu}) \rightarrow$ pixel coordinates of a feature, $\mathbf{b}_i(\boldsymbol{\pi}(\hat{\boldsymbol{\mu}}_j)) \rightarrow$ a 2D linear constraint for the $j$th feature which is predicted to be visible in the current frame with bearing vector $\hat{\boldsymbol{\mu}}_j$ |

$$\begin{bmatrix} X_j \\ Y_j \\ Z_j \end{bmatrix} = \mathbf{R}_C^W \left( \mathbf{K}_I^{-1} \frac{1}{\rho_j} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} \right) + \mathbf{T}_C^W \tag{4}$$

where $X_j$, $Y_j$, $Z_j$ are their calculated world coordinates, $\mathbf{R}_C^W$ and $\mathbf{T}_C^W$ represent extrinsic camera pose in world coordinates, $\mathbf{K}_I$ represents the intrinsic camera matrix, and $\boldsymbol{\pi}(\boldsymbol{\mu}_j) = [u, v]$ are the feature pixel coordinates. At each step of the belief propagation, only visible landmarks are considered. To evaluate the visibility of a certain landmark, an appropriate check given the robot pose, the map $\mathcal{M}$ and the camera frustrum model is performed (using ray–casting and checking collisions with the map). As bearing vectors for all the landmarks are propagated in the state propagation step, the algorithm subsequently utilizes those predicted to be visible in the

new planned configuration in order to update the covariance estimates through the filter update step. The calculated Jacobians are used to update the robot covariance estimate in standard EKF fashion, that is $\Sigma_{t,l} = (\mathbf{I} - \mathbf{K}_t\mathbf{H}_t)\bar{\Sigma}_{t,l}$ with $\mathbf{K}_t$ being the current gain, $\mathbf{H}_t$ the Jacobians, $\mathbf{I}$ the identity matrix, $\Sigma_{t,l}$ the updated covariance, and $\bar{\Sigma}_t$ denoting the covariance estimate of the state propagation step. This is analogous to ROVIO's covariance update operation following the filter update of Table 1, Eq. 4. By interacting this process for all simulated trajectories and the corresponding sampled locations along the tree $\mathbb{T}^M$ branches, the belief is propagated to obtain the estimates for the covariance matrix $\Sigma_l$ and its pose–features subset $\Sigma_{p,f}$. These estimates are then used in the belief–based gain calculation as shown below. Finally, from an implementation standpoint it is important to highlight that the belief propagation steps rely on a modification of the ROVIO framework. A specific ROS package called rovio_bsp has been developed and is organized as a submodule on the RHEM planner open–source repository. This is another example of the instrumental role of ROS in the implementation and deployment of the presented framework for autonomously exploring robots.

### 3.4.3 Belief Uncertainty–Based Gain Calculation

Among the key requirements for a robust visual–inertial odometry system are that it (a) must reobserve landmarks with good confidence, while (b) preferably following trajectories that excite the inertial sensors and are therefore informative to the estimation process. This serves to improve both the feature location and the robot pose estimates due to the statistical correlations linking the vehicle to the features. Especially when the robot explores an unknown environment, new features are initialized into the map. This imposes the need to reobserve previous features in order to reduce the growth in localization error. To encode this behavior into the planning process, each admissible branch of $\mathbb{T}^M$ is evaluated regarding the uncertainty of the robot belief about its pose and the tracked landmarks along its edges as explained in Sect. 3.4.2. Then, for the derived pose and tracked landmarks covariance matrix $\Sigma_{p,f}$ (subset of $\Sigma_l$), **BeliefGain** computes its *D-optimality* (*D-opt*) [12] metric which measures the area of the uncertainty ellipsoids. Therefore, for two robot path policies $\sigma_1^M$ and $\sigma_2^M$, D-opt is used to evaluate which of the two $(l_p + l_f) \times (l_p + l_f)$ covariance matrices $\Sigma_{p,f}(\sigma_1^M)$ and $\Sigma_{p,f}(\sigma_2^M)$ corresponds to a more confident belief state once the robot arrives at the *end–vertex* of each path. Following the unifying uncertainty criteria definitions of Kiefer [14], the formulation of [12] is employed:

$$D_{opt}(\sigma^M) = \exp(\log([\det(\Sigma_{p,f}(\sigma^M)]^{1/(l_p+l_f)})) \tag{5}$$

Accordingly, **BeliefGain** computes the D-opt at the end–vertex $\xi_{RH}$ for each admissible branch $\sigma_\alpha^M$(where $\alpha$ indexes the admissible branches extracted using the **ExtractBranch** function):

$$\textbf{BeliefGain}(\sigma_\alpha^M) = D_{opt}(\sigma_\alpha^M) \tag{6}$$

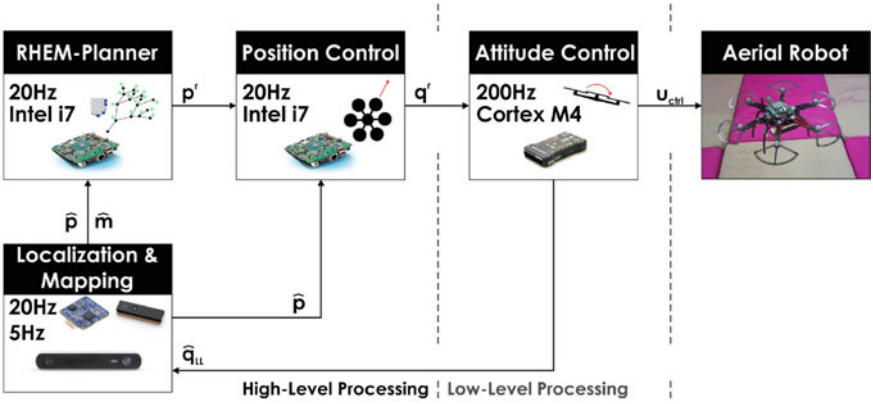


**Fig. 4** Diagram of the main functional loops of the robot. The planning, localization and mapping, as well as position control loops (high–level processing) are implemented and interconnected using ROS. Finally, the position controller provides reference attitude and thrust commands to a PX4 autopilot through the MAVROS package

This procedure has the effect of selecting the path which arrives to the viewpoint provided by the exploration planning level with minimal expected localization uncertainty.

### 3.4.4 Indicative Evaluation Study

A first experimental evaluation of the described planner took place using a custom–built hexarotor platform which has a weight of 2.6 kg. The system is relying on a Pixhawk–autopilot for its attitude control, while further integrating an Intel NUC5i7RYH, and a stereo visual–inertial sensor. The planning, localization and mapping, and position control loops are running on the NUC5i7RYH. The system performs visual–inertial odometry using ROVIO and dense mapping through the depth point cloud from the stereo camera and the robot pose estimates. For the position control task, the linear model predictive controller described in [15] is utilized. The complete set of high–level tasks run with the support of the ROS. In particular, in the utilized set-up, the localization pipeline based on ROVIO, the position model predictive control, as well as the exploration planner are implemented through ROS which facilitates their easier integration. Topics for the map and the pose are seamlessly shared among these core functional modules of the robot and enable its operational autonomy. Furthermore, the ROS-based implementation of ROVIO allowed the convenient modification of its propagation and correction pipeline in order to develop the uncertainty–aware capabilities of the RHEM planner. Our team has used similar systems before for relevant operations [16–19]. Figure 4 presents the main functional loops of the robot and highlights the importance of ROS in its realization.

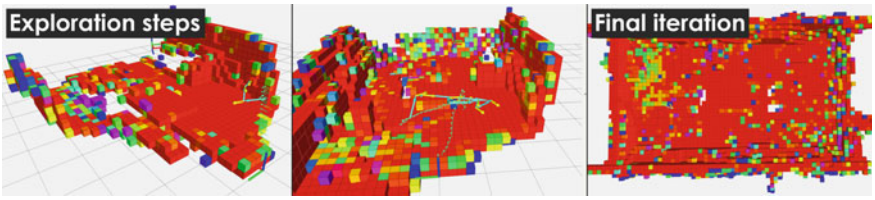


**Fig. 5** Exploration steps of the uncertainty–aware receding horizon exploration and mapping planner alongside with mapping

The experimental scenario refers to the mapping of an indoors environment with dimensions $12 \times 6.5 \times 2\,\text{m}$. Using 300 boxes with size $0.4 \times 0.3 \times 0.3\,\text{m}$, vertical and T–shaped walls, as well as other structural elements are created to complexify the robot exploration and mapping mission. During the experiment, the robot is constrained to fly with a maximum velocity of $v_{\max} = 0.75\,\text{m/s}$ and maximum yaw rate $\dot{\psi}_{\max} = \pi/4\,\text{rad/s}$. Figure 5 presents instances of the localization uncertainty–aware mission, while the colormap of the occupancy map relates to the probability of a voxel being occupied.

A second experimental study was conducted using the aerial robot now equipped with the visual–inertial localization module enhanced with flashing LEDs synchronized with the camera shutter, therefore enabling localization and mapping in dark, visually-degraded environments. Furthermore, a depth time-of-flight camera sensor is installed on the system to deliver reliable point cloud information in conditions of darkness. The experimental scenario refers to a subset of the same environment and the derived results are depicted in Fig. 6. In such a scenario, the ability of the proposed planner to account for the localization uncertainty is particularly important due to the increased challenges the perception system faces in such conditions.

The abovementioned results are indicative. An extensive set of studies using the `rhemplanner` are presented in [20, 21] and include indoor office–like environments in well–lit conditions, (b) dark rooms, as well as (c) dark and broadly visually–degraded (e.g. haze) city tunnel environments that particularly demonstrate the importance of accounting for the localization uncertainty of the robot. The video in https://youtu.be/iveNtQyUut4 presents an indicative experimental demonstration of the operation of the presented planner. Furthermore, the video in https://youtu.be/1-nPFBhyTBM presents the planner operation for a robot exploring its environment in conditions of darkness using a Near IR stereo camera.

## 4 Terrain Monitoring

This section overviews the Terrain Monitoring Planner (`tmplanner`): a generic finite-horizon informative planning method for monitoring flat terrain using aerial robots [22, 23]. This strategy aims for the timely and cost-effective delivery of high-

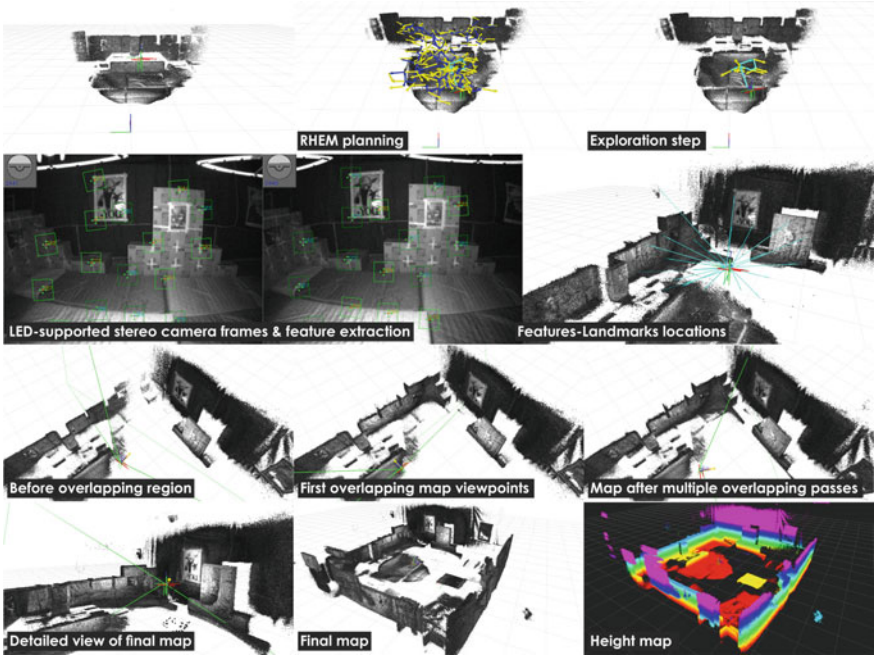


**Fig. 6** Experimental demonstration of the proposed planning in darkness supported by a specialized stereo visual–inertial sensor synchronized with flashing LEDs. First row: RHEM exploration steps enabled the volumetric mapping derived from the combination of visual–inertial odometry estimates and the dense point cloud from the ToF depth sensor. Second row: robust stereo localization based on the LED–assisted visual–inertial odometry, while operating in darkness. Third row: Mapping consistency demonstrated after multiple passes from overlapping viewpoints without loop closure - as shown 3D reconstruction consistency is achieved. Fourth row: Final map result and detailed views

quality data as an alternative to using portable sensors in arduous and potentially dangerous campaigns. The key challenge we address is trading off image resolution and FoV to find most useful measurement sites, while accounting for limited endurance and computational resources. The presented framework is capable of probabilistically mapping either discrete or continuous target variables using an altitude-dependent sensor. During a mission, the terrain maps maintained are used to plan information-rich trajectories in continuous 3D space through a combination of grid search and evolutionary optimization.

As a motivating application throughout this section, we consider the use-case of agricultural monitoring employing aerial robots, where the objective may be to detect the presence of an infectious growth (binary) or to monitor the distributions of green biomass level (continuous) on a field. This workflow enables quickly finding precision treatment targets, procuring detailed data for management decisions to reduce chemical usage and optimize yield.

Sections 4.1 and 4.2 present methods of map representation for mapping discrete and continuous targets, respectively, as the basis of our framework. The informative planning algorithm is then outlined in Sect. 4.3.

## 4.1 Mapping Discrete Variables

The task of monitoring a discrete variable is considered as an active classification problem. For simplicity in the following description, we consider binary variable monitoring. The mapped environment $\mathcal{E}$ (Sect. 2.2) is discretized at a certain resolution and represented using a 2D occupancy map $\mathcal{M}$ [24], where each cell is associated with a Bernoulli random variable indicating the probability of target occupancy (e.g., weed occupancy in agricultural monitoring). Measurements are taken with a downwards-looking camera supplying inputs to a classifier unit. The classifier provides occupancy updates for cells within FoV from an aerial configuration $\mathbf{x}$ above the terrain. At time $t$, for each observed cell $\mathbf{m}_i \in \mathcal{M}$, a log-likelihood update is performed given an observation $z$:

$$L(\mathbf{m}_i|z_{1:t}, \mathbf{x}_{1:t}) = L(\mathbf{m}_i|z_{1:t-1}, \mathbf{x}_{1:t-1}) + L(\mathbf{m}_i|z_t, \mathbf{x}_t), \tag{7}$$

where $L(\mathbf{m}_i|z_t, \mathbf{x}_t)$ denotes the altitude-dependent inverse sensor model capturing the classifier output.

Figure 7 shows an example sensor model for a hypothetical binary classifier labeling observed cells as "1" (occupied by target) or "0" (target-free). For each of the two classes, curves are defined to account for poorer classification with lower-resolution images at higher altitudes. In practice, these curves can be obtained through a Monte Carlo–type analysis of raw classifier data by averaging the number of true- and false-positives (blue and orange curves, respectively) recorded at different altitudes.

The described approach can be easily extended to mapping multiple class labels by maintaining layers of occupancy maps.

## 4.2 Mapping Continuous Variables

To monitor a continuous variable, our framework leverages a more sophisticated mapping method using Gaussian processes (GPs) as a natural way of encoding spatial correlations common in environmental distributions [25]. In brief, a GP is used to initialize a recursive procedure of Bayesian data fusion with probabilistic sensors at potentially different resolutions. A key benefit of this approach is that it replaces the computational burden of applying GPs directly with constant processing time in the number of measurements. This sub-section describes our method of creating prior maps before detailing the multiresolution data fusion technique.

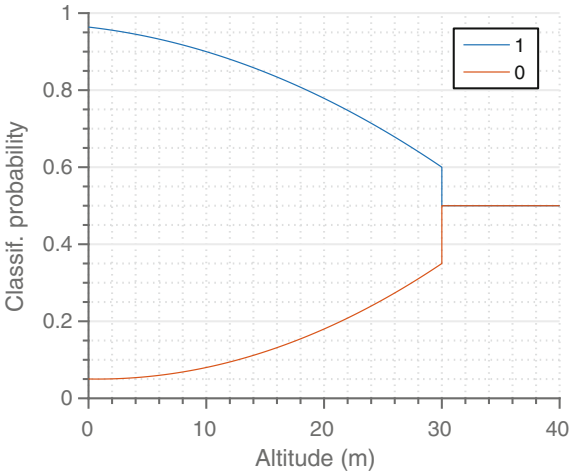


**Fig. 7** Probabilistic inverse sensor model for a typical snapshot camera-based binary classifier. The blue and orange curves depict the probability of label "1" given that label "1" or "0" was observed, respectively, i.e., $p(\mathbf{m}_i|z, \mathbf{x})$. As altitude increases, the curves approach unknown classification probability (0.5)

### 4.2.1 Gaussian Processes

A GP is used to model spatial correlations on the terrain in a probabilistic and non-parametric manner [25]. The target variable for mapping is assumed to be a continuous function in 2D space: $\zeta : \mathcal{E} \rightarrow \mathbb{R}$. Using the GP, a Gaussian correlated prior is placed over the function space, which is fully characterized by the mean $\mu = E[\zeta]$ and covariance $\mathbf{P} = E[(\zeta - \mu)(\zeta^\top - \mu^\top)]$ as $\zeta \sim \mathcal{GP}(\mu, \mathbf{P})$.

Given a pre-trained kernel $K(\mathcal{X}, \mathcal{X})$ for a fixed-size terrain discretized at a certain resolution with a set of $n$ locations $\mathcal{X} \subset \mathcal{E}$, we first specify a finite set of new prediction points $\mathcal{X}^* \subset \mathcal{E}$ at which the prior map is to be inferred. For unknown environments, the values at $\mathbf{m}_i \in \mathcal{X}$ are initialized uniformly with a constant prior mean. For known environments, the GP can be trained from available data and inferred at the same or different resolutions. The covariance is calculated using the classic GP regression equation [26]:

$$\mathbf{P} = K(\mathcal{X}^*, \mathcal{X}^*) - K(\mathcal{X}^*, \mathcal{X})[K(\mathcal{X}, \mathcal{X}) + \sigma_n^2 \mathbf{I}]^{-1} \times K(\mathcal{X}^*, \mathcal{X})^\top, \tag{8}$$

where $\mathbf{P}$ is the posterior covariance, $\sigma_n^2$ is a hyperparameter representing signal noise variance, and $K(\mathcal{X}^*, \mathcal{X})$ denotes cross-correlation terms between the predicted and initial locations.

The kernel function is chosen to describe the properties of the target distribution. For instance, a commonly used kernel in geostatistical analysis (e.g., for vegetation mapping) is the isotropic Matérn 3/2 function, defined as [25]:

$$k_{Mat3}(\mathbf{m}, \mathbf{m}^*) = \sigma_f^2 \left(1 + \frac{\sqrt{3}d}{l}\right) \exp\left(-\frac{\sqrt{3}d}{l}\right), \tag{9}$$

where $d$ is the Euclidean distance between inputs $\mathbf{m}$ and $\mathbf{m}^*$, and $l$ and $\sigma_f^2$ are hyperparameters representing the lengthscale and signal variance, respectively.

The resulting set of fixed hyperparameters $\theta = \{\sigma_n^2, \sigma_f^2, l\}$ controls relations within the GP. These values can be optimized using various methods [25] to match the properties of $\zeta$ by training on multiple maps previously obtained at the required resolution.

Once the correlated prior map $p(\zeta|\mathcal{X})$ is determined, independent noisy measurements at possibly different resolutions are fused as described in the following.

### 4.2.2 Sequential Data Fusion

Measurement updates are performed based on a recursive filtering procedure [27]. Given a uniform mean and the spatial correlations captured with Eq. 8, the map $p(\zeta|\mathcal{X}) \sim \mathcal{GP}(\mu^-, \mathbf{P}^-)$ is used as a prior for fusing new uncertain sensor measurements.

Let $\mathbf{z} = [z_1, \ldots, z_m]^\top$ denote a set of $m$ new independent measurements received at points $[\mathbf{m}_1, \ldots, \mathbf{m}_m]^\top \subset \mathcal{X}$ modelled assuming a Gaussian sensor as $p(z_i|\zeta_i, \mathbf{m}_i) = \mathcal{N}(\mu_{s,i}, \sigma_{s,i})$. To fuse the measurements $\mathbf{z}$ with the prior map $p(\zeta|\mathcal{X})$, the maximum a posteriori estimator is used:

$$\underset{\zeta}{\operatorname{argmax}}\ p(\zeta|\mathbf{z}, \mathcal{X}) \tag{10}$$

The Kalman Filter (KF) update equations are applied directly to compute the posterior density $p(\zeta|\mathbf{z}, \mathcal{X}) \propto p(\mathbf{z}|\zeta, \mathcal{X}) \times p(\zeta|\mathcal{X}) \sim \mathcal{GP}(\mu^+, \mathbf{P}^+)$ [26]:

$$\mu^+ = \mu^- + \mathbf{K}\mathbf{v} \tag{11}$$

$$\mathbf{P}^+ = \mathbf{P}^- - \mathbf{K}\mathbf{H}\mathbf{P}^-, \tag{12}$$

where $\mathbf{K} = \mathbf{P}^-\mathbf{H}^\top\mathbf{S}^{-1}$ is the Kalman gain, and $\mathbf{v} = \mathbf{z} - \mathbf{H}\mu^-$ and $\mathbf{S} = \mathbf{H}\mathbf{P}^-\mathbf{H}^\top + \mathbf{R}$ are the measurement and covariance innovations. $\mathbf{R}$ is a diagonal $m \times m$ matrix of altitude-dependent variances $\sigma_{s,i}^2$ associated with each measurement $z_i$, and $\mathbf{H}$ is a $m \times n$ matrix denoting a linear sensor model that intrinsically selects part of the state $\{\zeta_1, \ldots, \zeta_m\}$ observed through $\mathbf{z}$. The information to account for variable-resolution measurements is incorporated straightforwardly through the measurement model $\mathbf{H}$ as detailed in the following sub-section.

The constant-time updates in Eqs. 11 and 12 are repeated every time new data is registered. Note that, as all models are linear in this case, the KF update produces the optimal solution. This approach is also agnostic to the type of sensor as it permits fusing heterogeneous data into a single map.

### 4.2.3 Altitude-Dependent Sensor Model

This sub-section details an altitude-dependent sensor model for the downward-facing camera used to take measurements of the terrain. As opposed to the classification case in Sect. 4.1, our model needs to express uncertainty with respect to a continuous target distribution. To do this, the information collected is considered as degrading with altitude in two ways: (i) noise and (ii) resolution. The proposed model accounts for these issues in a probabilistic manner as described below.

We assume an altitude-dependent Gaussian sensor noise model. For each observed point $\mathbf{m}_i \in \mathcal{X}$, the camera provides a measurement $z_i$ capturing the target field $\zeta_i$ as $\mathcal{N}(\mu_{s,i}, \sigma_{s,i})$, where $\sigma^2_{s,i}$ is the noise variance expressing uncertainty in $z_i$. To account for lower-quality images taken with larger camera footprints, $\sigma^2_{s,i}$ is modelled as increasing with vehicle altitude $h$ using:

$$\sigma^2_{s,i} = a(1 - e^{-bh}), \tag{13}$$

where $a$ and $b$ are positive constants.

Figure 8 shows an example sensor noise model for a hypothetical snapshot camera. The measurements $z_i$ denote the levels of the continuous target variable of interest, derived from the images. In the agricultural monitoring case, for example, green biomass level can be computed from calibrated spectral indices in the images to map areas on a farmland with excessive growth [3]. As for the binary classifier, this model can be obtained from a raw analysis of sensor data.

To cater for poorer-quality sensors, we define altitude envelopes corresponding to different image resolution scales with respect to the initial points $\mathcal{X}$. This is motivated from the fact that an area per pixel ratio on the altitude of the sensor and its implemented resolution. At higher altitudes and lower resolutions, adjacent grid cells $\mathbf{m}_i$ are indexed by a single sensor measurement $z_i$ through the sensor model $\mathbf{H}$. At the maximum mapping resolution, $\mathbf{H}$ is simply used to select the part of the state observed with a scale of 1, i.e., one-to-one mapping. However, to handle lower-

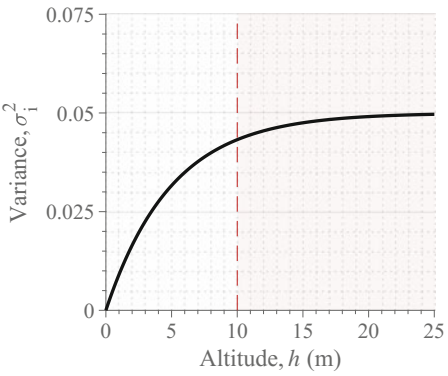


**Fig. 8** Sensor noise model for a snapshot camera providing measurements as $\mathcal{N}(\mu_{s,i}, \sigma_{s,i})$ with $a = 0.2$, $b = 0.05$ in Eq. 13. The uncertainty $\sigma^2_{s,i}$ increases with $h$ to represent degrading image quality. The dotted line at $h = 10$ m indicates the altitude above which (to the right) image resolution scales down by a factor of 2

resolution data, the elements of $\mathbf{H}$ are used to map multiple $\zeta_i$ to a single $z_i$ scaled by the square inverse of the resolution scaling factor $s_f$. Note that the fusion described in the previous section is always performed at the maximum mapping resolution, so that the proposed model $\mathbf{H}$ considers low-resolution measurements as a scaled average of the high-resolution map.

## *4.3 Planning Approach*

This sub-section details the proposed planning scheme for terrain monitoring, which generates fixed-horizon plans for maximizing an informative objective. To do this efficiently, an evolutionary technique is applied to optimize trajectories initialized by a 3D grid search in the configuration space. We begin with an approach to parametrizing trajectories before detailing the algorithm itself.

### 4.3.1 Trajectory Parametrization

A polynomial trajectory $\psi$ is represented with a sequence of $N$ control waypoints to visit $\mathcal{C} = [\mathbf{c}_1, \ldots, \mathbf{c}_N]$ connected using $N - 1$ $k$-order spline segments for minimum-snap dynamics [28]. The first waypoint $\mathbf{c}_1$ is clamped as the initial vehicle position. As discussed in Sect. 2.2, the function MEASURE$(\cdot)$ in Eq. 1 is defined by computing the spacing of measurement sites along $\psi$ given a constant sensor frequency.

### 4.3.2 Algorithm

A fixed-horizon approach is used to plan adaptively as data are collected. During a mission, we alternate between replanning and execution until the elapsed time $t$ exceeds the budget $B$.

The replanning approach consists of two stages. First, an initial trajectory, defined by a fixed $N$ control points $\mathcal{C}$ is derived through a coarse grid search (Lines 3-6) in the 3D configuration space. This step proceeds in a greedy manner, allowing a rough solution to quickly be obtained. Then, the trajectory is refined to maximize the informative objective using an evolutionary optimization routine (Line 8).

This procedure is described generally in Algorithm 3, where $\mathcal{M}$ symbolizes a discretized environmental model capturing a discrete or continuous target variable. The following sub-sections discuss possible objectives for informative planning, and outline the key steps of the algorithm.

**Algorithm 3** REPLAN_PATH procedure

**Input:** Current model of the environment $\mathcal{M}$, number of control waypoints $N$, lattice points $\mathcal{L}$
**Output:** Waypoints defining the next polynomial plan $\mathcal{C}$

1: $\mathcal{M}' \leftarrow \mathcal{M}$ ▷ Create local copy of the map.
2: $\mathcal{C} \leftarrow \emptyset$ ▷ Initialize control points.
3: **while** $N \geq |\mathcal{C}|$ **do**
4: $\mathbf{c}^* \leftarrow$ Select viewpoint in $\mathcal{L}$ using Eq. 1
5: $\mathcal{X}' \leftarrow$ UPDATE_MAP($\mathcal{M}'$, $\mathbf{c}^*$) ▷ Predict a measurement from this point.
6: $\mathcal{C} \leftarrow \mathcal{C} \cup \mathbf{c}^*$
7: **end while**
8: $\mathcal{C} \leftarrow$ CMAES($\mathcal{C}$, $\mathcal{M}$) ▷ Optimize polynomial trajectory.

### 4.3.3 Utility Defintion

The utility (information gain) function $I$ in Eq. 1 is generic and can be set to express application-specific interests with respect to the map representation. For mapping a binary variable, we consider two definitions of $I$ for evaluating a measurement from a configuration $\mathbf{x}$ (Line 4). To encourage exploration, the reduction of Shannon's entropy H in the occupancy map $\mathcal{M}$ is maximized:

$$I[\mathbf{x}] = \mathrm{H}(\mathcal{M}^-) - \mathrm{H}(\mathcal{M}^+), \tag{14}$$

where the superscripts denote the prior and posterior maps given a measurement from $\mathbf{x}$.

To encourage classification, $\mathcal{M}$ is divided into "occupied" and "free" cells using thresholds $\delta_o$ and $\delta_f$, leaving an unclassified subset $U = \{\mathbf{m}_i \in \mathcal{M} \mid \delta_f < p(\mathbf{m}_i) < \delta_o\}$. The reduction of $\mathcal{U}$ is then maximized:

$$I[\mathbf{x}] = |\mathcal{U}^-| - |\mathcal{U}^+|. \tag{15}$$

For mapping a continuous variable using a GP-based map, we consider maximizing uncertainty reduction, as measured by the covariance $\mathbf{P}$:

$$I[\mathbf{m}] = \mathrm{Tr}(\mathbf{P}^-) - \mathrm{Tr}(\mathbf{P}^+), \tag{16}$$

where $\mathrm{Tr}(\cdot)$ is the trace of a matrix.

In an adaptive-planning setting, a value-dependent objective can be defined to focus on higher- or lower-values regions of interest. To this end, Eq. 16 is modified so that the elements of $\mathrm{Tr}(\mathbf{P})$ mapping to the mean of each cell $\mu_i$ via location $\mathbf{m}_i \in \mathcal{X}$ are excluded from the computation, given that they exceed specified thresholds [23].

Note that, while Eqs. 14–16 define $I$ for a single measurement site $\mathbf{x}$, the same principles can be applied to determine the utility of a trajectory $\psi$ by fusing a sequence of measurements and computing the overall information gain.

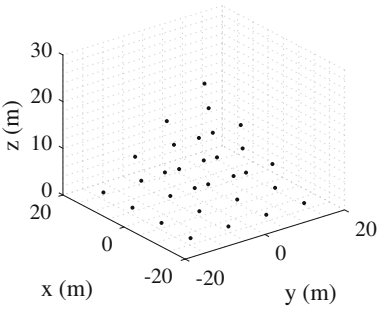


**Fig. 9** A visualization of an 3D lattice grid $\mathcal{L}$ example (30 points) for obtaining an initial trajectory solution. The length scales are defined for efficiency given the environment size and computational resources available

### 4.3.4 3D Grid Search

The first replanning step (Lines 3–6) supplies an initial solution for optimization in Sect. 4.3.5. To achieve this, the planner performs a 3D grid search based on a coarse multiresolution lattice $\mathcal{L}$ in the vehicle configuration space (Fig. 9). A low-accuracy solution neglecting sensor dynamics is obtained efficiently by using the points in $\mathcal{L}$ to represent candidate measurement sites and assuming constant velocity travel. Unlike in frontier-based exploration commonly used in cluttered environments [29], selecting measurement sites using map boundaries is not applicable in a terrain monitoring set-up. Instead, we conduct a sequential greedy search for $N$ waypoints (Line 3), where the next-best point $\mathbf{c}^*$ (Line 4) is found by evaluating Eq. 1 with the chosen utility $I$ in over $\mathcal{L}$. For each $\mathbf{c}^*$, a fused measurement is simulated (Line 5) via Eqs. 7 or 12 for a discrete or continuous mapping scenario, respectively. This point is then added to the initial trajectory solution (Line 6).

### 4.3.5 Optimization

The second replanning step (Line 7) optimizes the grid search solution by computing $I$ (Sect. 4.3.3) for a sequence of measurements taken along the trajectory. For global optimization, we propose the Covariance Matrix Adaptation Evolution Strategy (CMA-ES) [30]. This choice is motivated by the nonlinearity of the objective space (Eq. 1) as well as previous results in similar applications [22, 23, 31].

### 4.3.6 Indicative Evaluation Studies

Extensive simulation studies were conducted for mapping both discrete and continuous scenarios in realistic environments. The results obtained show that the proposed approach performs better than state-of-the-art strategies ("lawnmower" coverage planning, sampling-based planning) in terms of informative metrics. Applications of various utility functions and optimization methods in our framework were also compared. Finally, proof of concept experiments were demonstrated using rotorcraft

aerial robots to map different artificial agricultural farmland set-ups, including (a) AprilTag distributions and (b) color variations, corresponding to discrete and continuous targets, respectively. These experiments were performed in real-time with a preliminary version of `tmplanner`, the released open–source ROS release of our terrain monitoring framework. The employed robots rely on the ROS-based localization and position control methods utilized also in the framework of the previously mentioned experiments. In addition, we are working on adapting our framework and sensing interfaces for deployment in field trials on the farm.

Further details concerning the experimental results above can be found in our relevant publications [22, 23]. The video in https://youtu.be/1cKg1fjT54c is indicative of the planner experimental operation.

## 5 Optimized Inspection Path Planning

Unlike the exploration planners, the method described in this section aims to derive an optimized full coverage inspection path when a model of the structure of interest is known a priori. A triangular mesh–based representation of the structure is considered. Within the literature, a set of algorithms have been proposed to address the problem of 3D structural inspection.

In particular, approaches employing a two–step optimization scheme are renowned for their increased performance and versatility against increasing structural complexity. Such methods initially compute the minimal set of viewpoints that cover the whole structure (first step), which corresponds to solving an Art Gallery Problem (AGP). Subsequently (second step), the shortest connecting tour over all these viewpoints has to be computed, which corresponds to the Traveling Salesman Problem (TSP). Fast algorithms to approximately solve both the AGP and the TSP are known despite their inherent NP–hard properties. Indicative examples include the works in [32, 33] for the AGP, and [34, 35] for the TSP. Following a different approach, the works in [17, 36] concentrate on a truly optimal solution at the cost of expensive computational steps to a level that challenges their scalability. The fast inspection path planner described in this work retains a two–step optimization structure but, instead of attempting to find a minimal set of guards in the AGP, tries to sample them such that the connecting path is short, while ensuring full coverage. This alternative strategy is motivated from the fact that, given a continuously sampling sensor such as a camera or a LiDAR on a robot, it is not the number of viewpoints that makes the inspection path more efficient but rather their configuration and how this influences the travel cost of the overall inspection trajectory.

Given this fact, the presented Structural Inspection Planner (`siplanner`) selects one admissible viewpoint for every facet of the mesh of the structure to be inspected. In order to compute viewpoints allowing low–cost connections, an iterative resampling scheme is employed. Between each resampling, the best path connecting the current viewpoints is recomputed. The quality of the viewpoints is assessed by the cost to connect to their respective neighbours on the latest tour. This cost is minimized

in the subsequent resampling, resulting in locally optimized paths. Initialization of the viewpoints is arbitrarily done such that full coverage is provided with, at this stage, non–optimized viewpoints. A fast implementation of the Lin–Kernighan–Helsgaun Heuristic (LKH) TSP solver [37] is employed to compute the best tour, while the cost of the interconnecting pieces of path is calculated by means of a Boundary Value Solver (BVS). An overview of the proposed inspection path planning procedure is presented in Algorithm 4.

The following sub-sections outline the path generation approach and the optimization problem formulation to sample the viewpoints for a rotorcraft aerial robot. Then, we detail how the method deals with cluttered environments.

**Algorithm 4** `siplanner`

1: $k \leftarrow 0$
2: Sample initial viewpoint configurations
3: Compute cost matrix for the TSP solver (Sect. 5.1)
4: Solve the TSP problem to obtain initial tour
5: **while** running **do**
6: Resample viewpoint configurations (Sect. 5.2)
7: Recompute the cost matrix (Sect. 5.1)
8: Recompute best tour $T_{best}$ using the LKH and update best tour cost $c_{best}$ if applicable
9: $k \leftarrow k + 1$
10: **end while**
11: **return** $T_{best}$, $c_{best}$

## 5.1 Path Computation and Cost Estimation

In order to find the best tour among the viewpoints, the TSP solver requires a cost matrix containing the connection cost of all pairs of viewpoints. A two–state BVS is employed to generate paths between different viewpoints and estimate the respective costs. The BVS is either employed directly to connect the two viewpoints or as a component in a local planner. The latter applies in case the direct connection is not collision–free. In that case, our implementation uses the RRT*-planner [38] to find a collision–free connection. For rotorcraft aerial robots, the flat state consisting of position as well as yaw, $\xi = \{x, y, z, \psi\}$ is considered. Roll and pitch angles are assumed to be near zero as slow maneuvering is desired to achieve increased inspection accuracy. The path from configuration $\xi_0$ to $\xi_1$ is given by $\xi(s) = s\xi_1 + (1 - s)\xi_0$, where $s \in [0, 1]$. The single limitation considered is the speed limit, both in terms of maximum translational velocity $v_{max}$, and maximum yaw rate $\dot{\psi}_{max}$. To allow sufficiently accurate tracking of paths with corners, both values are small and should be adjusted for the scenario and the system being used. The resulting execution time is $t_{ex} = \max(d/v_{max}, \|\psi_1 - \psi_0\| / \dot{\psi}_{max})$, with $d$ the Euclidean distance. The considered cost metric of a path segment corresponds to the execution time $t_{ex}$. The

fact that time of travel, and not the Euclidean distance, is used as the cost function is particularly important given that the planner also supports fixed–wing Unmanned Aerial Vehicles. The open–source repository details how the method can be used for fixed–wing systems and account for the bank angle and velocity limits of the vehicle.

## *5.2 Viewpoint Sampling*

Considering a mesh–based representation of the environment, a viewpoint has to be sampled for each of its facets (triangular meshes are assumed). According to the method of this planner, the position and heading of each viewpoint is determined sequentially, while retaining visibility of the corresponding triangle. First, the position is optimized for distance w.r.t. the neighbouring viewpoints using a convex problem formulation and, only then, the heading is optimized. To guarantee a good result of this multi–step optimization process, the position solution must be constrained such as to allow finding an orientation for which the triangle is visible.

More specifically, the constraints on the position states $g = [x, y, z]$ consist of the inspection sensor limitations of minimum incidence angle, and minimum and maximum effective range $(d_{min}, d_{max})$ (depicted in Fig. 10). They are formulated as a set of planar constraints:

$$\begin{bmatrix} (g - x_i)^T n_i \\ (g - x_1)^T a_N \\ -(g - x_1)^T a_N \end{bmatrix} \succeq \begin{bmatrix} 0 \\ d_{min} \\ -d_{max} \end{bmatrix}, i = \{1, 2, 3\} \tag{17}$$

where $x_i$ are the corners of the mesh triangle, $a_N$ is the normalized triangle normal and $n_i$ are the normals of the separating hyperplanes for the incidence angle constraints as shown in Fig. 10.

Furthermore, as in the other planners presented in this chapter, the sensor is considered to have a limited FoV with certain vertical and horizontal opening angles and is mounted to the vehicle with a fixed pitch angle and relative heading. The imposed constraint on the sampling space resulting from the vertical camera opening is not convex (a revoluted 2D-cone, the height of which is depending on the relevant corners of the triangle over the revolution). Nonconvexity at a step that is iteratively computed multiple times can lead to significant computational cost for the planner. To approximate and convexify the problem, the space is divided in $N_C$ equal convex pieces according to Fig. 10. The optimum is computed for every slice in order to find the globally best solution. Multiple sensors with different vertical FoVs are handled equally, resulting in a multitude of $N_C$ convex pieces that possibly overlap. The constraints for piece $j$ are derived as follows: Left and right boundaries of the sampling space are the borders of the revolution segment and the cone top and bottom are represented by a single plane tangential to the centre of the slice. Angular camera constraints in horizontal direction are not encoded and instead $d_{min}$ is chosen high enough to allow full visibility of the triangle. This leaves some space

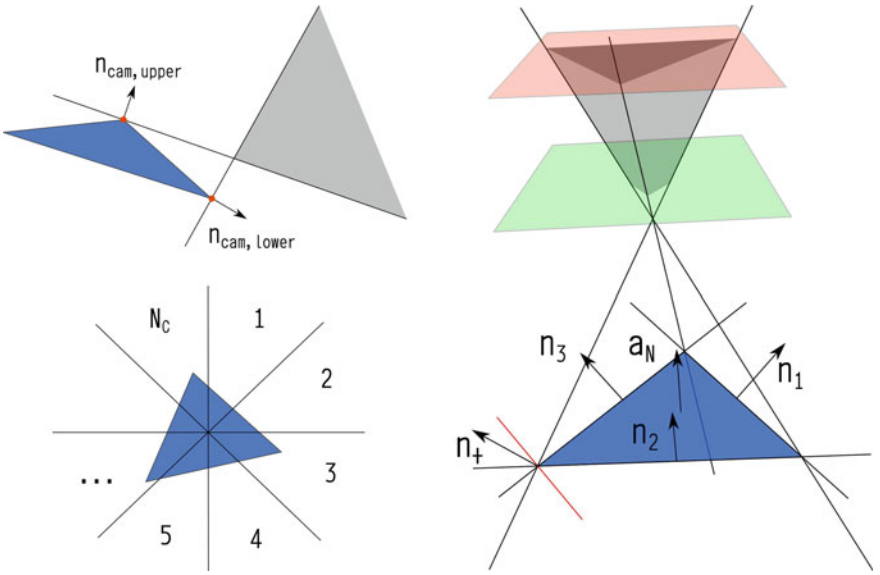


**Fig. 10** Left: The vertical camera angle constraints with the relevant corners of the triangle in red are depicted in the upper part, while the partition of the space for convexification is depicted beneath. Right: The figure depicts the three main planar angle of incidence constraints on all sides of the triangle. For a finite number of such constraints, the incidence angle is only enforced approximately. The red line (and $n_+$) remarks a sample orientation for a possible additional planar constraint at a corner. Minimum (green plane) and maximum (red plane) distance constraints are similar planar constraints on the sampling area. These constraints bound the sampling space, where $g$ can be chosen, on all sides (gray area)

for variation in the sampling of the heading, where the horizontal constraints are enforced. Specifically, the abovementioned constraints take the following form:

$$\begin{bmatrix} (g - x_{lower}^{rel})^T n_{lower}^{cam} \\ (g - x_{upper}^{rel})^T n_{upper}^{cam} \\ (g - m)^T n_{right} \\ (g - m)^T n_{left} \end{bmatrix} \succeq \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}, \tag{18}$$

where $x_{lower}^{rel}, x_{upper}^{rel}$ are the respective relevant corners of the mesh triangle, $m$ the middle of the triangle and $n_{lower}^{cam}, n_{upper}^{cam}, n_{right}$ and $n_{left}$ denote the normal of the respective separating hyperplanes.

The optimization objective for the viewpoint sampling in iteration $k$, in the case of a rotorcraft aerial robot, is to minimize the sum of squared distances to the preceding viewpoint $g_p^{k-1}$, the subsequent viewpoint $g_s^{k-1}$ and the current viewpoint $g^{k-1}$ in the old tour. The former two parts potentially shorten the tour by moving the viewpoints closer together, while the latter limits the size of the improvement step, as $g_p^{k-1}$ and $g_s^{k-1}$ potentially move closer as well. The weighting matrix $B$ for the neighbour

distance is given by $\mathbf{diag}(b_{const}, b_{const}, a_{const} + b_{const})$, where $b_{const}$ is the general weight for distance to neighbours, while $a_{const}$ additionally punishes changes in height. The distance to the current viewpoint in the old tour is weighted by the matrix $D = \mathbf{diag}(d_{const}, d_{const}, d_{const})$.

The resulting convex optimization problem is given below. Its structure as a Quadratic Program (QP) with linear constraints allows the use of an efficient solver [39].

$$\min_{g^k} \; (g^k - g_p^{k-1})^T B(g^k - g_p^{k-1}) + (g^k - g_s^{k-1})^T B(g^k - g_s^{k-1}) + \tag{19}$$
$$(g^k - g^{k-1})^T D(g^k - g^{k-1})$$

$$\text{s.t.} \quad \begin{bmatrix} n_1^T \\ n_2^T \\ n_3^T \\ a_N^T \\ -a_N^T \\ n_{lower}^{cam\ T} \\ n_{upper}^{cam\ T} \\ n_{right}^T \\ n_{left}^T \end{bmatrix} g^k \succeq \begin{bmatrix} n_1^T x_1 \\ n_2^T x_2 \\ n_3^T x_3 \\ a_N^T x_1 + d_{min} \\ -a_N^T x_1 - d_{max} \\ n_{lower}^{cam\ T} x_{lower}^{rel} \\ n_{upper}^{cam\ T} x_{upper}^{rel} \\ n_{right}^T m \\ n_{left}^T m \end{bmatrix} \tag{20}$$

For the computed optimal position, the heading is determined according to the criterion $\min_{\psi^k} = \left(\psi_p^{k-1} - \psi^k\right)^2 / d_p + \left(\psi_s^{k-1} - \psi^k\right)^2 / d_s$, subject to $\mathbf{Visible}(g^k, \psi^k)$, where $\mathbf{Visible}(g^k, \psi^k)$ means that from the given configuration, $g^k$ and $\psi^k$, the whole triangle is visible for at least one of the employed sensors. $d_p$ and $d_s$ are the Euclidean distances from $g^k$ to $g_p^{k-1}$ and $g_s^{k-1}$ respectively. For simple sensor setups establishing the boundaries on $\psi^k$ for $\mathbf{Visible}(g^k, \psi^k) = \text{TRUE}$ makes the solution explicit. Otherwise a grid search can be employed. It is highlighted that one of the important abilities of the `siplanner` from an applications standpoint is its ability to incorporate and account for multiple sensor models (e.g. multiple cameras).

### 5.3 Obstacle Avoidance

The `siplanner` performs efficiently in cluttered environments. Viewpoints are checked for occlusion by parts of the mesh that is to be inspected or meshes that represent obstacles of the environment for which sensor coverage is not desired. If interference is detected, the QP is reformulated to include an additional planar constraint defined by a corner $x^I$ of the interfering facet triangle, the centre of which is closest to the facet, as well as its normal $a_N^I$:

$$a_N^{I\,T} g^k \geq a_N^{I\,T} x^I \tag{21}$$

For meshes that do not selfintersect, this is a conservative exclusion of cluttered subareas. Furthermore, paths are checked for interference with the mesh to ensure collision–free connections between the viewpoints. This is easily achieved by integrating the collision check in the BVS. In order to shape a scenario according to special requirements, cuboidal obstacle regions can be defined being either transparent or non transparent. Both are respected in the planning of the connecting paths, while the latter are also included in the visibility check. When, such an obstacle is encountered during viewpoint sampling, the optimization problem is executed individually over eight subregions. Each is bounded away from the cuboid on one of its sides, similar to Eq. 21.

### *5.4 Indicative Evaluation Study*

A simulation evaluation study aiming to identify the optimized inspection trajectory for the case of a solar park covering an area of more than 4100 $m^2$ is presented in Figs. 11and 12. The total cost of the mission corresponds to 1259.36 s given a robot with FoV angles $[a_v, a_h] = [90, 120]^\circ$ and a mounting pitch of 20° downwards. The presented study is indicative, while more a set of experimental tests are presented in [16, 18, 40–42]. These include (a) the inspection of structures of the ETH Zurich Polyterrasse, (b) the inspection of a Wattmeter, (c) the mapping of a National Park in Switzerland using the fixed–wing AtlantikSolar solar–powered UAV, and more. It is noteworhy that beyond the planner, also the localization, and position control functionalities of the employed multirotor are also based on ROS and specifically on ROVIO [13] and Linear MPC [15] respectively. The videos in https://youtu.be/5kI5ppGTcIQ and https://youtu.be/qbwoKFJUm4k are indicative of the operation of the inspection planner.

## 6 Open–Source Code and Implementation Details

Towards enabling the research and development community, as well as the end–users to utilize and further develop the described autonomous exploration and inspection path planning algorithms, a set of open–source contributions have taken place. The "Receding Horizon Next-Best-View Planner" (`nbvplanner`), the "Localization Uncertainty–aware Receding Horizon Exploration and Mapping Planner" (`rhemplanner`), the "Terrain Monitoring Planner" (`tmplanner`) as well as the "Structural Inspection Planner" (`siplanner`) are already released as open–sourced Robot Operating System packages. Together with this chapter, we also organize and release a comprehensive single repository that links all the planners, further documents them and relates them in terms of their functionality and the relevant problems they address. This can be found at https://github.com/unr-arl/informative-planning.

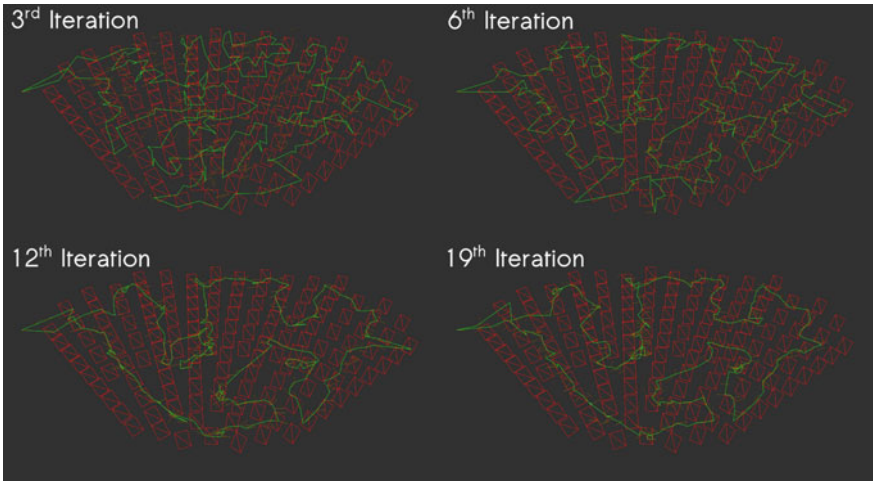


**Fig. 11** Iterations of the `siplanner` in order to identify the most optimized trajectory for the inspection of a solar power park covering an area of more than 4100 m$^2$

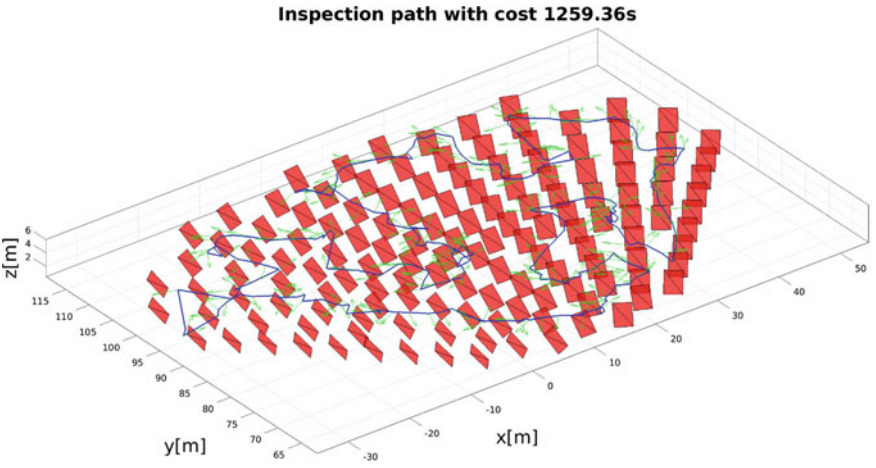


**Fig. 12** Final optimized inspection path for covering a solar power park covering an area of more than 4100 m$^2$. The total travel time is 1259.36 s given a robot with a sensor characterized by FoV angles $[a_v, a_h] = [90, 120]^\circ$ and a mounting pitch of 20° downwards and subject to dynamic constraints of maximum forward velocity of 0.5 m/s and maximum yaw rate of 0.5 rad/s

Additionally, a set of associated datasets from real flight tests and demo scenarios are released.

## 6.1 ROS Environment Configuration

In terms of ROS support, the released packages are tested to work with three of the latest ROS versions, namely Indigo, Jade, and Kinetic, while continuous maintenance takes place. As thoroughly explained in the Wiki and Readme sections of the respective open–source repositories the catkin build for the `nbvplanner`, `rhemplanner`, and `tmplanner`. Several package dependencies are required for the compilation of each package which are documented in detail within the respective Wkis for `nbvplanner`, `rhemplanner`, and `siplanner`, while a rosinstall file for the `tmplanner`.

## 6.2 Exploration Planners

For `nbvplanner` and `rhemplanner`, the command to the robot is provided iteratively in the form of a new reference waypoint to be tracked by the onboard position controller. For the implementation of both planners, the environment is represented using the octomap framework [6] and planning takes place within the free space mapped at every planning iteration. Both planners initialize each iteration from the currently estimated robot pose, while the `rhemplanner` further considers the latest set of tracked landmarks by ROVIO (or an alternative pipeline that tracks landmarks to estimate the robot pose) and the associated robot pose and landmarks covariance matrix.

The planners were developed having aerial robots in mind but can be extended to any robotic configuration that affords a BVS such that sampling can take place in the configuration space. The flat state $\xi = [x, y, z, \psi]^T$ considered in the work is an implementation detail that can be modified. Slow maneuvering aerial robots of the rotorcraft class are considered and the connection cost of a path $\sigma_{k-1}^{k}$ is considered to be the Euclidean distance $c(\sigma_{k-1}^{k}) = \sqrt{(x_k - x_{k-1})^2 + (y_k - y_{k-1})^2 + (z_k - z_{k-1})^2}$. The maximum speed and the yaw rate limit are enforced when sampling the path to generate a reference trajectory.

Both planners consider fixed–mounted cameras, limited by a certain horizontal and vertical FoV $[a_v, a_h]$, as well as a maximum distance of effective perception. For a sensor with a practical limitation of maximum sensing distance $d_{\max}^{\text{sensor}}$, the algorithm uses a value $d_{\max}^{\text{planner}} \leq d_{\max}^{\text{sensor}}$ to determine the volumetric gain of a configuration. Using a lower $d_{\max}^{\text{planner}}$ enhances the robustness against suboptimal sensing conditions.

### 6.2.1 `nbvplanner` Open–Source Release

The `nbvplanner` is implemented and released as an open–source ROS package at https://github.com/ethz-asl/nbvplanner and further documented through https://github.com/unr-arl/informative-planning. The planner is verified to work on ROS

Indigo, Jade and Kinetic. Experimental datasets relevant with the planner can also be acquired through its repository under "Wiki-Example Results". The simplest way to test and evaluate the planner is through its integration with the RotorS Simulator [43]. Provided that the installation instructions have been thoroughly followed, execution of the simplest scenario refers to the execution of the following command

```
$ roslaunch interface_nbvp_rotors flat_exploration.launch
```

During its operation, the planner is called via a rosservice. To embed the open-sourced `nbvplanner` in your own C++ project, add the following snippet:

```
#include <ros/ros.h>
#include <nbvplanner/nbvp_srv.h>

...

int iteration = 0;
while (ros::ok()) {
  nbvplanner::nbvp_srv planSrv;
  planSrv.request.header.stamp = ros::Time::now();
  planSrv.request.header.seq = iteration;
  planSrv.request.header.frame_id = ros::this_node::
    getNamespace();
  if (ros::service::call("nbvplanner", planSrv)) {

      ...

      // Process the computed path in planSrv.response.
    path to make
      // the robot track it

      ...

      iteration++;
    }
  } else {
    ROS_WARN_THROTTLE(1, "Planner not reachable");
    ros::Duration(1.0).sleep();
  }
}
```

A set of parameters can be defined to account for limitations of the particular scenario and adjust the behavior of the planner. Those include system–related parameters and model constraints (e.g. maximum velocity and heading rate), octomap parameters (e.g. voxel edge length size, probability of a hit), exploration gain computation parameters (e.g. gain for visible free volumes, maximum distance of volumes to be considered for the gain computation), path planning parameters (e.g. maximum extension range for new tree branches when sampling for a holonomic system, time step for path sampling), logging parameters (e.g. time interval to log values), and scenario parameters (e.g. minimum coordinates values of the scenario bounding only the gain computation and not the path planning tree expansion). Further parameters

are documented in the code repository and its associated wiki, alongside installation instructions and demo scenarios.

In addition, it is highlighted that the released `nbvplanner` implementation provides a demo example for a multi–robot system exploring a common space. This can be triggered by running the following command:

```
$ roslaunch interface_nbvp_rotors multiagent_area_exploration.
launch
```

### 6.2.2 `rhemplanner` Open–Source Release

The `rhemplanner` is implemented and released as an open–source ROS package at https://github.com/unr-arl/rhem_planner and further documented in https://github.com/unr-arl/informative-planning. Accompanying datasets are provided at the planner's repository under "Wiki-How to use Demo". The planner is verified to work on ROS Indigo, Jade and Kinetic. The planning pipeline is implemented in the `bsp_planner` which incorporates the two layered–planning procedure. As mentioned, the `rhemplanner` conducts propagation of the robot's localization belief which relies on the internally incorporated estimation and propagation pipeline. This is based on the ROVIO visual–inertial framework [13] and a respectively modified version is incorporated in the code release. It calculates and maintains the real–time localization and mapping state and associated statistics (estimation), and forward–simulates the process given an initial state and its associated statistics and a sample planned trajectory (propagation). These two key pipelines (planning, estimation and propagation) are then accordingly fused to realize the iteratively planning process of the `rhemplanner`. As in the case for the `nbvplanner`, the `rhemplanner` is called via a rosservice. The easiest way to test the planner is through the released dataset, as getting feedback from onboard sensors and the localization pipeline is critical for the operation of the planner. To do so, one has to execute:

```
$ cd rhem_planner
$ wget –P rhem_demo https://www.cse.unr.edu/%7Ekalexis/datasets/
icra2017–datasets/icra17_visensor_data.bag
```

Then the relevant launch file has to be fired:

```
$ roslaunch rhem_demo/rhem_demo_icra17.launch
```

As this refers to a recorded demo, executions of the planner should be triggered through the rosservice:

```
$ rosservice call /bsp_planner '{header: {stamp: now, frame_id:
    world}}'
```

Extended details for the subscribed topics and the algorithm parameters are provided within the corresponding code repositories. The algorithm parameters refer to the system constraints and the sensor model, the resolution and fidelity of the estimation and propagation pipeline, octomap parameters, planning parameters for

the first–layer of planning, and further parameters for the random tree expansion and associated gains of the uncertainty–aware second layer of planning. Finally, visualization and further system parameters can be set.

## 6.3 *Structural Inspection Planner*

For the exact implementation of the `siplanner`, a set of heuristic improvements were implemented to further optimize the behavior of the algorithm. Having rotorocraft–type aerial robots in mind, faster and more rigorous ordering of the viewpoints can be achieved. In particular, the initial iterations of the algorithm conduct viewpoint allocation by not only considering the nearest neighbor on the tour to minimize the distance to, but also the neighbors that are $N_{Neighbour}$ away on both sides. This allows for extending the locality of the viewpoint sampling considerations to a broader neighborhood and therefore better optimization within this locally generated graph. In every iteration, $N_{Neighbour}$ is then decremented to finally reach 1.

### 6.3.1 `siplanner` Open–Source Release

The `siplanner` open–source release can be found directly at https://github.com/ethz-asl/StructuralInspectionPlanner or linked and further documented through https://github.com/unr-arl/informative-planning. The first open–source release was in the form of a ROS Indigo package, while its use is verified for both Jade and Kinetic versions. The open–sourced algorithm is developed for both rotorcraft and fixed–wing systems and supports triangled–meshes for the representation of the inspection structure of interest. Accompanying experimental results are provided at the planner's repository under "Wiki-Example Results". The easiest way to test the planner is through one of the provided demos. For example, one may run two instances of a terminal and execute the following two commands:

```
$ roslaunch koptplanner hoaHakanaia.launch
```

```
$ rosrun request hoaHakanaia
```

In order to write a client based on `siplanner`, the following code may be used as an example. The key steps of the code example below refer to: (a) setting up a publisher for obstacles and mesh visualization. (b) delaying the publishing of display elements to avoid buffer overflow, (c) instantiating the datastructure to call the service with, (d) defining the bounding box of the operating space, (e) filling the vector of `requiredPoses` with all fixed poses that should be included in the path, (f) specifying the minimum incidence angle, the minimal and maximal inspection distances, as well as the number of iterations of the planner, (g) reading and visualizing the mesh, (h) definition of the obstacle regions, (i) calling the `siplanner` service, and (j) reading the mesh from a non–binary STL file.

```
#include <ros/ros.h>
#include "koptplanner/inspection.h"
#include "shape_msgs/SolidPrimitive.h"
#include <cstdlib>
#include <visualization_msgs/Marker.h>
#include <fstream>
#include <geometry_msgs/PolygonStamped.h>
#include <nav_msgs/Path.h>
#include <std_msgs/Int32.h>
#include <ros/package.h>
#include "tf/tf.h"

std::vector<nav_msgs::Path> * readSTLfile(std::string name);

int main(int argc, char **argv)
{
  ros::init(argc, argv, "requester");
  ROS_INFO("Requester is alive");
  if (argc != 1)
  {
    ROS_INFO("usage: plan");
    return 1;
  }

  ros::NodeHandle n;
  ros::Publisher obstacle_pub = n.advertise<visualization_msgs::
    Marker>
    ("scenario", 1);
  ros::Publisher stl_pub = n.advertise<nav_msgs::Path>("stl_mesh",
    1);
  ros::ServiceClient client = n.serviceClient<koptplanner::
    inspection>
    ("inspectionPath");

  ros::Rate r(50.0);
  ros::Rate r2(1.0);
  r2.sleep();

  /* define the bounding box */
  koptplanner::inspection srv;
  srv.request.spaceSize.push_back(1375);
  srv.request.spaceSize.push_back(2165);
  srv.request.spaceSize.push_back(0.001);
  srv.request.spaceCenter.push_back(1375.0/2.0);
  srv.request.spaceCenter.push_back(2165.0/2.0);
  srv.request.spaceCenter.push_back(200.0);

  geometry_msgs::Pose reqPose;

  /* starting pose (comment the push_back if no explicit starting
    pose
     is desired) */
  reqPose.position.x = 300.0;
```

```
reqPose.position.y = 300.0;
reqPose.position.z = 200.0;
tf::Quaternion q = tf::createQuaternionFromRPY(0.0, 0.0, 0.0);
reqPose.orientation.x = q.x();
reqPose.orientation.y = q.y();
reqPose.orientation.z = q.z();
reqPose.orientation.w = q.w();
srv.request.requiredPoses.push_back(reqPose);

/* final pose (remove if no explicit final pose is desired) */
reqPose.position.x = 400.0;
reqPose.position.y = 300.0;
reqPose.position.z = 200.0;
q = tf::createQuaternionFromRPY(0.0, 0.0, 0.0);
reqPose.orientation.x = q.x();
reqPose.orientation.y = q.y();
reqPose.orientation.z = q.z();
reqPose.orientation.w = q.w();
srv.request.requiredPoses.push_back(reqPose);

/* parameters for the path calculation (such as may change
during mission) */
srv.request.incidenceAngle = M_PI/6.0;
srv.request.minDist = 40.0;
srv.request.maxDist = 300.0;
srv.request.numIterations = 20;

/* read STL file and publish to rviz */
std::vector<nav_msgs::Path> * mesh = readSTLfile(ros::package::
  getPath(
  "request")+"/meshes/regularPlanes/rPlane.stl");
for(std::vector<nav_msgs::Path>::iterator it = mesh->begin();
    it != mesh->end() && ros::ok(); it++)
{
  stl_pub.publish(*it);
  geometry_msgs::Polygon p;
  geometry_msgs::Point32 p32;
  p32.x = it->poses[0].pose.position.x;
  p32.y = it->poses[0].pose.position.y;
  p32.z = it->poses[0].pose.position.z;
  p.points.push_back(p32);
  p32.x = it->poses[1].pose.position.x;
  p32.y = it->poses[1].pose.position.y;
  p32.z = it->poses[1].pose.position.z;
  p.points.push_back(p32);
  p32.x = it->poses[2].pose.position.x;
  p32.y = it->poses[2].pose.position.y;
  p32.z = it->poses[2].pose.position.z;
  p.points.push_back(p32);
  srv.request.inspectionArea.push_back(p);
  r.sleep();
}

/* define obstacle regions as cuboids that are coordinate
```

```
104   system aligned */
105   shape_msgs::SolidPrimitive body;
106   body.type = shape_msgs::SolidPrimitive::BOX;
107   body.dimensions.push_back(40.0);
108   body.dimensions.push_back(50.0);
109   body.dimensions.push_back(4.0);
110   srv.request.obstacles.push_back(body);
111   geometry_msgs::Pose pose;
112   pose.position.x = 600.0;
113   pose.position.y = 600.0;
114   pose.position.z = 200.0;
115   pose.orientation.x = 0.0;
116   pose.orientation.y = 0.0;
117   pose.orientation.z = 0.0;
118   pose.orientation.w = 1.0;
119   srv.request.obstaclesPoses.push_back(pose);
120   srv.request.obstacleIntransparancy.push_back(0);
119
120   // publish obstacles for rviz
121   visualization_msgs::Marker marker;
122   marker.header.frame_id = "/kopt_frame";
123   marker.header.stamp = ros::Time::now();
124   marker.ns = "obstacles";
125   marker.id = 0; // enumerate when adding more obstacles
126   marker.type = visualization_msgs::Marker::CUBE;
127   marker.action = visualization_msgs::Marker::ADD;
128
129   marker.pose.position.x = pose.position.x;
130   marker.pose.position.y = pose.position.y;
131   marker.pose.position.z = pose.position.z;
132   marker.pose.orientation.x = pose.orientation.x;
133   marker.pose.orientation.y = pose.orientation.y;
134   marker.pose.orientation.z = pose.orientation.z;
135   marker.pose.orientation.w = pose.orientation.w;
136
137   marker.scale.x = body.dimensions[0];
138   marker.scale.y = body.dimensions[1];
139   marker.scale.z = body.dimensions[2];
140
141   marker.color.r = 0.0f;
142   marker.color.g = 0.0f;
143   marker.color.b = 1.0f;
144   marker.color.a = 0.5;
145
146   marker.lifetime = ros::Duration();
147   obstacle_pub.publish(marker);
148   r.sleep();
149
150   if (client.call)
151   {
152     ROS_INFO("Successfully planned inspection path");srv))
153   }
154   else
155   {
```

```
156     ROS_ERROR("Failed to call service planner");
157     return 1;
158   }
159
160   return 0;
161 }
162
163 std::vector<nav_msgs::Path> * readSTLfile(std::string name)
164 {
165   std::vector<nav_msgs::Path> * mesh = new std::vector<nav_msgs::
        Path>;
166   std::fstream f;
167   f.open(name.c_str());
168   assert(f.is_open());
169   int MaxLine = 0;
170   char* line;
171   double maxX = -DBL_MAX;
172   double maxY = -DBL_MAX;
173   double maxZ = -DBL_MAX;
174   double minX = DBL_MAX;
175   double minY = DBL_MAX;
176   double minZ = DBL_MAX;
177   assert(line = (char *) malloc(MaxLine = 80));
178   f.getline(line, MaxLine);
179   if(0 != strcmp(strtok(line, " "), "solid"))
180   {
181     ROS_ERROR("Invalid mesh file! Give in ascii-format.");
182     ros::shutdown();
183   }
184   assert(line = (char *) realloc(line, MaxLine));
185   f.getline(line, MaxLine);
186   int k = 0;
187   while(0 != strcmp(strtok(line, " "), "endsolid") && !ros::
        isShuttingDown())
188   {
189     int q = 0;
190     nav_msgs::Path p;
191     geometry_msgs::PoseStamped v1;
192     for(int i = 0; i<7; i++)
193     {
194       while(line[q] == ' ')
195         q++;
196       if(line[q] == 'v')
197       {
198         // used to rotate the mesh before processing
199         const double yawTrafo = 0.0;
200         // used to scale the mesh before processing
201         const double scaleFactor = 1.0;
202         // used to offset the mesh before processing
203         const double offsetX = 0.0;
204         // used to offset the mesh before processing
205         const double offsetY = 0.0;
206         // used to offset the mesh before processing
207         const double offsetZ = 0.0;
```

```
208
209         geometry_msgs::PoseStamped vert;
210         char* v = strtok(line+q," ");
211         v = strtok(NULL," ");
212         double xtmp = atof(v)/scaleFactor;
213         v = strtok(NULL," ");
214         double ytmp = atof(v)/scaleFactor;
215         vert.pose.position.x = cos(yawTrafo)*xtmp-sin(yawTrafo)*
    ytmp;
216         vert.pose.position.y =  sin(yawTrafo)*xtmp+cos(yawTrafo)*
    ytmp;
217         v = strtok(NULL," ");
218         vert.pose.position.z =  atof(v)/scaleFactor;
219         vert.pose.position.x -= offsetX;
220         vert.pose.position.y -= offsetY;
221         vert.pose.position.z -= offsetZ;
222         if(maxX<vert.pose.position.x)
223           maxX=vert.pose.position.x;
224         if(maxY<vert.pose.position.y)
225           maxY=vert.pose.position.y;
226         if(maxZ<vert.pose.position.z)
227           maxZ=vert.pose.position.z;
228         if(minX>vert.pose.position.x)
229           minX=vert.pose.position.x;
230         if(minY>vert.pose.position.y)
231           minY=vert.pose.position.y;
232         if(minZ>vert.pose.position.z)
233           minZ=vert.pose.position.z;
234         vert.pose.orientation.x =  0.0;
235         vert.pose.orientation.y =  0.0;
236         vert.pose.orientation.z =  0.0;
237         vert.pose.orientation.w =  1.0;
238         p.poses.push_back(vert);
239         if(p.poses.size() == 1)
240           v1 = vert;
241       }
242       assert(line = (char *) realloc(line, MaxLine));
243       f.getline(line, MaxLine);
244     }
245     p.poses.push_back(v1);
246     p.header.frame_id = "/kopt_frame";
247     p.header.stamp = ros::Time::now();
248     p.header.seq = k;
249     mesh->push_back(p);
250     k++;
251   }
252   free(line);
253   f.close();
254   return mesh;
255 }
```

As with the other two planners, further details on the planning parameters are provided in the corresponding code repository. Among the critical parameters to

be set is the flag for rotorcrafts or fixed–wing aircraft, the model constraints for each, as well as algorithm parameters that influence its behavior (e.g. collision–check interval, iterations of RRT* if no connection could be established, minimum distance for viewpoints to an obstacle and more).

## 6.4 *Terrain Monitoring Planner*

The `tmplanner` is implemented and operates in a finite–horizon alternating between replanning and plan execution, while taking new sensor data into account. The replanning stage consists of two steps, namely: (a) coarse 3D grid search in the UAV configuration space and (b) optimization of this trajectory for maximized information/exploratory gain using an evolutionary scheme.

### 6.4.1 `tmplanner` Open–Source Release

The `tmplanner` open–source release can be found directly at https://github.com/ethz-asl/tmplanner or through the additional and unifying repository https://github.com/unr-arl/informative-planning. The release has been tested to operate with ROS Kinetic, while back compatibility with ROS Indigo and Jade has been verified. The repository contains two stand-alone ROS packages corresponding to discrete (`tmplanner_discrete`) and continuous (`tmplanner_continuous`) variable monitoring, as described in Sects. 4.1 and 4.2, respectively.

The implementation of both planners includes the following key components:

- Node interface for ROS communication (`tmplanner_node` in the respective packages)
- Planning unit
- Mapping unit

A third package, `tmplanner_tools`, provides convenience functions used by both planners, in addition to the infrastructure needed to run examples set-up in the Gazebo-based RotorS environment [44]. As per the examples, the algorithms are intended for use on-board a rotorcraft-type aerial robot equipped with a downward-facing camera. All parameters relevant for mapping (e.g. environment dimensions, grid resolution), sensing (e.g. measurement frequency, FoV angles), and planning (e.g. time budget, optimization parameters) are exposed in seperate file. Note that the complexity of the programs can be easily adapted for online operation even with limited computational resources, e.g. by using a coarser grid for 3D search or limiting the number of optimization iterations in the first or second replanning stages, respectively.

The following steps outline the terminal commands to run an illustrative demo in RotorS for the application of mapping a continuous variable. The same procedure can

be applied to map discrete variables, by replacing for the `tmplanner_discrete` package.

1. Start the simulation environment by running:

```
$ roslaunch tmplanner_continuous
  monitoring_example.launch
```

2. Start the planner node:

```
$ roslaunch tmplanner_continuous tmplanner.launch
```

   Note that the appropriate parameters must be loaded in the launch file.
3. Initialize the finite-horizon planning routine via the rosservice:

```
$ rosservice call /start planning
```

Complete documentation, featuring extended details for the subscribed topics and algorithm parameters, are provided within the code repository, in addition to visualization tools.

## 7 Conclusion

This chapter presented a path planning ensemble addressing the problems of autonomous exploration, terrain monitoring and optimized coverage, alongside their utilization as ROS packages and experimental studies using aerial robots. All the algorithms presented are already open–sourced. In their combination, these tools represent a path planning ensemble capable of supporting important application domains for aerial robots, including those of infrastructure and industry inspection, precision agriculture, security monitoring, and more. The open–source contributions, along with the limited set of assumptions considered, permit their generic utilization for any robotic system for which a boundary value solver is available. Moreover, extensive experimental evaluation and field demonstration using aerial robots support the vision for a collective contribution in the field of path planning that can have a transformable impact, better equip end–users of robotic systems and further enable developers. Finally, experimental datasets are released to allow systematic scientific comparisons.

## References

1. L. Yoder, S. Scherer, Autonomous exploration for infrastructure modeling with a micro aerial vehicle, *Field and Service Robotics* (Springer, Berlin, 2016), pp. 427–440
2. P. Lottes, R. Khanna, J. Pfeifer, R. Siegwart, C. Stachniss, Uav-based crop and weed classification for smart farming, in *2017 IEEE International Conference on Robotics and Automation (ICRA)* (IEEE, 2017), pp. 3024–3031

3. F. Liebisch, M. Popovic, J. Pfeifer, R. Khanna, P. Lottes, A. Pretto, I. Sa, J. Nieto, R. Siegwart, A. Walter, Automatic UAV-based field inspection campaigns for weeding in row crops, in *EARSeL SIG Imaging Spectroscopy Workshop, Zurich* (2017)
4. H. Balta, J. Bedkowski, S. Govindaraj, K. Majek, P. Musialik, D. Serrano, K. Alexis, R. Siegwart, G. Cubber, Integrated data management for a fleet of search-and-rescue robots. J. Field Robot. **34**(3), 539–582 (2017)
5. C. Bolkcom, Homeland security: Unmanned aerial vehicles and border surveillance. DTIC Document, 2004
6. A. Hornung, K.M. Wurm, M. Bennewitz, C. Stachniss, W. Burgard, Octomap: an efficient probabilistic 3d mapping framework based on octrees. Auton. Robot. **34**(3), 189–206 (2013)
7. S.M. LaValle, Rapidly-exploring random trees a new tool for path planning, 1998
8. H.H. González-Banos, J.-C. Latombe, Navigation strategies for exploring indoor environments. Int. J. Robot. Res. **21**(10–11), 829–848 (2002)
9. A. Bircher, M. Kamel, K. Alexis, H. Oleynikova, R. Siegwart, Receding horizon "next-best-view" planner for 3d exploration, in *IEEE International Conference on Robotics and Automation (ICRA)* (2016)
10. C. Papachristos, K. Alexis, Autonomous detection and classification of change using aerial robots, in *IEEE Aerospace Conference* (2017)
11. A. Bircher, M. Kamel, K. Alexis, H. Oleynikova, R. Siegwart, Receding horizon path planning for 3d exploration and surface inspection. Auton. Robot. 1–16 (2016)
12. H. Carrillo, I. Reid, J.A. Castellanos, On the comparison of uncertainty criteria for active slam, in *2012 IEEE International Conference on Robotics and Automation (ICRA)* (IEEE, 2012)
13. M. Bloesch, S. Omari, M. Hutter, R. Siegwart, Robust visual inertial odometry using a direct ekf-based approach, in *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (IEEE, 2015). (to appear)
14. J. Kiefer, General equivalence theory for optimum designs (approximate theory). Ann. Stat. **2**, 849–879 (1974)
15. M. Kamel, T. Stastny, K. Alexis, R. Siegwart, Model predictive control for trajectory tracking of unmanned aerial vehicles using ros, *Robot Operating System (ROS)* (Springer, Cham, 2017)
16. A. Bircher, M. Kamel, K. Alexis, M. Burri, P. Oettershagen, S. Omari, T. Mantel, R. Siegwart, Three-dimensional coverage path planning via viewpoint resampling and tour optimization for aerial robots. Auton. Robot. 1–25 (2015)
17. A. Bircher, K. Alexis, U. Schwesinger, S. Omari, M. Burri, R. Siegwart, An incremental sampling-based approach to inspection planning: the rapidly-exploring random tree of trees, 2016
18. A. Bircher, K. Alexis, M. Burri, P. Oettershagen, S. Omari, T. Mantel, R. Siegwart, Structural inspection path planning via iterative viewpoint resampling with application to aerial robotics, in *IEEE International Conference on Robotics and Automation (ICRA)* (2015), pp. 6423–6430, https://github.com/ethz-asl/StructuralInspectionPlanner
19. K. Alexis, C. Papachristos, R. Siegwart, A. Tzes, Uniform coverage structural inspection path-planning for micro aerial vehicles, 2015
20. C. Papachristos, S. Khattak, K. Alexis, Autonomous exploration of visually-degraded environments using aerial robots, in *2017 International Conference on Unmanned Aircraft Systems (ICUAS)* (IEEE, 2017)
21. C. Papachristos, S. Khattak, K. Alexis, Uncertainty–aware receding horizon exploration and mapping using aerial robots, in *IEEE International Conference on Robotics and Automation (ICRA)* (2017)
22. M. Popovic, G. Hitz, J. Nieto, I. Sa, R. Siegwart, E. Galceran, Online informative path planning for active classification using UAVs, in *IEEE International Conference on Robotics and Automation* (IEEE, Singapore, 2017)
23. M. Popovic, T. Vidal-Calleja, G. Hitz, I. Sa, R. Y. Siegwart, J. Nieto, Multiresolution mapping and informative path planning for UAV-based terrain monitoring, in *IEEE/RSJ International Conference on Intelligent Robots and Systems* (IEEE, Vancouver, 2017)

24. A. Elfes, Using occupancy grids for mobile robot perception and navigation. Computer **22**(6), 46–57 (1989)
25. C.E. Rasmussen, C.K.I. Williams, *Gaussian Processes for Machine Learning* (MIT Press, Cambridge, 2006)
26. S. Reece, S. Roberts, An introduction to gaussian processes for the Kalman filter expert, in *FUSION* (2013), pp. 1–9
27. T. Vidal-Calleja, D. Su, F. D. Bruijn, J.V. Miro, Learning spatial correlations for bayesian fusion in pipe thickness mapping, in *IEEE International Conference on Robotics and Automation* (IEEE, Hong Kong, 2014)
28. C. Richter, A. Bry, N. Roy, Polynomial trajectory planning for quadrotor flight, in *International Conference on Robotics and Automation* (Springer, Singapore, 2013)
29. B. Charrow, S. Liu, V. Kumar, N. Michael, Information-theoretic mapping using Cauchy–Schwarz quadratic mutual information, in *IEEE International Conference on Robotics and Automation* (IEEE, Seattle, 2015), pp. 4791–4798
30. N. Hansen, The CMA evolution strategy: a comparing review. Stud. Fuzziness Soft Comput. **192**(2006), 75–102 (2006)
31. G. Hitz, E. Galceran, M.-È. Garneau, F. Pomerleau, R. Siegwart, Adaptive continuous-space informative path planning for online environmental monitoring, 2016
32. J. O'rourke, *Art Gallery Theorems and Algorithms*, vol. 57 (Oxford University Press, Oxford, 1987)
33. H. González-Baños, A randomized art-gallery algorithm for sensor placement, in *Proceedings of the Seventeenth Annual Symposium on Computational Geometry* (ACM, 2001), pp. 232–240
34. G. Dantzig, R. Fulkerson, S. Johnson, Solution of a large-scale traveling-salesman problem. J. Oper. Res. Soc. Am. **2**(4), 393–410 (1954)
35. S. Lin, B.W. Kernighan, An effective heuristic algorithm for the traveling-salesman problem. Oper. Res. **21**(2), 498–516 (1973)
36. G. Papadopoulos, H. Kurniawati, N.M. Patrikalakis, Asymptotically optimal inspection planning using systems with differential constraints, in *2013 IEEE International Conference on Robotics and Automation (ICRA)* (IEEE, 2013), pp. 4126–4133
37. K. Helsgaun, An effective implementation of the lin-kernighan traveling salesman heuristic. Eur. J. Oper. Res. **126**(1), 106–130 (2000)
38. S. Karaman, E. Frazzoli, Sampling-based algorithms for optimal motion planning. Int. J. Robot. Res. **30**(7), 846–894 (2011)
39. H. Ferreau, C. Kirches, A. Potschka, H. Bock, M. Diehl, qpOASES: a parametric active-set algorithm for quadratic programming. Math. Program. Comput. **6**(4), 327–363 (2014)
40. C. Papachristos, K. Alexis, L.R.G. Carrillo, A. Tzes, Distributed infrastructure inspection path planning for aerial robotics subject to time constraints, in *2016 International Conference on Unmanned Aircraft Systems (ICUAS)* (IEEE, 2016), pp. 406–412
41. C. Papachristos, K. Alexis, Augmented reality-enhanced structural inspection using aerial robots, in *2016 IEEE International Symposium on Intelligent Control* (IEEE, 2016), pp. 1–6
42. P. Oettershagen, T. Stastny, T. Mantel, A. Melzer, K. Rudin, G. Agamennoni, K. Alexis, R. Siegwart, Long-endurance sensing and mapping using a hand-launchable solar-powered uav, 2015
43. F. Furrer, M. Burri, M. Achtelik, R. Siegwart, Rotorsa modular gazebo mav simulator framework, *Robot Operating System (ROS)* (Springer, Cham, 2016), pp. 595–625
44. RotorS: An MAV gazebo simulator, https://github.com/ethz-asl/rotors_simulator