Efficient Autonomous Exploration Planning of Large Scale 3D-Environments

Magnus Selin
magse761@student.liu.se
IDA, Linköping University and CAS, Royal Institute of Technology

Mattias Tiger
mattias.tiger@liu.se
IDA, Linköping University

Daniel Duberg
dduberg@kth.se
CAS, Royal Institute of Technology

Fredrik Heintz
fredrik.heintz@liu.se
IDA, Linköping University

Patric Jensfelt
patric@kth.se
CAS, Royal Institute of Technology

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.

Overview

The Autonomous Exploration Planner (AEP) is a further development from the Receding Horizon Next Best View Planner (RH-NBVP). When developing AEP, we looked at the shortcomings of RH-NBPV:

- RH-NBVP does not scale well with map resolution.
- RH-NBVP can get stuck in already explored dead ends.

AEP solves these problems with **sparse ray-casting** to estimate the potential information gain, **caching and re-usage of already calculated information gains** and **frontier exploration as a global exploration strategy**, while using RH-NBVP for local exploration.

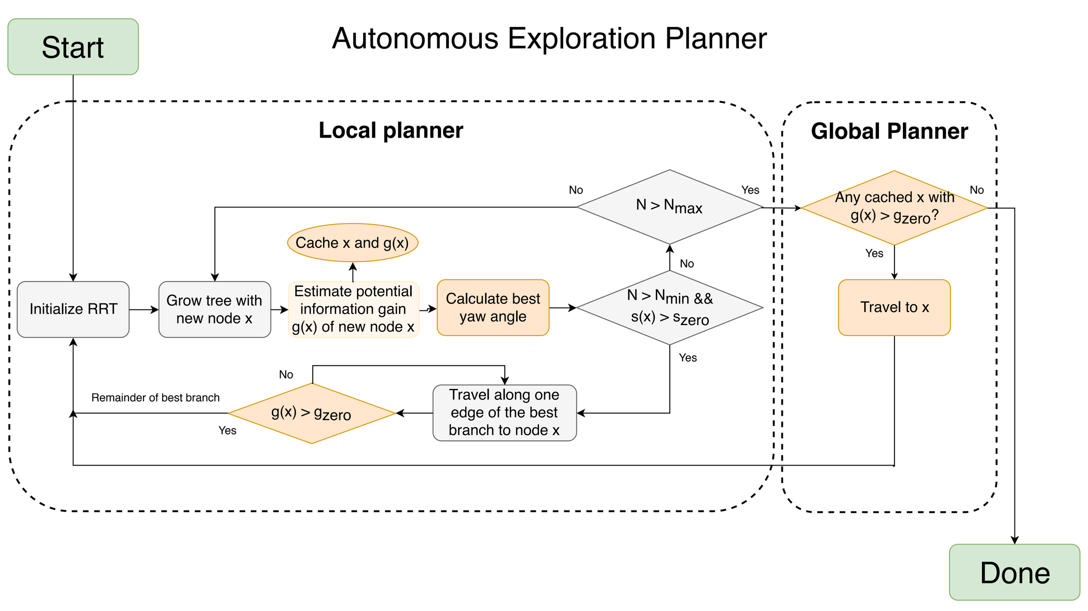
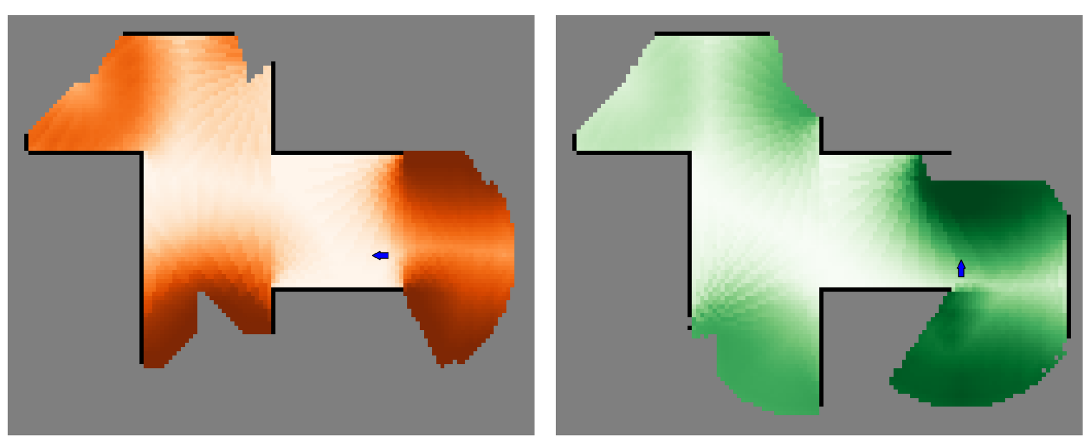


Figure 1: Overview of AEPlanner

Figure 1 shows an overview of the AEP. AEP grows an RRT outwards from the agent's current position. In every node in the RRT the potential information gain is estimated. The score for every node is the score of the parent node plus the gain of the node weighted with the travel cost to reach it. When the tree has reached N_{max} nodes, the best branch is extracted and the first action executed. The remainder of the best branch is saved and used for the next iteration. If there are no nodes with high enough score the global exploration planner takes over.

The potential information gain function g

The potential gain function (g) describes how much information that could potentially be gained in a point. It is defined as the volume of all unmapped space that would be covered by a sensor placed in that point, concidering the field of view and range of the sensor.



(a) Potential information gain function g (b) Weighted g, $s = g\exp(-\lambda c)$

Figure 2: Both figures show a map of a partially explored environment. Black is occupied (walls) and gray unmapped. Everything else is free space. The agent is located at the blue arrow. The left figure shows the potential information gain function for every position "pixel". The right figure shows the score function, which also takes the travel cost c into account. λ is a parameter which controls how much travelling should be punished.

Sparse ray casting

The estimation of g given a position $\mathbf{x}$ is efficiently done by performing sparse ray tracing. We look at small volume elements dV in spherical coordinates, which are defined as:

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

Rays are cast, in the field of view, outwards from the sensor until they hit an obstacle or the max range is reached. Every part of a ray that passes through unmapped space, will contribute with dV to the potential information gain.

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

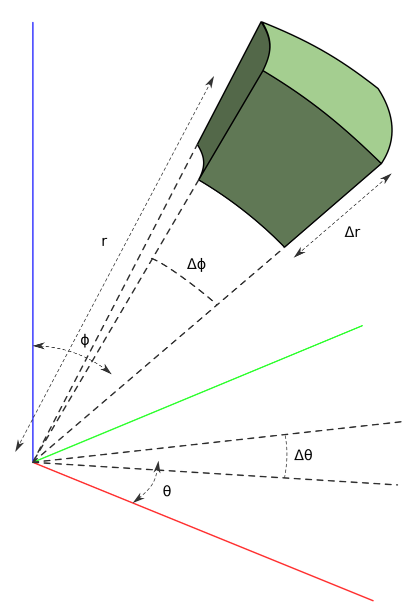
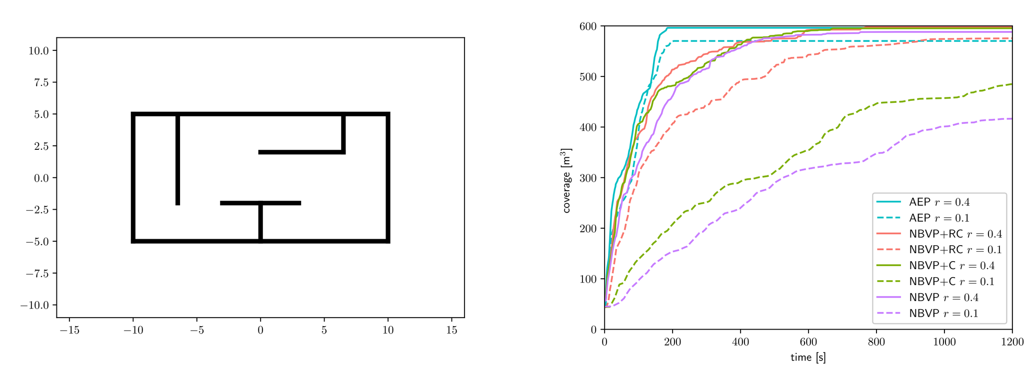


Figure 3: Volume element dV

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

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



(a) Apartment (b) Results for apartment scenario

Figure 4: The apartment scenario is used for benchmarking of AEP against NBVP. Results are also provided for NBVP with modified ray tracing and collision detection.

Calculating the information gain this way greatly reduces the computational time. In figure 4b our method AEP is compared against RH-NBVP. We can see that our method manages to explore the entire environment within 200 seconds independent of the map resolution. RH-NBVP on the other side performs decently for map resolution 0.4 m, but when the resolution is increased to 0.1 m the time taken for gain estimation is so long that RH-NBVP cannot finish exploration within 1200 seconds. After substituting both the gain estimation and collision detection methods (NBVP+RC), the entire environment is explored within the time limit.

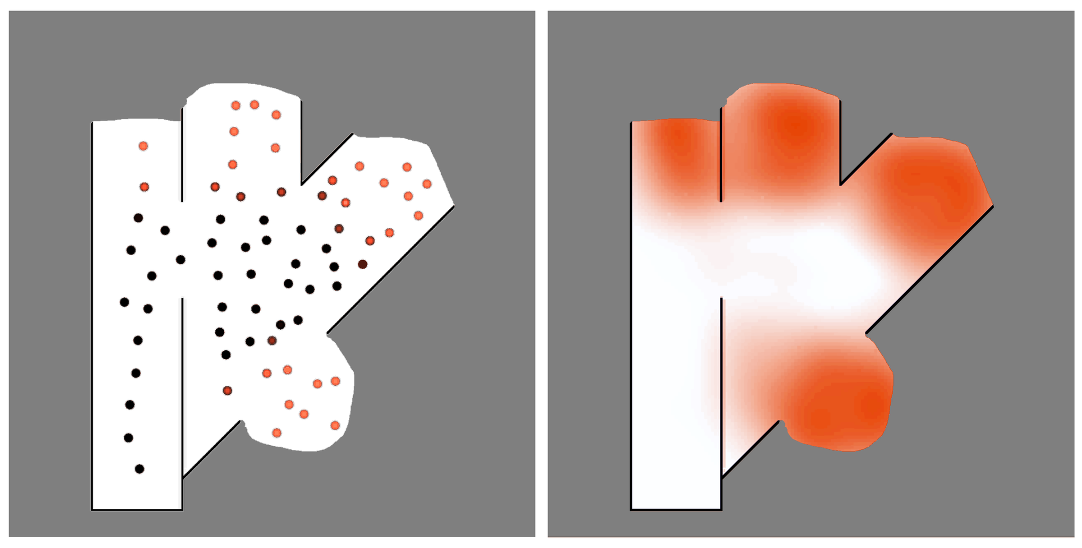
Best yaw calculation

The sample space for new queries is reduced from four dimensions (x,y,z,φ) to three (x,y,z) by sampling only the position and optimizing for the best yaw. This is done by performing ray-casting 360° around the agent and using window summation to find the best yaw.

Gaussian process interpolation of g

We cache calculated information gains from earlier iterations (Fig 5a). These are used as data points in a Gaussian process (Fig. 5b). Whenever a new point $\mathbf{x}$ in g is queried, we first check the result of the Gaussian process. If the posterior variance is low enough, we accept the interpolated data, otherwise we calculate it explicitly and add it to the cache.

Table 1 shows that we save time by querying the Gaussian process instead of calculating the values explicitly. We can also see that the time for explicit gain estimation grows quickly when the rays are denser, while for the GP the time naturally stays the same.



(a) Cached points, black low gain, orange high (b) Gaussian process over the cached points

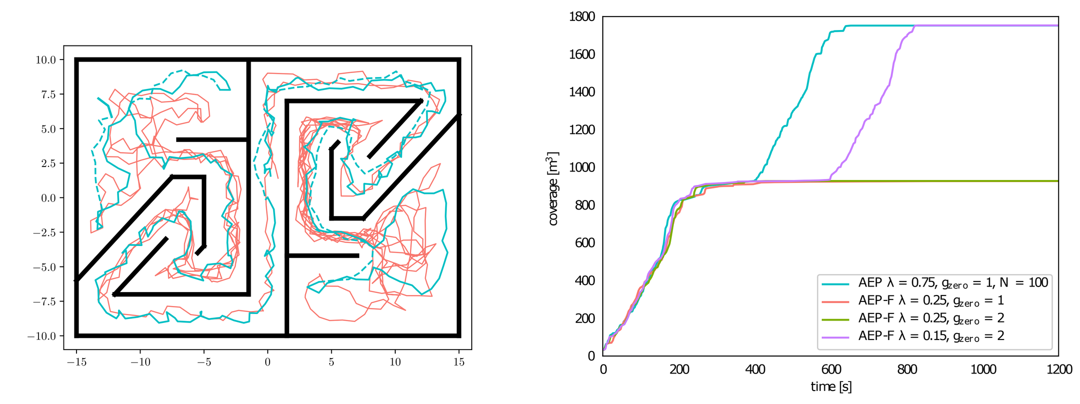
Figure 5

$\Delta\phi$, $\Delta\theta$	[deg]	30	10	5	2
calculation	[ms]	11.3	23.2	63.2	340
interpolation	[ms]	1.52	1.13	1.11	1.13

Table 1: Calculation times for explicit calculation vs. GP interpolation of the potential information gain in one node.

Frontier exploration

When testing RH-NBVP, we saw the behavior that it sometimes got stuck in dead ends. We have tackled this problem by combining RH-NBVP with frontier exploration. RH-NBVP works as a local exploration strategy and frontier exploration takes over when there is no new information to be gained nearby.

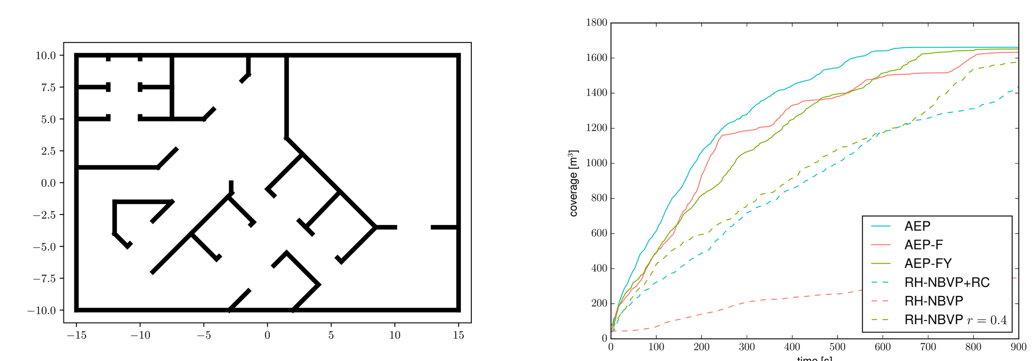


(a) Maze, with two paths drawn. (b) Results for maze scenario

Figure 6: The blue line shows AEP with frontier exploration enabled and the red with frontier exploration disabled.

Figure 7a shows the paths for AEP with frontier exploration enabled (blue) and disabled (red). We can see how both paths start off exploring the right side. When the right side has been completely explored, the blue path switches to frontier exploration and goes out directly, which is indicated by the dashed line. The red path gets stuck and goes back and forth a lot before finally finding its way out. The blue path also terminates directly while the red path goes around for quite some time before terminating.

Results in office environment



(a) Office (b) Results for office scenario

Acknowledgements

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

LINKÖPING UNIVERSITY

KTH VETENSKAP OCH KONST