Efficient Autonomous Exploration Planning of Large Scale 3D-Environments

Magnus Selin[1,2], Mattias Tiger[1], Daniel Duberg[2], Fredrik Heintz[1], Patric Jensfelt[2]

***Abstract*—Exploration is an important aspect of robotics, whether it is for mapping, rescue missions or path planning in an unknown environment. Frontier Exploration planning (FEP) and Receding Horizon Next-Best-View planning (RH-NBVP) are two different approaches with different strengths and weaknesses. FEP explores a large environment consisting of separate regions with ease, but is slow at reaching full exploration due to moving back and forth between regions. RH-NBVP shows great potential and efficiently explores individual regions, but has the disadvantage that it can get stuck in large environments not exploring all regions. In this work we present a method that combines both approaches, with FEP as a global exploration planner and RH-NBVP for local exploration. We also present techniques to estimate potential information gain faster, to cache previously estimated gains and to exploit these to efficiently estimate new queries.**

***Index Terms*—Search and Rescue Robots; Motion and Path Planning; Mapping**

I. INTRODUCTION

IN this paper we study the problem of planning for exploring an unknown area. We propose a novel method, Autonomous Exploration Planner (AEP), which improves upon the state-of-the-art method Receding Horizon Next-Best-View planning (RH-NBVP) [1]. RH-NBVP uses a sampling based approach to pick out the next best view point [2] in combination with Rapidly-exploring Random Trees (RRT) [3] to produce traversable paths, weight the samples and execute the first edge before replanning again. The score for each node in the RRT is the volume of unmapped space that would be covered by the sensors from the corresponding pose, weighted with the cost of going there.

In this work *Receding Horizon Next-Best-View planning* is used as a local exploration strategy and is combined with *Frontier exploration* [4] for global exploration. When new information is available close to the agent, the local exploration strategy is used, but when it is far away from any information gain, cached points with high information gain are planned to instead. This leads to a *frontier exploration* behavior, but where frontiers are whole regions of space. This avoids the problem of RH-NBVP getting stuck locally.

Furthermore, our proposed approach (AEP) out-performs RH-NBVP in terms of run-time and computational complexity, without compromising on performance. Our contributions are as follows:

1) The potential information gain g is estimated using ray-casting and cubature integration. The ray-casting is done sparsely to reduce computational load, while still maintaining performance. This reduces the computational complexity of estimating g from inversely quartic growth $\mathcal{O}(1/r^4)$ of RH-NBVP down to inversely linear growth $\mathcal{O}(1/r)$ in our approach, where r is the map resolution.
2) The sampling space for the RRT is reduced by one dimension. Instead of sampling in the full planning space (x, y, z, φ), sampling is done in (x, y, z). This is made possible by a 360 degree ray-casting operation to estimate the potential information gain and selecting the best yaw.
3) The potential information gain g is estimated using cached points from earlier iterations.
4) Cached points are used for interpolation of new queries. Earlier works do not discuss the underlying continuous nature of the potential information gain function g. In this work we model g as a Gaussian Process over the continuous domain of $\mathbb{R}^3$. This allows us to calculate an estimate of g in the entire $\mathbb{R}^3$ together with the uncertainty of the estimate, given the cached points.

The approach is evaluated on three synthetic benchmark environments, one synthetic large 3D environment, and a real experiment running onboard a small indoor drone. Everything presented in this work is released as open source[1] to enable replication of experiments, easy integration and to encourage further development.

II. PROBLEM DESCRIPTION

The problem addressed can be summarized as follows: Given a bounded 3D-volume V that an agent e.g. a drone wishes to explore, which actions should the agent perform to explore V completely and as fast as possible? Each point in this volume, $\mathbf{x} \in V \subset \mathbb{R}^3$, can take the two values: $free$ or $occupied$, i.e. $V(\mathbf{x}) = (free|occupied)$.

Complete exploration means that the agent has created a map M covering the volume V and every point $\mathbf{x}$ in the map, that can be observed by any of the agent's sensors, has been observed as $free$ or $occupied$, i.e. $M(\mathbf{x_{reachable}}) = (free|occupied)$. Points that are non-observable, e.g. hollow spaces without openings, the inside of thick walls or space out of range, will still be $unmapped$ even after the exploration has

This work was partially supported by the Wallenberg AI, Autonomous Systems and Software Program (WASP) and the SSF project FACT.

[1]Selin, Tiger, and Heintz are with Department of Computer and Information Science, Linköping University, Linköping, SE-58183, Sweden, magse761@student.liu.se

[2]Selin, Duberg and Jensfelt are with the Centre for Autonomous Systems at KTH Royal Institute of Technology, Stockholm, SE-10044, Sweden

[1]github.com/mseln/aeplanner

been completed. If no prior information is given, every $\mathbf{x}$ in the map is initially $unmapped$. Due to the active nature of the problem, it has to be solved online.

III. RELATED WORK

Autonomous exploration is a problem that has been studied for more than 20 years. Early methods explored the environment for example by following walls. Frontier exploration [4] was the first exploration method that could explore a generic 2D environment. It defines frontier regions as the borders between free and unmapped space. Exploration is accomplished by extracting frontiers and navigating to them sequentially. This idea is still the basis for much of the literature on mobile robot exploration. Depending on the application different trade-offs are made between paths lengths, information gain, etc. Additional constraints can be added to ensure, for example, that the uncertainty in the robot position is kept low [5] or that the system actively tries to close loops [6] as a way to reduce uncertainty in SLAM. Frontier exploration is further developed and modified for high speed drone flight in [7]. By extracting frontiers in the field of view (FoV) and selecting one that minimizes the change in velocity, they can ensure high speed flight. When there are no more frontiers in the FoV, a plan is made to the nearest frontier outside the FoV using Dijkstra shortest path.

In computer vision and graphics, view planning or sensor placement is the equivalent of exploration in robotics. The aim is to find placements of the sensor to build a model of an object or structure. Much of the work in this domain can be traced back to [8], where the Next-Best-View problem was introduced.

In [2] Gonzalez-Banos and Latombe take the Next-Best-View algorithm into the mobile robot exploration domain. In this domain the cost of moving the sensor, i.e. moving the robot has to be considered. They also introduced a sensor model in the planning process. This is rational since the aim of exploration is to see all of the environment with the equipped sensors rather than going to all the places that have not been explored yet. The score of a view point is weighed with the cost of going there.

The methods so far only consider the gain in information at the next view point. However, a mobile robot will acquire information when moving there as well. By growing a tree outward from the agent, the planning process can take into account the information gain over the entire path. This strategy was presented as the Receding Horizon Next-Best-View planner (RH-NBVP) [1]. To plan paths that reduce positioning uncertainty, [9] presents an extension to the RH-NBVP. A second planning step is performed to find paths that makes sure that the agent revisits known landmarks.

There are also other ways to approach the exploration problem. Potential field methods have been proposed to perform exploration [10], [11], in which the agent follows stream lines into unmapped space. By using a simulation of expanding gas [12] finds frontiers that are used for exploration. Exploration using multiple robots has been done in several works, e.g. [13], [14], to reduce the total exploration time.

In summary, while [7] report that their frontier based exploration method perform better than RH-NBVP, by optimising the way the drone flies so as to maintain high speed, we find RH-NBVP to be the most promising method for general 3D-exploration. The potential gain function allows us to control how to perform the exploration in ways that go beyond directing the sensor towards frontiers between unknown and free space, and it is easy to adapt to any sensor configuration. We therefore base our method on RH-NBVP and show that we can get better result than [7] for low speed flight and there is room for improvement to adapt it for high speed flight, which we leave for future work. We perform our method against RH-NBVP in detail as there is source code available, and against [7] by evaluating it in the same simulated environment.

IV. BACKGROUND

A. Potential Information Gain

The potential information gain at a pose is the volume of the unmapped area that is covered if the agent is placed there. This value is used to give the nodes a score, which is used to prioritize and choose one that should be planned to. The potential information gain is weighted with the negative exponential of the cost to travel there [2], i.e. the score of node $\mathbf{x}$ is

$$s(\mathbf{x}) = g(\mathbf{x}) \exp(-\lambda c(\mathbf{x})), \tag{1}$$

where $g(\mathbf{x})$ is the potential information gain in $\mathbf{x}$, $c(\mathbf{x})$ is the cost of going to $\mathbf{x}$ and λ is a coefficient to control how much distance should be penalized.

Different λ values make the agent act differently. A high λ penalizes movement hard and makes the agent perform careful exploration nearby before moving into the next region, while a low λ will have the effect that the agent will move faster towards completely unexplored space, missing details that it has to go back to later and fill in.

B. Receding Horizon Next-Best-View planning (RH-NBVP)

In RH-NBVP the score $s(\mathbf{x})$ is used for exploration planning by growing an RRT [3] outward from the agent [1]. Each node $\mathbf{x}$ in the RRT is given the score $s(\mathbf{x}) + s(p(\mathbf{x}))$, where $p(\mathbf{x})$ is the parent node of $\mathbf{x}$.

When the tree has been grown to a predetermined size, N, the branch that leads to the node with the highest score is extracted. Only the first edge of this branch is executed, after which the planning process is repeated and a new RRT is grown. The remainder of the best branch is kept as a seed to next iteration.

If the RRT has been grown to a size of N nodes, but the score of the best node is still under a threshold g_{zero}, the tree is grown further until the best node has a score greater than g_{zero}. If the tree reaches a size of N_{max} and the currently best node still has a score less than g_{zero}, the exploration process is considered complete and terminates.

V. PROPOSED APPROACH

Our method, Autonomous Exploration Planner (AEP), builds on RH-NBVP [1]. We observe that when an agent using RH-NBVP has explored everything in its nearby surroundings and the nearest frontier is far away, this frontier will typically have a very low score and the agent tend to terminate the exploration prematurely. This is caused by the exponential decay in the score as a function of cost of travel. Either small scores are ignored and the exploration will be incomplete or all scores above zero will be included, but then the exploration will focus on very small information gains nearby that could be skipped. This means that large environments becomes expensive to explore. This can partly be handled by careful tuning of the λ parameter. However, this has to be done for every environment and it typically requires several attempts. To be able to continue exploration also in large environments and reduce the need for tuning, we suggest to marry the ideas of Next-Best-View planning and frontier based exploration. We use the former for local exploration planning and the latter for global planning.

When the agent has explored everything in its nearby surroundings, the nearest frontier will be far away. The score for nodes at the frontier, if any found, have dropped down to almost zero due to the exponential weighting. To continue exploration also in large environments, we cache nodes with high potential information gain from previous RRTs and consider them as planning targets. This addition will lead to a frontier exploration behavior [4] on the global scale, while the receding horizon NBV exploration behavior is still kept on the local scale.

A. The Potential Information Gain Function (g)

In [2] $g(\mathbf{x})$ is defined for all $\mathbf{x} \in (x, y, z, \varphi)$. We define instead $g(\mathbf{x})$ for $\mathbf{x} \in (x, y, z)$, and let φ be the value that maximizes $g_\varphi(\mathbf{x}, \varphi)$, i.e.

$$g(\mathbf{x}) = \arg\max_{\varphi} g_\varphi(\mathbf{x}, \varphi) \tag{2}$$

That is, the potential information gain function $g(\mathbf{x})$ is defined as the volume of the unmapped space that would be covered when the agent is located in position $\mathbf{x}$ and face the direction that would cover most unmapped space.

This is efficiently calculated by doing ray-tracing 360° around the agent, calculating the potential information gain for every narrow slice and using sliding-window summation to find the information gain for every yaw angle. The yaw angle that corresponds to the maximum value for $g_\varphi(\mathbf{x}, \varphi)$ is then chosen. The estimation of $g_\varphi(\mathbf{x}, \varphi)$ is further discussed in the implementation section VI. $g(\mathbf{x})$ is dependent on the FoV since $g_\varphi(\mathbf{x}, \varphi)$ also depends on the FoV.

Calculating the best yaw instead of sampling yaw angles reduces the sampling space from four dimensions (x, y, z, φ) to three (x, y, z). As the experiments show, this leads to more efficient exploration.

B. Caching and Estimation

We cache queries, made when growing the RRT, for later use. These are observations of the underlying continuous potential information gain function g. When a new point is queried, we first assess if it can be confidently estimated from the cached points. If the posterior variance for the cached point is low enough, $\sigma^2(\mathbf{x}) \leq \sigma^2_{\text{thres}}$, it will be used. Otherwise, the potential information gain will be calculated explicitly in the queried point and the result will be added to the cache.

In this work we view g as a continuous function. This function was presented by [2], but it was only sampled from and no attempt was made to explicitly estimate it. In their work they present an image of a map with points sampled from this function. We have taken the same map (shown in figure 1), and calculated the information gain for every point in the map to illustrate the continuous nature of g.

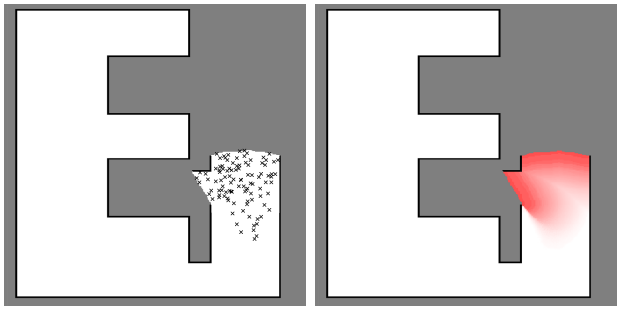


Fig. 1: Left: figure reconstructed from [2] that shows sampled points in which the potential information gain has been evaluated. Right: the continuous potential information gain function g evaluated over the same map. Red means high information gain. White means zero information gain.

A Gaussian Process [15] is used to model the continuous potential information gain function g, with cached points as observations of the latent function. In figure 2b a Gaussian Process predictive distribution has been evaluated on a grid based on the observations shown in figure 2a.

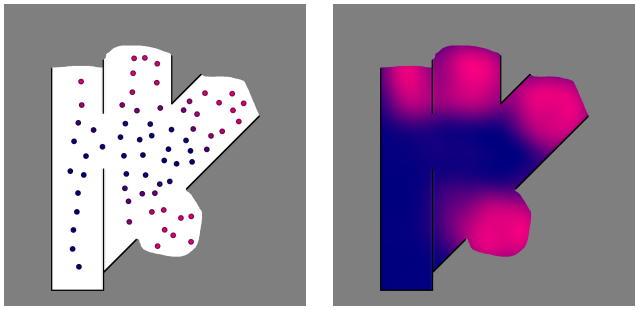
New measurements might affect g and we need to recalculate cached points that are in the range of being possibly affected. In our system this corresponds to points that are within double the distance of the maximum sensing range from the agent's position. If the cached point has a value of zero, it will not be recalculated as g is a monotonic decreasing function over time given our assumptions and it cannot be lower than zero.

C. Frontier Exploration

We suggest to define the frontiers as nodes with high potential information gain from previously expanded RRTs, i.e. the points cached in the previous section. These cache points support both faster calculation of the potential information gain and the frontier exploration. It can clearly be seen in figure 2a how cached points have high information gain close to the frontier (the border between free and unmapped space).

The exploration process is considered complete when there is no potential information gain nearby and there are no more cached points with high value.

A possible direction forward is to investigate ways to base the exploration strategy on the estimates from the Gaussian process directly, but this is left for future work.



(a) The map as seen from above with cached points. Bright pink means high potential information gain, dark blue means low.

(b) The Gaussian Process posterior mean over g given the observations (cached points) from figure 2a.

Fig. 2: Both images illustrates the map during the exploration process. The black area is occupied, white free and gray unmapped.

VI. IMPLEMENTATION

This section describes key details of our method. As an underlying map representation we use OctoMap [16].

A. Estimation of $g_\varphi(\mathbf{x}, \varphi)$ Using Sparse Ray Casting

For a given $\mathbf{x}$ the value of $g_\varphi(\mathbf{x}, \varphi)$ is estimated by casting rays outward from the sensor and summing up all the unmapped volume elements that the ray crosses.

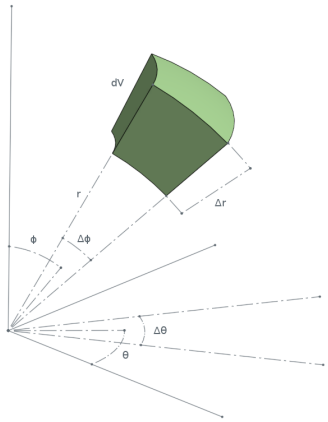


Fig. 3: Volume element dV

A volume element dV is depicted in figure 3 at radius r away from the sensor in direction (θ, ϕ). Its dimensions are $(\Delta_r, \Delta_\theta, \Delta_\phi)$ which gives a volume

$$\begin{aligned} dV(r,\theta,\phi) &= \int_{\theta_1}^{\theta_2}\int_{\phi_1}^{\phi_2}\int_{r_1}^{r_2} \gamma^2 \sin(\alpha) d\gamma d\alpha d\beta \qquad (3) \\ &= \left(2r^2\Delta_r + \frac{1}{6}\Delta_r^3\right)\Delta_\theta \sin(\phi)\sin(\Delta_\phi/2)) \end{aligned}$$

where

$$\begin{cases} r_1 = r - \Delta_r/2 & r_2 = r + \Delta_r/2 \\ \phi_1 = \phi - \Delta_\phi/2 & \phi_2 = \phi + \Delta_\phi/2 \\ \theta_1 = \theta - \Delta_\theta/2 & \theta_2 = \theta + \Delta_\theta/2 \end{cases}. \qquad (4)$$

The potential information gain for that volume element is

$$g_{dV}(r,\theta,\phi) = \begin{cases} dV(r,\theta,\phi) & \text{if } M(r,\theta,\phi) \text{ is } unmapped \\ 0 & \text{otherwise} \end{cases} \qquad (5)$$

For a given yaw direction φ, the potential information gain is the sum of the potential information gain of all volume elements inside the FoV.

$$g_\varphi(\varphi) = \sum_{\theta=\varphi-fov_\theta/2}^{\varphi+fov_\theta/2} \sum_{\phi=-fov_\phi/2}^{fov_\phi/2} \sum_{r=0}^{max_r \text{ or obstacle hit}} g_{dV}(r,\theta,\phi) \qquad (6)$$

The information gain given a yaw direction and a position $g_\varphi(\mathbf{x}, \varphi)$ is simply $g_\varphi(\varphi)$ with the world frame translated so that the sensor is in the origin.

B. Collision Checking

The collision checking between two positions (p_1, p_2) is done efficiently by querying the OctoMap for all voxels inside the bounding box spanning the space between the two points, expanded with the radius of the bounding sphere around the agent r_b.

All voxels inside the bounding box are then queried, if marked as occupied, it is checked whether it is inside a cylinder with end points between p_1 and p_2, and has the radius r_b. It is also checked whether the voxel is inside one of the spheres with radius r_b and origin in p_1 and p_2.

If an occupied voxel is inside the cylinder or any of the spheres, the path is marked as not collision free, otherwise it is considered free to traverse.

C. Gaussian Process Interpolation

Each measurement of the underlying function g is expensive and so is querying the current world model represented by OctoMap. The physically motivated assumption that g is a continuous function that varies smoothly across space implies that collected measurements will contain information about not yet measured points on g. Uncertainty in g across continuous space is represented by placing a Gaussian process prior on g, $g \sim GP(m(\cdot), k(\cdot,\cdot))$, modeling g with a distribution over functions [15].

Using a Gaussian likelihood makes the posterior a closed form solution, and it is possible to infer (interpolate) the value of g everywhere in $\mathbb{R}^3$ as a mean and a variance, conditioned on the observed data (cached points). For example, for $\mathbf{x}_* \in \mathbb{R}^3$ then $g(\mathbf{x}_*) \sim \mathcal{N}(\mu_{\mathbf{x}_*}, \sigma^2_{\mathbf{x}_*})$.

A Gaussian process is a Bayesian nonparametric model fully defined by its mean function $m(\cdot)$ and kernel (covariance) function $k(\cdot,\cdot)$. Here zero mean is assumed $m(\cdot) = 0$ for convenience and the RBF-kernel $k(\mathbf{x}_i, \mathbf{x}_j) = \exp\left(-\frac{1}{2} || \mathbf{x}_i - \mathbf{x}_j ||^2\right)$ is used because of the smooth shape of g.

If the posterior variance is below a threshold σ^2_{thresh} when $g(\mathbf{x}_*)$ is evaluated for a query then the posterior mean $\mu_{\mathbf{x}_*}$ is returned. Otherwise the query is performed explicitly and the node $\mathbf{x}_*$ (together with $g(\mathbf{x}_*)$) is added to the cache. For computational reasons a local GP is used, where only points within a certain distance of $\mathbf{x}_*$ are used for estimating $g(\mathbf{x}_*)$.

Each data point $\mathbf{x} = (x, y, z)$ also has an associated yaw angle, which is the yaw in that point that maximizes $g_\varphi(\mathbf{x}, \varphi)$. When a point is queried it is assigned the mean value of the GP posterior and the yaw angle of the nearest neighbor.

D. Computational Complexity

The computational complexity is critical, since the agent often has limited computational resources. Especially if it is an aerial vehicle. For every iteration of the algorithm an RRT is grown with N nodes. For every added node one gain estimation and one collision check has to be performed.

1) Gain estimation: The complexity of the gain estimation is the number of horizontal rays n times the number of vertical rays m divided by the resolution of the map r, i.e. $\mathcal{O}(nm/r)$. An improvement over RH-NBVP ($\mathcal{O}(1/r^4)$[1]).

2) Collision checking: For collision checking the number of voxels, inside the bounding box spanning the start and the end point, are inversely cubically proportional to the map resolution, i.e. $\mathcal{O}(1/r^3)$.

3) Total computational complexity: The total computational complexity per iteration is $\mathcal{O}(N(nm/r + 1/r^3))$. The number of iterations needed scales linearly with the volume V of the environment. This gives the overall computational complexity for the exploration problem using our method (AEP) as $\mathcal{O}\left(VN\left(\frac{nm}{r} + \frac{1}{r^3}\right)\right)$.

VII. EXPERIMENTAL EVALUATION

We evaluate our method (AEP) in the context of a small indoor drone. As [1] is the method against we primarily compare, we perform some of the experiments in the same environment as [1] (Fig. 4a). We perform both simulated and real world experiments.

Simulated experiments have been conducted in three different environments to evaluate how the different parts of the proposed method contribute to the overall performance.

All tests have been performed under the following conditions:

- The agent starts in the origin with zero yaw angle.
- The agent performs an initial action, which is to go 1 meter forward. This to make sure that the planning is performed with some initial information at hand.
- A hard time limit is set to 20 minutes, to limit the time which an experiment can take.

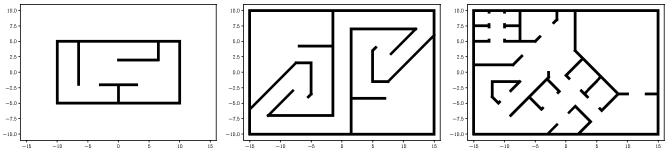
Unless specified otherwise these parameters are used:

Map res.	r	0.1	RRT max len.	l	1 m
Nodes in RRT	N	30	Max nodes	N_{max}	400
Horiz. rays	n	10	Vert. rays	m	10
Gain thres.	g_{zero}	2	Var. thresh.	σ^2_{thres}	0.2
Degress. coeff.	λ	0.5	Bounding rad.	r_b	0.75 m

A. Effects of Sparse Ray Casting and Collision Checking

The *apartment* environment, shown in figure 4a, is a quite simple environment, but used for computational benchmarking (also used in [1]). In this environment, exploration is performed with two different resolutions (0.4 m and 0.1 m) to investigate influence resolution has on the runtime. Each resolution is tested with the following configurations:

- AEP: Our method.
- RH-NBVP[2]: No modifications except for interfacing with our simulation environment.
- RH-NBVP+C: RH-NBVP with efficient collision checking.
- RH-NBVP+RC: RH-NBVP+C with sparse ray gain estimation.



(a) Apartment (b) Maze (c) Office

Fig. 4: The three environments used for benchmarking.

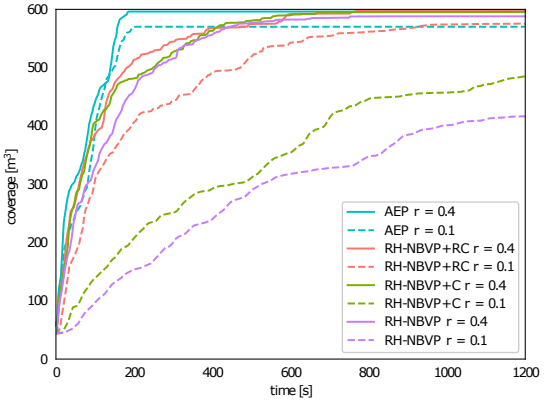


Fig. 5: Exploration progress for resolutions 0.1 and 0.4 m, $N = 15$, with sparse ray-casting and more efficient collision checking turned on or off in environment Fig 4a.

	per	gain estimation iteration	gain estimation node	collision checking iteration	collision checking node
AEP	r=0.4	0.0786	0.0042	0.0132	0.0003
AEP	r=0.1	0.0556	0.0033	0.0735	0.0029
RH-NBVP+RC	r=0.4	0.0160	0.0009	0.0032	0.0002
RH-NBVP+RC	r=0.1	0.0368	0.0020	0.0168	0.0009
RH-NBVP+C	r=0.4	0.1043	0.0046	0.0039	0.0002
RH-NBVP+C	r=0.1	10.4860	0.6349	0.0135	0.0008
RH-NBVP	r=0.4	0.0981	0.0042	0.6648	0.0277
RH-NBVP	r=0.1	10.0713	0.6317	4.8951	0.3073

TABLE I: Computational times in seconds for gain estimation and collision checking.

Figure 5 shows clearly that RH-NBVP performs similarly with or without sparse ray-casting and efficient collision checking when the resolution is 0.4 m. When the resolution is increased to 0.1 m, a big difference can be noticed between whether ray-casting and efficient collision checking is enabled or not. With both enabled, the exploration time is almost independent of resolution.

When the sparse ray-casting has been disabled, the exploration process takes significantly more time and it cannot finish within the time limit of 20 minutes. We can see in table I that

[2]RH-NBVP has been taken from the nbvplanner git repo of ETHZ ASL `https://github.com/ethz-asl/nbvplanner`

the computational time has increased from 0.098 s to 10.07 s per iteration, as opposed to the increase from 0.016 s to 0.037 s when the sparse ray casting estimation was enabled. This means that the drone has to stand still in the air for about 10 seconds every iteration performing calculations.

Disabling the efficient collision checking makes the exploration process even slower (almost 5 seconds per iteration) with the higher resolution (0.1 m).

Our method (AEP) explores the entire apartment in less than 200 seconds on average, no matter the resolution. The computational time for collision checking grows with a factor 6 for the finer resolution, but it is still low enough not to impact the exploration progress. The computational time for gain estimation actually shrinks slightly. This is probably due to that it is so small, and other overhead processes are taking time.

B. Global Exploration Planning Using Frontiers

In this experiment, the exploration is performed with and without frontier exploration enabled, denoted AEP and AEP-F respectively. The tests are conducted in the *maze* environment (fig. 4b). Figure 6 shows how AEP manages to explore the first half completely in 240 seconds and reaches the other side of the maze in 400 seconds. The AEP-F gets stuck in the first half. For high λ it never gets out, since it is only focusing on very small information gains in the first half. With a low λ it eventually manages to get out and reaches the second half after 600 seconds. Lower λ will make exploration go faster forward but will be less careful along the way. Figure 6 shows one run for each method. Repeated experiments confirm that these curves are representative.

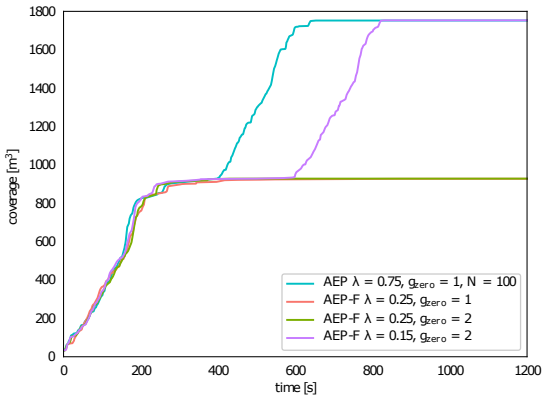


Fig. 6: Exploration progress in the maze environment with frontier exploration enabled or disabled.

In figure 7 the paths of two runs are shown, the blue one with frontier exploration enabled and $\lambda = 0.75, g_{zero} = 1$, and the red one with frontier exploration disabled and $\lambda = 0.25, g_{zero} = 1$. Both runs starts in the origin. The different λs were chosen because in AEP we want the potential information gain to decay fast with respect to distance, so that global exploration can take over when the gain is too low. If this setting were to be used with AEP-F instead, the exploration would be terminated too early in this case, as soon as the right side has been completely explored.

The blue path with frontier exploration (AEP), explores everything on the right side first. Then it makes a plan to the next frontier, in this case where it started. This path is the dashed line from the right side back to the origin. When at the frontier, it explores the left side completely, and when there is no more information gain left it terminates immediately.

The red path without frontier exploration (AEP-F), explores the right side similarly to the blue path. However, when the right side is explored completely it gets stuck there, looking for very small gains. After several iterations it finally manages to find its way to the frontier again. This relies on randomly sampling the frontier. Repeated experiments shows that it can find its way out sometimes after not too long time, but other times it never finds its way out. It takes long time between everything being explored and the termination condition is met. This can be seen in figure 7 by looking at how much the red line goes back and forth on the right side before leaving for the left side. This is controlled by g_{zero}. Too high g_{zero} and exploration terminates too early, too low g_{zero} and exploration never finishes.

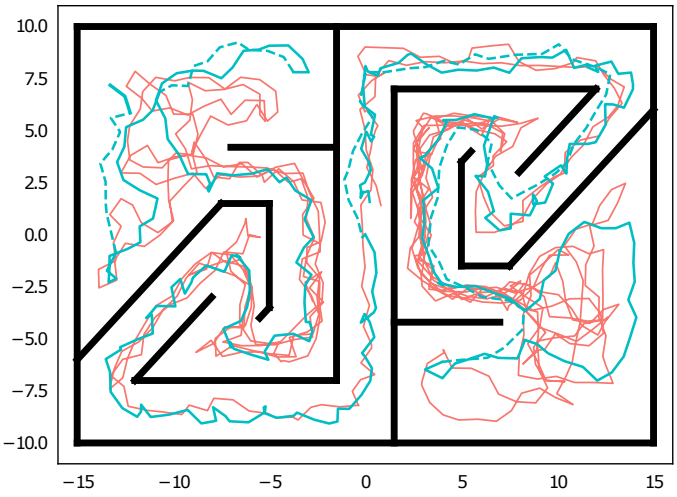


Fig. 7: The blue line shows the path for when frontier exploration is enabled and the red when it has been disabled. Solid line is the local exploration strategy, while the dashed line is a path that has been planned to a frontier further away. We clearly see that the blue path is shorter than the red and that the drone moves less back and forth.

C. Best Yaw

The *office* scenario (see figure 4c) is designed to resemble a normal office. The environment is consciously made varied to test different aspects of the methods, for example a big meeting room, smaller cubicles and a room not connected with any walls. This environment will be used to compare all methods and configurations against each other.

Figure 8 shows the results for our method (AEP), with frontier exploration disabled (AEP-F), and with best yaw also disabled (AEP-FY). Initially AEP-F performs better, but eventually AEP-FY catches up and even passes AEP-F. An explanation for this could be that AEP-F is indeed better at exploration when there is something to explore nearby.

However, AEP-F gets stuck in the same space and for these experiments it seems like AEP-FY finds its way out faster, due to the randomness of the RRT algorithm.

D. Summary of Method Comparison

Figure 8 and table II also summarizes the result of all methods in the office environment (fig. 4c). We can see that our method, AEP, covers space very well without getting stuck. Figures 6 and 7 also support this claim.

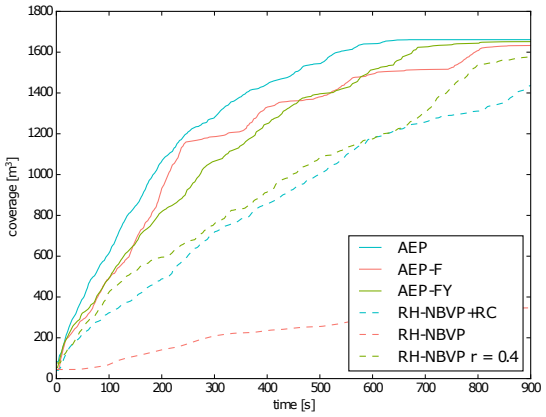


Fig. 8: Exploration progress in the office environment.

	per	gain estimation iteration	gain estimation node	collision checking iteration	collision checking node
AEP	r=0.1	0.0768	0.0025	0.1313	0.0040
AEP-F	r=0.1	0.0687	0.0023	0.1183	0.0039
AEP-FY	r=0.1	0.0679	0.0021	0.0675	0.0021
RH-NBVP+RC	r=0.1	0.0267	0.0009	0.0606	0.0020
RH-NBVP	r=0.1	18.7900	0.6263	10.7180	0.3573
RH-NBVP	r=0.4	0.1563	0.0042	1.3655	0.0371

TABLE II: Office environment computational time (seconds).

The difference between AEP and AEP-F might not seem so big, but note that the environment does not contain that many dead ends. The maze (Fig. 4b) shows the weakness of AEP-F, that it gets stuck, and only with a very low λ it manages to get out. AEP on the other hand manages to reach the left part of the map after 400 seconds (seen in Fig. 6). RH-NBVP suffers from its expensive gain estimation and collision checking. It performs much better when the resolution is turned down to 0.4 m or those functions have been changed for the faster ones proposed in this work.

RH-NBVP+RC still performs slower than AEP-FY, although they should be equivalent. This is likely explained by different implementations and integration of measurements into the OctoMap.

E. Large 3D World

We have tested our method in the *Power plant* scenario, also used in [7], which can be obtained from the Gazebo model library[3] . The scenario is (33 x 31 x 26) m. The fully mapped world is shown in figure 9. We have used the same map resolution, FoV and camera range as in [7] (table III), to make the results comparable with their.

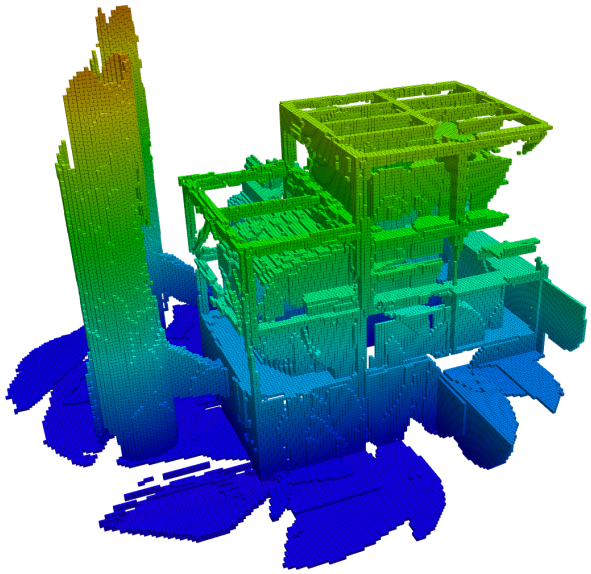


Fig. 9: Exploration in *Power plant* scenario also used in [7].

Map res.	0.2 m	Camera range	7 m
FoV	115 x 60 deg	RRT max len.	3 m
λ	0.75	N_{max}	200
N	50		

TABLE III: Parameters used in *Power plant* scenario.

v_{max} m/s	AEP	Rapid [7]
0.7	1185 ± 95	1245 ± 151
1.5	1037 ± 87	717 ± 94
2.5	941 ± 91	582 ± 26

TABLE IV: Exploration times for *Power plant* scenario.

Table IV shows the exploration times for AEP and Rapid. AEP performs exploration faster than Rapid for $v_{max} = 0.7$ m/s, but slower for the higher speeds. The results cannot be directly compared since we are not running their method in our simulator, but it shows that our AEP method is competitive although not yet leveraging the benefits of high-speed flights.

F. Real World Experiments

The method was tested on a real drone in our indoor drone lab. The size of the area the drone can navigate is (9 m x 5 m x 2.5 m). Mocap was used to get the pose of the drone, everything else was running onboard. The environment and the final OctoMap can be seen in figure 10. The initial position of the drone was in the middle of the environment. The drone started off by flying into the center of the smaller, inner, area and mapped it by rotating in place, thereafter it proceeded to the bigger room and mapped it completely. Videos of the real world experiment can be found at: https://www.youtube.com/playlist?list=PL5wRR7C61QDYQUtvWw5_QC0LM_ndGhbaQ.

VIII. SUMMARY & CONCLUSION

We have presented a new exploration planner (AEP) that combines local and global planning for exploration, thereby

[3] https://bitbucket.org/osrf/gazebo_models/src

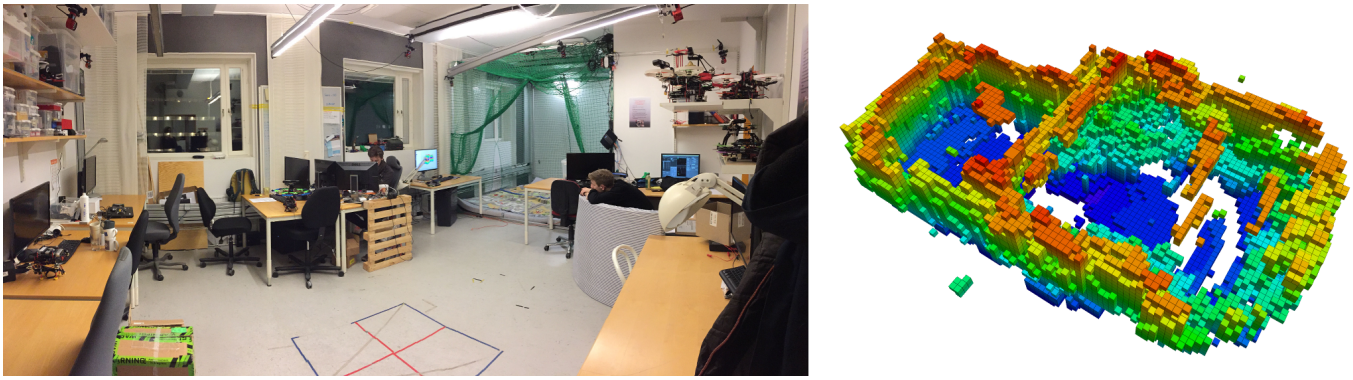


Fig. 10: Real world experiment consisting of two complex rooms and with exploration running onboard a real drone.

combining the strengths of receding horizon NBV planning and frontier exploration. Our planner can explore large unknown environments fast and without getting stuck. By using a new way of estimating potential information gain, our method scales well with map resolution as well as with the size of the environment. The exploration process is sped up by a 360 degree potential information gain estimation to make the agent point in the direction where most unmapped space is covered, instead of relying on randomly sampling the orientation. Computational time is saved by reusing previously estimated potential information gains. Cached potential information gain with a high value will always be close to a frontier, which we use for global planning.

With our contributions, we have shown that RH-NBVP, can perform on par with methods optimized for high speed drone flights. As future work we propose that a kinodynamic model is introduced in the RRT to incorporate a better model for the motion of the drone and thereby allow the drone to maintain higher speeds. Apart from that we also plan to investigate strategies based directly on the estimated continuous information gain function in the future.

References

[1] A. Bircher, M. Kamel, K. Alexis, H. Oleynikova, and R. Siegwart, "Receding horizon "next-best-view" planner for 3d exploration," in *IEEE International Conference on Robotics and Automation (ICRA)*, May 2016, pp. 1462–1468.

[2] H. H. Gonzalez-Banos and J.-C. Latombe, "Navigation strategies for exploring indoor environments," *The International Journal of Robotics Research*, vol. 21, no. 10-11, pp. 829–848, Oct. 2002.

[3] S. M. Lavalle, "Rapidly-exploring random trees: A new tool for path planning," 1998.

[4] B. Yamauchi, "A frontier-based approach for autonomous exploration," *The International Symposium on Computational Intelligence in Robotics and Automation (CIRA)*, pp. 146–151, Jul. 1997.

[5] C. Stachniss, G. Grisetti, and W. Burgard, "Information gain-based exploration using rao-blackwellized particle filters," in *Robotics: Science and Systems (RSS)*, 2005.

[6] C. Stachniss, D. Hahnel, and W. Burgard, "Exploration with active loop-closing for fastslam," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, vol. 2, Sep. 2004, pp. 1505–1510.

[7] T. Cieslewski, E. Kaufmann, and D. Scaramuzza, "Rapid exploration with multi-rotors: A frontier selection method for high speed flight," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Sep. 2017, pp. 2135–2142.

[8] C. Connolly, "The determination of next best views," in *IEEE International Conference on Robotics and Automation (ICRA)*, vol. 2, Mar. 1985, pp. 432–435.

[9] C. Papachristos, S. Khattak, and K. Alexis, "Uncertainty-aware receding horizon exploration and mapping using aerial robots," in *IEEE International Conference on Robotics and Automation (ICRA)*, May 2017, pp. 4568–4575.

[10] E. P. e Silva, P. M. Engel, M. Trevisan, and M. A. Idiart, "Exploration method using harmonic functions," *Robotics and Autonomous Systems*, vol. 40, no. 1, pp. 25–42, 2002.

[11] R. Shade and P. Newman, "Choosing where to go: Complete 3d exploration with stereo," in *IEEE International Conference on Robotics and Automation (ICRA)*, May 2011, pp. 2806–2811.

[12] S. Shen, N. Michael, and V. Kumar, "Autonomous indoor 3d exploration with a micro-aerial vehicle," in *IEEE International Conference on Robotics and Automation*, May 2012, pp. 9–15.

[13] A. Howard, L. E. Parker, and G. S. Sukhatme, "Experiments with a large heterogeneous mobile robot team: Exploration, mapping, deployment and detection," *The International Journal of Robotics Research*, vol. 25, no. 5-6, pp. 431–447, 2006.

[14] C. Dornhege, A. Kleiner, A. Hertle, and A. Kolling, "Multirobot coverage search in three dimensions," *J. Field Robot.*, vol. 33, no. 4, pp. 537–558, Jun. 2016.

[15] C. E. Rasmussen, "Gaussian processes for machine learning," 2006.

[16] A. Hornung *et al.*, "Octomap: An efficient probabilistic 3d mapping framework based on octrees," *Autonomous Robots*, vol. 34, pp. 189–206, Apr. 2013.