Available online at www.sciencedirect.com

ScienceDirect

Robotics and Autonomous Systems 00 (2018) 1–18

Journal Logo

Motion Planning under Uncertainty in Graduated Fidelity Lattices

Adrián González-Sieira[a,*], Manuel Mucientes[a], Alberto Bugarín[a]

[a] *Centro Singular de Investigación en Tecnoloxías da Información (CiTIUS), Universidade de Santiago de Compostela, Spain*

Abstract

We present a new approach to motion planning in mobile robotics under sensing and motion uncertainty based on state lattices with graduated fidelity. Uncertainty is predicted at planning time and used to estimate the safety of the paths. Our approach takes into account the real shape of the robot, introducing a deterministic sampling based method to estimate the probability of collision. Anytime Dynamic A*, an informed search algorithm, is used to find safe and optimal paths in the lattice. Moreover, due to the anytime search capabilities of this algorithm our planner is able to retrieve a solution very fast and refine it iteratively until the optimal one is found. We present a novel graduated fidelity approach to build a lattice whose complexity adapts to the obstacles in the environment, along with a multi-resolution heuristic based on the same idea. Thus, the running time of the planner is drastically reduced while maintaining its performance. Experimental results show the potential of the approach in several scenarios, with different robot shapes, motion models and under different uncertainty conditions. The impact of the graduated fidelity approach and the multi-resolution heuristic in the efficiency and performance of the planner is also detailed.

Keywords: state lattices, graduated fidelity, multi-resolution, motion planning under uncertainty

1. Introduction

Representing the state space of the robot in a discrete manner has proven to be a successful approach to reduce the computational complexity of motion planning. Both stochastic and deterministic sampling strategies have been described in the literature [1]. In the state lattices the sampled states are arranged in a regular way, thus giving way to a very efficient representation of the state space. Moreover, state lattices encode a graph whose vertices are the discrete states and the edges connecting them are feasible motions generated in accordance with the dynamics model. Consequently, an informed search algorithm, which relies on heuristics to find the optimal solution faster, can be used to find the path with the minimum cost in it.

The benefits of sampling the state space come at the expense of sacrificing optimality of the planned paths and, more importantly, the feasibility of the planner —its capacity to compute suitable solutions satisfying the constraints. However, asymptotic optimality may be retained under certain conditions [2, 3]. Focusing on state lattices, the fidelity is the resolution of the sampled states. Increasing the fidelity, and thus making the number of samples tend to infinite, would make the state space and the corresponding control set approach the continuum, which makes the state lattices a resolution-complete approach. However, in order to obtain reasonable planning times the fidelity has to be the lowest possible. A state lattice with graduated fidelity varies the resolution of the sampled states in different areas of the state space, which allows the planner to be more efficient while maintaining the feasibility and the optimality of the solutions retrieved. Heuristics also play a significant role in the planning efficiency, and their precision and computation time are equally relevant. To manage the trade-off between them, multi-resolution techniques can be used.

In order to maintain the feasibility of the planner and the optimality of the solutions, both the fidelity of the lattice and the resolution of the heuristic should adapt to the environment. The use of multi-resolution maps, in which the best resolution is selected for each area depending on whether it is cluttered with obstacles or not, allows improving the overall planning efficiency. In this sense, octree based maps [4, 5] are noteworthy for being able to represent large environments with very low computational resources. Moreover, this kind of maps are a valuable source of information for developing multi-resolution

*Corresponding author.

Email addresses: adrian.gonzalez@usc.es (Adrián González-Sieira), manuel.mucientes@usc.es (Manuel Mucientes), alberto.bugarin.diz@usc.es (Alberto Bugarín)

heuristics and graduated fidelity lattices.

In mobile robots, uncertainty may arise from unmodeled external influences on the motion of the robot, or imperfect state information due to noisy or incomplete sensor measurements. The amount of uncertainty varies for each path, since it depends on the executed motions and the quality of the observations about the state the robot is in. In real world domains the safety and accuracy of the planned paths are critical, and therefore the best plan has to be chosen taking into account its associated uncertainty. Managing the uncertainty at planning time allows estimating the safety of each path, such that the best one can be selected accordingly. Moreover, this estimation should take into account the real dimensions of the robot and the inaccuracies of the measurements and the motion model, including the uncertainty in heading. Otherwise, the reliability of the planner might be affected under certain conditions.

In this paper, we present a motion planner based on state lattices that manages the uncertainty at planning time, automatically adapts the fidelity of the lattice and obtains safe and optimal paths. Our proposal combines and extends existing methods in the state of the art, addressing some of their drawbacks and resulting in an efficient motion planning approach. These are the contributions of our proposal:

- A deterministic sampling based method to estimate the probability of collision of each path from its associated uncertainty. This method takes into account the real shape of the robot, also dealing with the uncertainty in heading.
- An approach for obtaining a graduated fidelity lattice which, unlike prior works, adapts to the obstacles in the environment and the maneuverability of the robot.
- A novel multi-resolution heuristic that takes advantage of the resolution of the map to efficiently estimate the cost to the goal.

All the above allows the obtention of safe and optimal solutions in a reliable and efficient way.

2. Related work

Sampling-based techniques in combination with search algorithms have been successfully applied in the field of motion planning. There are approaches based on either random sampling or deterministic sampling. The Probabilistic Roadmaps, PRM [1], and the Rapidly-exploring Random Trees, RRT [6], are noteworthy among the former; while the state lattices described by [7] are the most representative of the latter, standing out for the regular arrangement of the sampled states. PRM and RRT are very efficient exploring the state space, although the regularity of the state lattices makes possible to obtain a finite set of actions from the vehicle motion model. These actions can be computed offline, opening the door to apply techniques to boost the performance of the search, like multi-resolution planning.

State lattices with graduated fidelity, or multi-resolution state lattices, were introduced by [8]. Their approach uses high fidelity only in selected areas, but elsewhere the dynamics model is not taken into account. Thus, the uncertainty cannot be estimated throughout the entire lattice, so it cannot be managed at planning time. In [9] they present a similar approach that uses a subset of the motion primitives to connect those states within the low fidelity areas, addressing this issue. However, [8] and [9] share an important drawback that would make it difficult to combine them with uncertainty management: their approach is based on obtaining a pre-planned path very fast and improve it in real time as the robot moves along it. To achieve this, they place the high fidelity areas around the robot, the start and the goal. However, every time the lattice changes, the uncertainty of all the affected paths must be re-computed and their probabilities of collision updated, which is a non trivial operation.

The motion planner presented in this work relies on a novel graduated fidelity approach. The fidelity of each lattice state is defined in accordance with the obstacles in the the map and the maneuverability of the robot. Therefore, our approach increases the fidelity only in those areas which are challenging for planning. By doing so, the trade-off between the computational complexity and the quality of the solution is managed, and the computation time is significantly reduced. Moreover, our method does not require updating the solution unless the map changes, since the fidelity does not depend on the position of the robot. Thus, the drawbacks of prior approaches in the literature are addressed.

The use of heuristics significantly improves the efficiency of state lattice based motion planners, especially in high dimensional domains. In this sense, [10] described an admissible heuristic which copes with the robot kinematic constraints assuming free space. This is a good informed heuristic in uncluttered environments, and it can be computed offline and stored in a look-up table. In [9] they presented a low dimensional heuristic which copes with the obstacles in the environment. This heuristic, in combination with that of [10], obtained very good results. However, the approach of [9] is based on applying Dijkstra's algorithm over a grid with a fixed resolution which matches the maximum fidelity of the lattice. Computing this heuristic can be costly and affect the planning efficiency, specially in large, uncluttered environments.

To address this, in this paper we present a heuristic based on that of [9] which, instead of a fixed resolution, uses the information of the map to build a multi-resolution grid. This allows improving the efficiency and scalability of the heuristic in large environments.

While classical planners do not take into account motion uncertainty, which originates in noisy controls and

measurements, approaches which manage it at planning time have received increasing attention in the last years. Some of them only consider uncertainty in control, like [11], which combines PRM and the theory of Markov Decision Processes, MDP, to maximize the probability of success of the given paths. This method can be extended and consider Partially Observable MDPs, POMPDs, to also manage sensing uncertainty, as in [12]. However, this approach has scalability issues which can only be addressed with approximate sample-based solutions, as described by [13]. In spite of these efforts, discretizing high-dimensional continuous dynamics to be used with POMPDs did not get promising results [14]. The work developed by [15] overcame this drawback, but further research is needed to extend it to non smooth dynamics and observations.

Current state of the art for motion planning with uncertainty is the algorithm proposed by [16] and the similar approach of [17], as both consider motion and sensing uncertainty and do not assume maximum likelihood measurements. The former, LQG-MP, relies on RRT to find the path that minimizes the probability of collision, but the obtained paths might be non smooth. This issue is addressed applying smoothing techniques over the planned path, which might affect the predicted uncertainty and therefore the probability of collision in execution time. Although results for LQG-MP combined with a search algorithm are outlined, they were obtained from simple roadmaps made by hand. Moreover, in [2] they demonstrate that RRT is not asymptotically optimal. As a consequence, there is no guarantee that the optimal path will be found even for a high number of samples, either in terms of cost or probability of collision. The latter presents a similar approach that makes use of RRT*, an extension of RRT which addresses this issue with an increased connectivity. This work achieves good results in the uncertainty prediction. However, a significant number of iterations of RRT* is required to find a near-optimal solution. While this method obtains paths significantly more smooth than RRT, like in [18], this issue is not completely overcome, especially when the number of samples is low.

In this work, motion uncertainty is managed with the method of [17]. However, in this proposal it is combined with a state lattice, addressing those issues related with the smoothness of the paths which arise from the use of random sampling-based methods.

Despite the good results obtained by prior works in predicting motion uncertainty, the probability of collision is estimated assuming simplified versions of the robot shape, such as circles [16, 19]. While this allows the probability of collision to be estimated faster, the influence of the uncertainty in heading is not taken into account. Others rely on chance-constrained search [17, 18], only checking collisions between the Probability Density Functions —PDFs— and the obstacles, therefore considering punctual robots. Since these approaches disregard the real shape of the robot, there is no guarantee that the provided paths will be safe under all circumstances. In fact, their reliability falls when the uncertainty in heading is clearly significant —i.e. when the shape is long or asymmetric.

To solve this drawback, this proposal introduces a novel method to accurately estimate the probability of collision, based on sampling the PDFs. This method uses the real shape and deals with the uncertainty in heading. Thus, reliable collision free paths for all kinds of robot shapes are obtained.

3. Planning on state lattices

3.1. Motion primitives

The motion planner presented in this work relies on a state lattice to sample the state space, $\mathcal{X}$, in a deterministic and regular manner. In this work a rectangular arrangement of the samples was chosen, although other configurations are possible. The states belonging to the lattice, $\mathcal{X}_{lat}$, are connected by a set of actions —$\mathcal{U}$, also called motion primitives— extracted from the dynamics model. Due to the regular arrangement of the sampled states, these actions can be computed offline and efficiently stored. As they are position-independent, the same motion primitive connects every pair of states equally arranged. Fig. 1 illustrates the regularity of the states belonging to the lattice and the connectivity obtained via replication of the set of actions $\mathcal{U}$.

It is straightforward that using the motion primitives to connect the lattice states ensures that, by construction, this structure is generated in accordance with the robot dynamics. Since the kinematic restrictions are observed, all paths contained in it are feasible.

In this work, the control set was obtained applying a numerical optimization technique based on the Newton-Raphson method, introduced by [20]. The resulting ac-

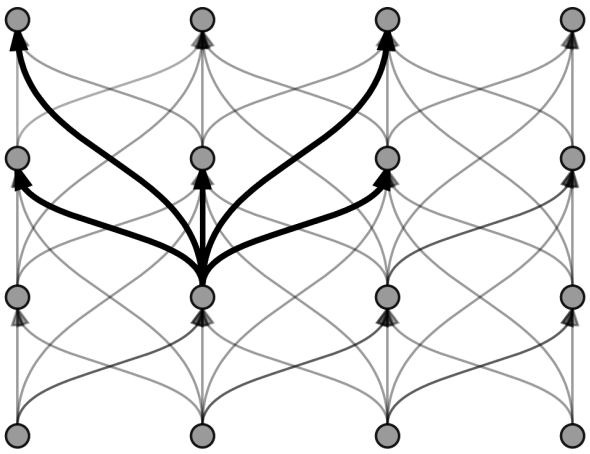


Figure 1: Motion planning based on state lattices. Due to the regular sampling, the motion primitives can be obtained offline and replicated to connect the sampled states. Then, an informed search algorithm can be used to find optimal paths in the lattice.

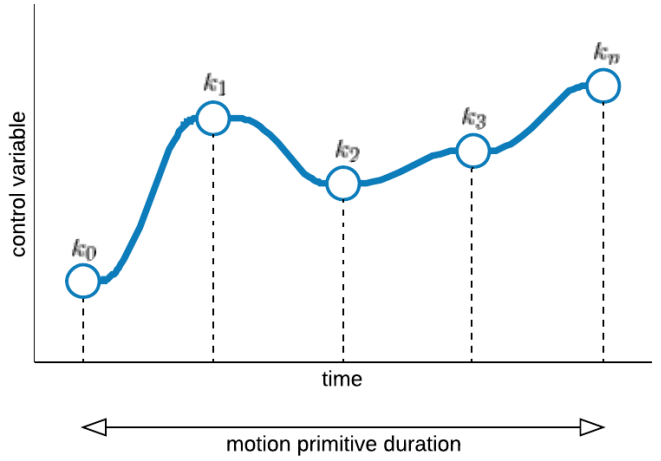


Figure 2: Parametrization of the control function via cubic spline interpolation. k_0 and k_n are given by the initial and final states connected by the motion primitive, while k_i are the knot points defining the rest of the function.

tions are optimal in terms of cost, given the constraints: the initial an final states —which belong to $\mathcal{X}_{lat}$— and the dynamics model.

The motion primitives have been parametrized via cubic spline interpolation [21] to represent the evolution of the controls over time. The parameters are the *knot points* of the splines, a set of equally spaced values —$k_1, k_2, ..., k_n$— from which the rest of the function is interpolated, as shown in Fig. 2. Each control variable is represented by a different spline, which might be given by a different number of knots. Hence, the parameter vector for a motion primitive is defined as follows:

$$p = [(k_0^1, k_1^1, ..., k_{n_1}^1), (k_0^2, k_1^2, ..., k_{n_2}^2), ..., t] \quad (1)$$

where k_i^j is the i-th knot belonging to the spline of the j-th control variable, and t is the duration of the motion primitive.

Finally, the parameter vector p is optimized with respect to an error function, $e(p)$, which measures the difference between the desired final state and the one given by the parameters —obtained from the dynamics model. The parameter vector is modified following:

$$\Delta p = -\left[\frac{\partial e(p)}{\partial p}\right]^{-1} \cdot e(p) \quad (2)$$

This is repeated until the constraints are satisfied, typically when $e(p)$ is under a threshold.

The dynamics model is learned from motion data of the robot, as described in [22]. The approach relies on parametrizing the equations for all linear and angular velocities in this manner:

$$v_{t+1} = \beta_1^v \cdot v_t + \beta_2^v \cdot u_t^v + \beta_0^v \quad (3)$$

$$\omega_{t+1} = \beta_1^\omega \cdot \omega_t + \beta_2^\omega \cdot u_t^\omega + \beta_0^\omega \quad (4)$$

making use of an iterative least-squares method to obtain the parameters, β_i^v and β_i^ω, which best fit the input data. u_t^v and u_t^ω are the linear and angular controls, respectively. The motion primitives obtained with this model are accurate representatives of the robot maneuvering capabilities, since they encode its real response to the different controls.

3.2. Optimal path

As the state lattice has the structure of a graph, an informed search algorithm can be used to find the optimal path in it. This proposal uses Anytime Dynamic A* —AD* [23]— because of its capability to obtain sub-optimal bounded solutions varying an heuristic inflation parameter, ϵ. The solutions can be iteratively refined taking advantage of the information previously calculated, without need for replanning from scratch.

Alg. 1 outlines the main operations of AD* —see [23] for the detailed pseudocode of AD*. The inputs are the initial state and the goal, x^0 and x^G, and the output is the path with minimal cost connecting them. In each iteration a state x^a is extracted from *OPEN* —Alg. 1:9. This state is the one which minimizes the sum of the cost from the start, c_x, and the estimated cost to the goal —h_x, given by the heuristic—, scaled by ϵ. Next, the successors of x^a are retrieved —X^b, in Alg. 1:10— and the evaluation of the outgoing actions is done in Alg. 1:12. Finally, in Alg. 1:13-16, the cost of x^b is updated, its heuristic obtained and it is inserted into the *OPEN* queue to be explored in following iterations —only if x^b is visited for the first

Algorithm 1 Main operations of the search algorithm

Require: x^0, initial state
Require: x^G, goal state
Require: ϵ_0, initial value of ϵ
1: **function** MAIN (x^0, x^G, ϵ_0)
2: INITIALIZEHEURISTIC(x^0, x^G) ▷ Alg. 5
3: $\epsilon = \epsilon_0$
4: **while** $\epsilon >= 1$ **do**
5: $c_{x^0} = 0$
6: h_{x^0} = HEURISTIC(x^0)
7: $OPEN = \{x^0\}$
8: **repeat**
9: $x^a = \arg\min_{x \in OPEN}(c_x + \epsilon \cdot h_x)$
10: X^b = SUCCESSORS(x^a) ▷ Alg. 4
11: **for all** $x^b \in X^b$ **do**
12: $\hat{c}_{x^b} = c_{x^a}$ + COST(x^a, x^b) ▷ Alg. 3
13: **if** x_b not visited or $\hat{c}_{x^b} < c_{x^b}$ **then**
14: $c_{x^b} = \hat{c}_{x^b}$
15: h_{x^b} = HEURISTIC(x^b) ▷ Alg. 5
16: $OPEN = OPEN \cup \{x^b\}$
17: $OPEN = OPEN - \{x^a\}$
18: **until** $x^a = x^G$
19: publish $path(x^0, x^a)$
20: decrease ϵ
21: **return**

time or the current path improves an existing one. The algorithm finds a valid path when the state extracted from OPEN, x^a, is the goal x^G.

The planning algorithm relies on the use of heuristics to efficiently explore the state space and obtain an optimal solution in fewer iterations. The heuristic function provides an estimation of the cost between each state x^a and the goal x^G, which influences the order in which the states are processed and therefore the number of iterations needed to find the optimal path.

AD* introduces a parameter, ϵ, which inflates the values of the heuristic. This allows obtaining sub-optimal bounded solutions faster than the optimal one. Thus, the algorithm is run in an iterative way, obtaining an initial solution for $\epsilon = \epsilon_0$. This solution is refined in subsequent executions, after decreasing the value of ϵ, taking advantage of the information previously calculated —Alg. 1:19-20. This is less computationally expensive than obtaining a new solution from scratch every time ϵ changes.

The heuristic function is a combination of two values —proposed by [9] and [24]— which allows using both the information of the obstacles in the map and the dynamics model: one is the cost of the path considering only the kinematic restrictions, FSH, while the other is the cost of the path only taking into account the information of the map, H2D.

As it takes into account the kinematic restrictions, obtaining FSH is a costly operation. Therefore, it is computed offline and stored in an Heuristic Look-Up-Table, as described in [10]. The process starts with a first step in which Dijkstra's algorithm is applied to populate the table in a rapid way, followed by another step in which the most complex maneuvers are included. This heuristic takes advantage of the regularity of the lattice and the symmetries in the control set to improve the efficiency of its calculation and storage.

On the contrary, H2D has to be initialized every time the planner is run —Alg. 1:2—, since it depends on the location of the goal and the obstacles in the environment. Therefore, the lower the obtention time of this heuristic, the higher the overall efficiency of the planner.

3.3. Uncertainty management

Uncertainty management requires to predict the probability of the robot being in each state of the path. This uncertainty depends on the one at the initial state, the executed controls and the location accuracy, and therefore it varies along the different candidate paths in the lattice. The prediction of these probability distributions is integrated in planning time.

This proposal focuses on nonlinear, partially observable systems. Dynamics —f— and observations —z— are described in a discrete time manner:

$$x_{t+1} = f(x_t, u_t) + m_t, \qquad m_t \sim \mathcal{N}(0, M_t) \tag{5}$$

$$z_t = z(x_t) + n_t, \qquad n_t \sim \mathcal{N}(0, N_t), \tag{6}$$

Algorithm 2 Uncertainty propagation along a trajectory between x^a and x^b

Require: x^a and x^b, beginning and final states

1: **function** UNCERTAINTY(x^a, x^b)
2: $u^{a:b} = cmd(x^a, x^b)$
3: $\bar{x}_{t-1} = x^a$
4: $\Sigma_{t-1} = \Sigma_{x^a}$
5: $P^{a:b} = \emptyset$
6: **for all** $u_t^{a:b} \in u^{a:b}$ **do**
7: $\bar{x}_t = f(\bar{x}_{t-1}, u_t^{a:b})$
8: $\bar{\Sigma}_t = A_t \Sigma_{t-1} A_t^T + M_t$
9: $K_t = \bar{\Sigma}_t H_t^T (H_t \bar{\Sigma}_t H_t^T + N_t)^{-1}$
10: $\widetilde{\Sigma}_t = (I - K_t H_t) \bar{\Sigma}_t$
11: $C_t = A_t + B_t L_t$
12: $\Lambda_t = C_t \Lambda_{t-1} C_t^T + K_t H_t \bar{\Sigma}_t$
13: $\Sigma_t = \widetilde{\Sigma}_t + \Lambda_t$
14: $P^{a:b} = P^{a:b} \cup \{x_t \sim \mathcal{N}(\bar{x}_t, \Sigma_t)\}$
15: $\bar{x}_{t-1} = \bar{x}_t$
16: **return** $P^{a:b}$ ▷ PDFs of the path

where $x_t \in \mathcal{X}$ are the states of the robot, $u_t \in \mathcal{U}$ are the controls and z_t are the measurements. m_t and n_t are random motion and observation disturbances, which are described by Gaussian distributions. M_t and N_t are their respective covariances.

Obtaining the cost for a path between two states x^a and x^b is a two step process. First the uncertainty is propagated along the trajectory, and then the resulting probability distributions —PDFs— are used to estimate the probability that the robot collides when executing the path.

The former is done following the approach of [17], which has good results for the kind of systems described above. It is an EKF-based method and it manages the uncertainty which arises from the controls and the observations. Moreover, it also takes into account the influence of using a Linear Quadratic Gaussian controller —LQG, a widely extended controller to correct deviations from the planned path in execution time, detailed in [25]. For approximating the probability of collision along the paths we introduce a novel method in Sec. 4.1 which takes into account the real shape of the robot and provides a reliable estimation.

Uncertainty prediction is detailed in Alg. 2. Inputs are the beginning and final states of the trajectory —x^a and x^b—, while the output is the list of PDFs along the path between them —$P^{a:b}$, which is obtained iteratively propagating the uncertainty at $x^a \sim \mathcal{N}(\bar{x}^a, \Sigma^a)$. $u^{a:b}$ are the control commands of the trajectory —Alg. 2:2. L_t is the gain of a LQG controller, which is taken into account due to its influence on the PDFs. H_t is the Jacobian of the measurement model and A_t and B_t are the Jacobians of the dynamics model.

Motion uncertainty is predicted as follows: first, an EKF is used to calculate the distributions of the state in the prediction step —$\mathcal{N}(\bar{x}_t, \bar{\Sigma}_t)$, in Alg. 2:7-8— and the

true one after the update —$\tilde{x}_t \sim \mathcal{N}(\bar{x}_t, \widetilde{\Sigma}_t)$, Alg. 2:9-10. This distribution can be seen as $P(x_t|\tilde{x}_t)$, which represents the probability of being in x_t if the EKF predicts so. After, $P(\tilde{x}_t) = \mathcal{N}(\bar{x}_t, \Lambda_t)$ is obtained in Alg. 2:11-12. This models the uncertainty due to obtaining the state estimation without having taken the real observations. These two distributions are used to calculate $P(x_t, \tilde{x}_t) = P(x_t|\tilde{x}_t)P(\tilde{x}_t)$, the joint one of the real robot state and the true state given by the EKF. With all of the above we can finally get the PDF of the real state, $P(x_t) = \mathcal{N}(\bar{x}_t, \widetilde{\Sigma}_t + \Lambda_t)$, in Alg. 2:13, which the motion planner uses as:

$$x_t \sim \mathcal{N}(\bar{x}_t, \Sigma_t) \tag{7}$$

This distribution is the one the planner uses to estimate the probability of collision along the paths.

4. Improving the reliability and efficiency of the motion planner

In this section the contributions of this work are detailed. First, we present a method to estimate the probability of collision of each path from its associated uncertainty which takes into account the real shape of the robot and the uncertainty in heading. Then, we propose a novel graduated fidelity approach which goes in accordance with the obstacles in the environment and the maneuverability of the robot. Finally we present H2DMR, an heuristic based on the idea of H2D which makes use of multi-resolution techniques to improve its scalability and efficiency.

4.1. Reliable probability of collision

The goal of the planner is to obtain paths minimizing the probability of collision and the traversal time. To achieve this, each candidate path between two states, x^a and x^b, is evaluated to obtain a cost comprised by three elements: a safety measurement —$c^{a:b}$, proportional to the probability of collision along it—, the traversal time $t^{a:b}$, and the uncertainty at the final state Σ^b.

Algorithm 3 Cost of a trajectory between x^a and x^b

Require: x^a and x^b, beginning and final states
1: **function** COST(x^a, x^b)
2: $P^{a:b}$ = UNCERTAINTY(x^a, x^b) ▷ Alg. 2
3: $t^{a:b} = time(u^{a:b})$
4: $c^{a:b} = 0$
5: **for all** $x_t^{a:b} \sim \mathcal{N}(\bar{x}_t^{a:b}, \Sigma_t^{a:b}) \in P^{a:b}$ **do**
6: $w_c = 0$
7: $w_t = 0$
8: $X^S = sampling(x_t^{a:b})$
9: **for all** $x^s \in X^S$ **do**
10: $w = pdf(x^s, x_t^{a:b})$
11: $w_t = w_t + w$
12: **if** $collision(x^s)$ **then** ▷ with real shape
13: $w_c = w_c + w$
14: $p_c = w_c / w_t$
15: $c^{a:b} = c^{a:b} - \log(1 - p_c)$
16: **return** $\left[c^{a:b}, t^{a:b}, \Sigma^b\right]$ ▷ Σ^b, uncertainty at x^b

Alg. 3 details how this evaluation is done. As mentioned before, this requires first obtaining the PDFs in Alg. 3:2, since they are needed to calculate $c^{a:b}$ and to know the final uncertainty Σ^b. On the contrary, the traversal time is directly obtained from the motion primitives —$t^{a:b}$, in Alg. 3:3.

The probability of collision is estimated from the PDFs by obtaining a set of samples X^S and checking collisions with the obstacles in the environment taking into account the real shape of the robot, as detailed in Alg. 3:8-13. The sampling strategy is based on the method of the Unscented Kalman Filter —UKF [26]— to obtain the sigma points of a PDF, which represents the distribution reasonably well with a small number of samples. These sigma points are distributed in all dimensions, making them suitable for checking collisions also dealing with the uncertainty in heading. Moreover, their obtention only depends on the parameters of the PDF, retaining the deterministic nature of the motion planner.

A n-dimensional distribution belonging to the path, $x_t \sim (\bar{x}_t, \Sigma_t)$, is sampled as follows:

$$\begin{aligned} x^s_{[i^+]} &= \bar{x}_t + \lambda \cdot \left(\sqrt{\Sigma_t}\right)_i \qquad \text{for } i = 1, ..., n \\ x^s_{[i^-]} &= \bar{x}_t - \lambda \cdot \left(\sqrt{\Sigma_t}\right)_i \end{aligned} \tag{8}$$

The scaling factor λ allows obtaining samples with different distance to the mean —i.e. $\lambda = 3$ will generate samples with a distance of 3 standard deviations from the most probable state. Two samples —$x^s_{[i^+]}$ and $x^s_{[i^-]}$— are obtained for each dimension of the distribution, adding and subtracting each column of the factorized matrix —$\left(\sqrt{\Sigma_t}\right)_i$— to the mean of the PDF, $\bar{x}_t$. This generates $2 \cdot n + 1$ samples for each value of λ, the sigma points. While these are good representatives of the distribution, their coverage for checking collisions with the obstacles in the environment may be not enough. For example, samples with different headings would be generated only in the position of $\bar{x}_t$. For this reason, our method also operates with different columns of the covariance matrix to obtain samples not only in the main axes of the distribution, but also in the diagonals, which results in an improved coverage of the PDF. Thus, for each sigma point from Eq. 8 —$x^s_{[i^-]}$ and $x^s_{[i^+]}$— the following samples are obtained:

$$\begin{aligned} x^s_{[i^-,\ j^-]} &= x^s_{[i^-]} - \lambda \cdot \left(\sqrt{\Sigma_t}\right)_j \qquad \text{for } i, j = 1, ..., n \\ x^s_{[i^-,\ j^+]} &= x^s_{[i^-]} + \lambda \cdot \left(\sqrt{\Sigma_t}\right)_j \qquad j \neq i \\ x^s_{[i^+,\ j^-]} &= x^s_{[i^+]} - \lambda \cdot \left(\sqrt{\Sigma_t}\right)_j \\ x^s_{[i^+,\ j^+]} &= x^s_{[i^+]} + \lambda \cdot \left(\sqrt{\Sigma_t}\right)_j \end{aligned} \tag{9}$$

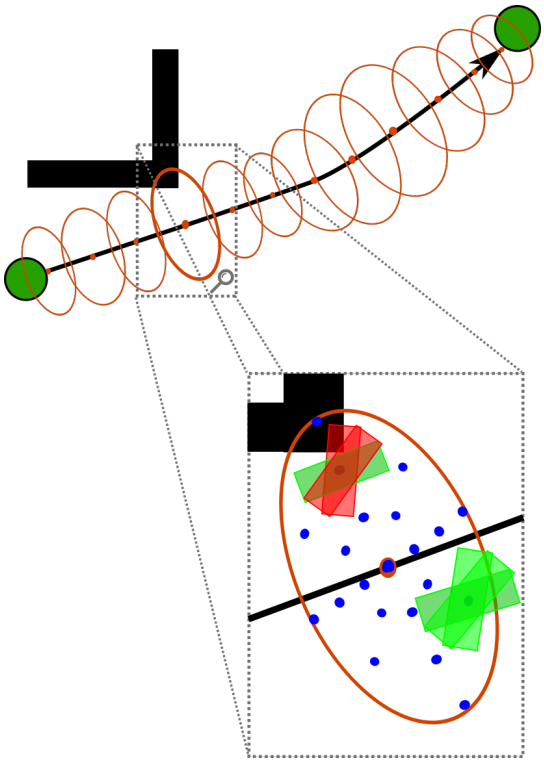


Figure 3: Estimation of the probability of collision taking into account the real shape of the robot, via sampling the PDF using the strategy of an UKF.

Fig. 3 shows the samples obtained applying this method for $\lambda = 1, 2$ and 3. These samples are used to check collisions with the surrounding obstacles, for which the real shape of the robot is taken into account. Each sample x^s has a weight, w, in accordance with its probability —function *pdf*, in Alg. 3:10. w_t is the sum of weights of all samples, while w_c is the sum of those in which the real shape collides —Alg. 3:11 and Alg. 3:13, respectively. The quotient between them is the probability of collision of the distribution, p_c, in Alg. 3:14. Finally, the individual estimations are combined in a logarithmic scale to obtain a cost related to the probability of collision of the entire path, as detailed in Alg. 3:15. By doing so we assume that the probabilities of collision in different stages of the path are independent, in the same way that most approaches in the state of the art. Although this is not the case, it is a reasonable assumption for practical purposes.

Each element returned by the cost function —Alg. 3:16— represents an objective to be minimized by the motion planner, and therefore an order of priority was introduced when comparing the cost of two paths, which is done as follows:

$$\begin{aligned} cost(x^{a:b}) < cost(x^{a:c}) \Leftrightarrow (c^{a:b} < c^{a:c}) \vee \\ (c^{a:b} = c^{a:c} \wedge t^{a:b} < t^{a:c}) \vee \\ (c^{a:b} = c^{a:c} \wedge t^{a:b} = t^{a:c} \wedge \Sigma^b < \Sigma^c) \end{aligned} \quad (10)$$

Thus, the motion planner first minimizes the cost related to the probability of collision, prioritizing the safety of the planned paths. Among all the safe paths, it selects the one minimizing the traversal time and, finally, the uncertainty at the goal. Therefore, the planned paths are safe and optimal. Moreover, since the probability of collision is estimated taking into account the real dimensions of the robot, it is reliable for all kinds of shapes.

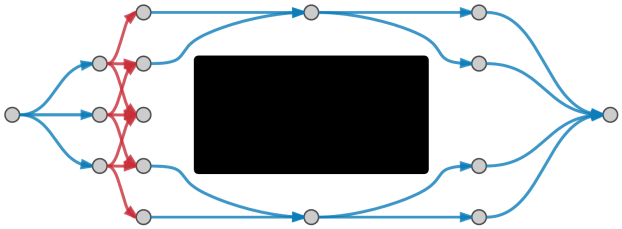


Figure 4: Example of graduated fidelity lattice. Trajectories in red are those with highest fidelity, required to maneuver near the obstacle —in black. The rest of trajectories have lower fidelities —in blue—, since the obstacles do not affect the maneuverability.

4.2. Graduated fidelity lattice

The efficiency of the planner strongly depends on the fidelity of the state lattice. While a higher fidelity allows more precision in representing the state space and the maneuvering of the robot, decreasing it significantly diminishes the runtime of the search. Although the latter may result in paths with higher costs, the use of a graduated fidelity lattice can balance precision and efficiency adequately. Moreover, if the fidelity adapts to the obstacles in the environment and the maneuverability of the robot, the efficiency can be improved with minimal impact in the planning results, as shown in Sec. 5.

Figure 4 depicts a state lattice with graduated fidelity, in which only those areas which require complex maneuvering to avoid obstacles in the environment are represented with high fidelity. The proposed approach is to select, whenever possible, the longest maneuver of each type to move between states. To achieve this, those motion primitives in $\mathcal{U}$ which are similar maneuvers of different lengths are grouped. Then, the longest primitive of each group which does not affect the probability of collision is selected, and the rest discarded. This allows to diminish the number of states belonging to the lattice, and at the same time the number of candidate paths connecting them, therefore simplifying the state space. This technique is applied when generating the successors of the state x^a explored by the search algorithm, in Alg. 1:10.

Alg. 4 details how the outgoing trajectories of a state x^a are selected using the proposed graduated fidelity technique. As mentioned before, this procedure first requires grouping the motion primitives from $\mathcal{U}$ in maneuvers of the same kind but different length. Those trajectories with the same values for orientations, linear and angular speeds both at the beginning —θ_i, v_i, ω_i— and at the end —

θ_f, v_f, ω_f— belong to the same group —$\mathcal{U}_{(\theta_i,v_i,\omega_i,\theta_f,v_f,\omega_f)}$. The union of all groups is the whole set of primitives —$\mathcal{U} = \bigcup \mathcal{U}_{(\theta_i,v_i,\omega_i,\theta_f,v_f,\omega_f)}, \forall(\theta, v, \omega) \in \mathcal{X}_{lat}$—, whereas the intersection of any two of them is empty.

The successors of a state, Γ, are generated as follows: First, those groups of trajectories with the same initial speeds —v_i and ω_i— and orientations —θ_i— as x^a are selected, in Alg. 4:4. Then, the algorithm chooses a primitive of each group to be part of the successors and discards the rest. The candidates, U^c, are evaluated in descending order by length until one that fulfills the restrictions is found, as Alg. 4:5-8 details. If none of the trajectories in the group fulfills the restrictions, the shortest one is selected.

As regards to the evaluation of candidates, two conditions must be fulfilled to select them: the resolution of both the source and destination octree cells, s^a and s^b, and the probability of collision of the candidate trajectory U^c. The former is related to the structure of the octree: in the vicinity of the obstacles the cells contain different occupancy information and cannot be compacted in higher level cells, so the higher the cluttering the lower the size of the cells in the map. Thus, limiting the length of the maneuvers in accordance with the size of the cells —Alg. 4:12-13— results in a lattice with high fidelity only in those areas challenging for the planner. The second condition is introduced to maintain the safety of the solutions. Those maneuvers which affect the probability of collision —$c^{a:b}$, in Alg. 4:14— are discarded. This is obtained from the cost of the trajectory, as detailed in Alg. 3.

Fig. 5 illustrates how this approach discards the longest maneuvers in the vicinity of obstacles due to their probability of collision, while navigating in uncluttered areas causes the acceptance of the first explored candidates. This graduated fidelity approach results in a lattice with a considerably lower density of states and maneuvers except in those areas in which complex maneuvering is required for obstacle avoidance. Consequently, the efficiency of the planner is considerably improved, while its performance —the cost of solutions— is barely affected.

Algorithm 4 Successor generation for a graduated fidelity state lattice

Require: $\mathcal{U} = \{\mathcal{U}_{(\theta_i,v_i,\omega_i,\theta_f,v_f,\omega_f)}\}, \forall(\theta, v, \omega) \in \mathcal{X}_{lat}$
1: **function** SUCCESSORS(x^a)
2: $\theta_i = x^a_\theta$; $v_i = x^a_v$; $\omega_i = x^a_\omega$
3: $\Gamma = \emptyset$
4: **for** $U \in \{\mathcal{U}_{(\theta_i,v_i,\omega_i,\theta_f,v_f,\omega_f)}\}, \forall(\theta_f, v_f, \omega_f)$ **do**
5: **repeat**
6: $U^c = \arg\max_{t^{a:b}}(U)$ ▷ Get longest
7: $U = U \setminus U^c$
8: **until** CHECK($U^c_{x^a}, U^c_{x^b}$) $\vee$ $U == \emptyset$
9: $\Gamma = \Gamma \cup U^c$
10: **return** Γ
11: **function** CHECK(x^a, x^b)
12: $\kappa^a = cell(x^a)$; $\kappa^b = cell(x^b)$ ▷ Get map cells
13: $s^a = size(\kappa^a)$; $s^b = size(\kappa^b)$ ▷ Get size of cells
14: $c^{a:b} = \text{COST}(x^a, x^b)[0]$ ▷ Alg. 3
15: **return** $(s^a + s^b \geqslant \|x^a - x^b\|) \wedge (c^{a:b} == 0)$

4.3. Multi-resolution heuristic

As mentioned in Sec. 3, our approach combines two heuristics: one takes into account the kinematic restrictions in free space, FSH, while the other only considers the obstacles in the environment, H2D. In this section H2DMR, a multi-resolution heuristic based on the latter, is proposed. Unlike H2D, this novel heuristic takes advantage of the octree structure of the map to obtain a multi-resolution grid, resulting in improved efficiency and scalability.

Alg. 5 details how H2DMR is calculated. The process is done applying Dijkstra's algorithm to obtain a multi-resolution grid. The inputs are the initial state of the robot and the goal, x^0 and x^G. Since the heuristic uses positions instead of states, their counterparts —p^0 and p^G— are obtained in Alg. 5:4-5. The grid is generated backwards, starting in p^G. Thus, the resulting grid will contain the estimated cost between each point and the goal, which is used as the heuristic value for planning.

Iteratively, the point with the lowest cost from the start —p, in Alg. 5:9— is selected and its successors obtained. κ is the cell of the map containing the selected point —Alg. 5:10. To generate its successors in accordance with the

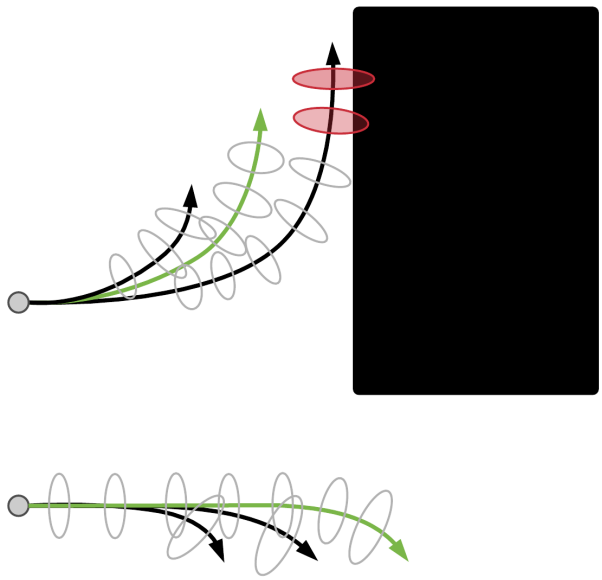


Figure 5: Example of the selected candidates with the graduated fidelity approach in different situations —in green. Long trajectories are discarded when they affect the probability of collision with obstacles —in black—, while in free space they are selected to have a lower fidelity.

resolution of the map, the adjacent cells of κ —$adjacent(\kappa)$, in Alg. 5:12— are explored.

Figure 6 details how the resolution of the octree is taken into account when obtaining the neighbors of p. Given the size of a cell —s, in Alg. 5:13— and the highest fidelity of the motion primitives, f^+, two situations are considered: on the one hand, a cell is split into subcells when f^+ exceeds the resolution of the map, in order to keep the accuracy of the heuristic —Alg. 5:16-20. On the other hand an upper bound in the resolution was introduced to avoid generating neighbors with a distance lower than f^+ —Alg. 5:23-27. Therefore, the size of the cells this heuristic works with is in fact limited according to f^+ —Alg. 5:23. In both cases the resulting neighbors are obtained from the center of the selected cells —$position(\kappa'')$, in Alg. 5:18 and Alg. 5:25.

Finally, the cost between p and each neighbor p' —the distance between them— is obtained and p' is introduced into the *OPEN* queue to be explored by the algorithm later. Collisions are checked using the inscribed circle in the robot shape, also called optimistic shape, in Alg. 5:30. Thus, the optimistic nature of the heuristic is maintained.

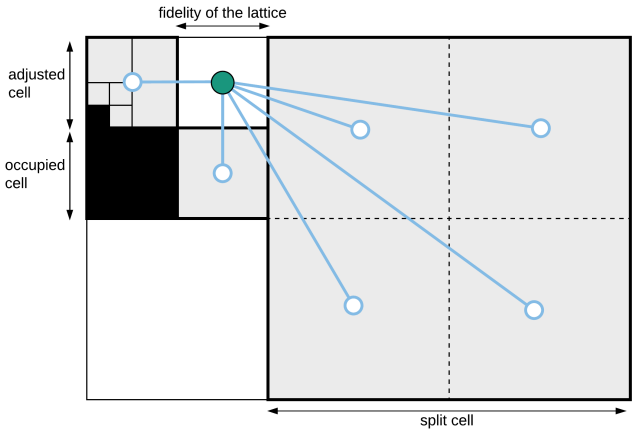


Figure 6: Neighborhood of H2DMR —in blue. Given a point —in green—, those cells adjacent to the one containing it are explored —in gray. When a cell is smaller than the highest fidelity of the lattice —f^+—, a bigger one containing it is selected. On the contrary, a cell is split into subcells when its size exceeds f^+.

Algorithm 5 Obtention of H2DMR

Require: f^+, highest fidelity of the lattice

1: **function** HEURISTIC(x)
2: **return** max $\left(h2dmr(x^b), fsh(x^b)\right)$
3: **function** INITIALIZEHEURISTIC(x^0, x^G)
4: $p^0 = position(x^0)$ ▷ Initial position
5: $p^G = position(x^G)$ ▷ Goal position
6: $c(p^G) = 0$
7: $OPEN = \{p^G\}$
8: **repeat**
9: $p = \arg\min_{p \in OPEN} c(p)$
10: $\kappa = cell(p)$ ▷ Get map cell containing p
11: /* Iterate over those cells adjacent to κ */
12: **for all** $\kappa' \in adjacent(\kappa)$ **do**
13: $s = size(\kappa')$ ▷ Size of cell κ'
14: **if** $s > f^+$ **then**
15: /* Split κ' into subcells */
16: **for all** $\kappa'' \in subcells(\kappa')$ **do**
17: /* Get center of cell κ'' */
18: $p' = position(\kappa'')$
19: $c(p') = c(p) +$ COSTH(p, p')
20: $OPEN = OPEN \cup \{p'\}$
21: **else**
22: /* Adjust size of κ' to f^+ */
23: $\kappa'' = adjust(\kappa', f^+)$
24: /* Get center of cell κ'' */
25: $p' = position(\kappa'')$
26: $c(p') = c(p) +$ COSTH(p, p')
27: $OPEN = OPEN \cup \{p'\}$
28: **until** $c(p) > 2 \cdot c(p^0)$
29: **function** COSTH (p, p')
30: **if** $collision_0(p')$ **then** ▷ Optimistic shape
31: **return** ∞
32: **else**
33: **return** $\|p' - p\|$

The stopping condition of the algorithm is to expand a point with a cost which doubles the one between p^0 and p^G —Alg. 5:28. Those areas of the map left outside the generated grid are not interesting for planning due to their distance to the most promising path. H2D uses this same stopping condition, which was introduced by [9] for efficiency purposes.

This algorithm allows the obtention of a multi-resolution grid which contains the cost between each point and the goal, which is used by AD* as heuristic. Unlike H2D, this grid takes into account both the resolution of the octree map and the highest fidelity of the motion primitives. Thus, this approach outperforms H2D in the number of iterations required to explore the map, and consequently the time spent in initializing the heuristic. This is specially noticeable in large environments, since H2DMR scales better than H2D due to its capability to use lower resolutions in uncluttered areas.

Since the positions of the multi-resolution grid and the states in the lattice do not directly match, obtaining the heuristic value for a state x^a is done as follows: first, the octree cell containing it is retrieved, and then all positions of the grid within this cell and the adjacent ones are explored. The heuristic of the state is given by the position —p— which minimizes the sum of its cost —$c(p)$— and the distance to x^a:

$$h2dmr(x^a) = \arg\min_p(\|x^a - p\| + c(p)) \qquad (11)$$

Heuristics play a significant role in the anytime search capabilities of AD*, since their value is scaled by the parameter ϵ. By doing so, a sub-optimal solution can be

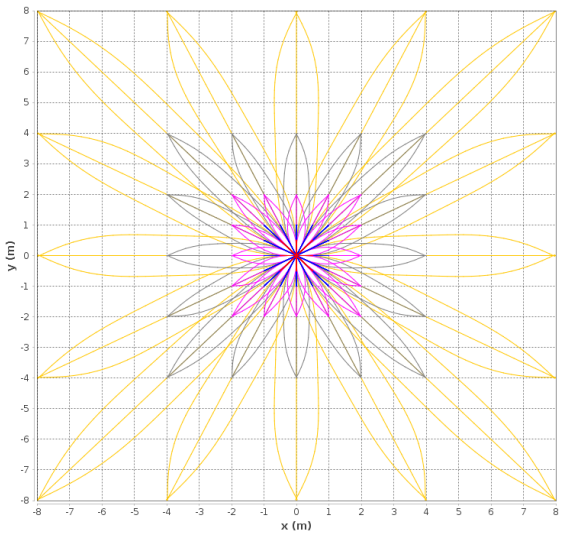


Figure 7: Control set used in the experiments: 336 trajectories connecting neighbors of levels 1, 2, 4, 8 and 16 —in red, blue, pink, gray and yellow, respectively. The highest fidelity, f^+, is 0.5 m.

retrieved faster and then improved iteratively until the optimal one is found or the available computing time is used up. The planner takes advantage of this possibility by adjusting the quality of the solution in terms of traversal time. However, the safety of the solutions is not affected by the use of anytime search. This is because the heuristic only estimates the traversal time of the path, even though the cost function has three elements —probability of collision, traversal time and uncertainty at the goal. Since anytime search works inflating the heuristic and these elements are compared hierarchically —see Eq. 10—, the probability of collision is always minimal regardless the value of ϵ.

5. Results

In this section results of the proposed motion planner in different scenarios and uncertainty conditions are reported. Moreover, tests varying the robot shape were run, showing the relevance of taking it into account to predict the probability of collision along the paths. Also, results for the proposed graduated fidelity approach are detailed, comparing them with those of a standard state lattice planner. Thus, we show the ability of the proposed method to improve the efficiency while maintaining the performance —the cost of the solutions. Finally, we detail results for H2DMR, the proposed multi-resolution heuristic, focusing on its validity and the improved efficiency, specially in large scenarios. Runtimes reported in this section are for a computer with a CPU Intel Core^{TM} i7-4790 at 3.6 GHz and 16 GB of RAM.

All tests were run on a 2D world and a robot with Ackermann dynamics, in which $M_t = 0.01 \cdot I$. Robot dimensions are 3.0×0.75 m, with the rotation center located at 0.9 m from its back side, centered in the short axis. Linear speed ranges between 0 and 0.5 m/s, and the angular speed is between -30 and 30 deg/s. The state vector is 5-dimensional and contains the pose of the rotation center of the robot, and also the linear and angular speeds:

$$[x, y, \theta, v, \omega] \tag{12}$$

The control vector contains the linear and angular speeds. These commands are sent to the robot at 3 Hz. The measurements are the position and the heading, with an uncertainty of $N_t = 0.01 \cdot I$ in normal conditions. However, there are location denied areas in the environments in which the robot does not receive any measurement. Within them, sensing uncertainty is $N_t = \infty \cdot I$.

With regard to the dynamics model, it was learned from 183 s of navigation data for a simulated robot in Gazebo. Then, a set of 336 motion primitives was obtained —shown in Fig. 7. These primitives connect states of distances 1, 2, 4, 8 and 16 in a lattice with a highest fidelity, f^+, of 0.5 m.

Figure 8 shows a plan obtained with the proposed algorithm. Areas with high uncertainty exist in the region where maneuvering is required to enter the corridor, which together with the robot dimensions makes difficult to find a safe path trough it. In spite of this, the planner provides a solution which is not only optimal but also smooth. This contrasts with other approaches based on RRT and RRT*, which rely on smoothing techniques to achieve similar results, affecting the predicted uncertainty.

Figure 9 details the behavior of the planner under different uncertainty conditions. In this environment, several alternatives are available to reach the goal, but the maneuvering is limited due to the robot length. Moreover, motion uncertainty has a great impact in the front side of the robot, since any small deviation in heading can re-

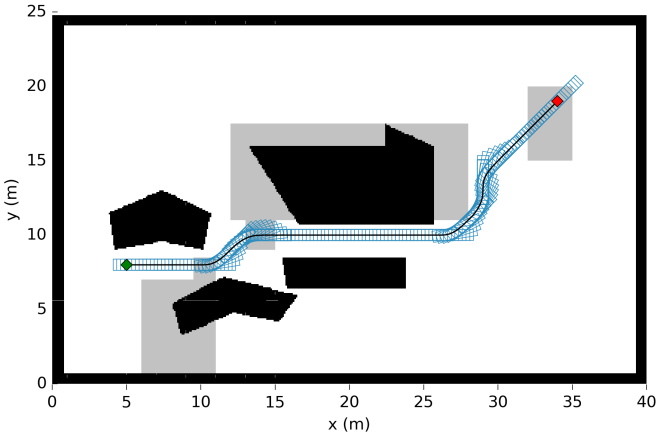
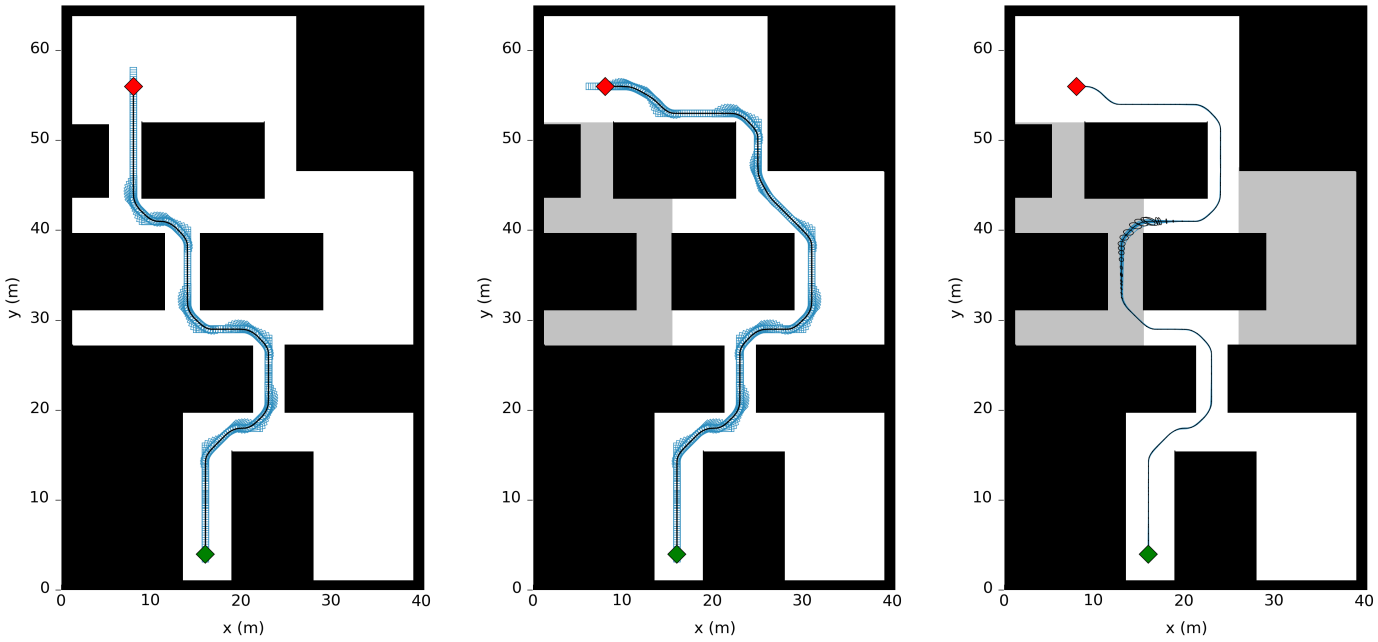


Figure 8: Planned path for a car-like robot with dimensions 3.0×0.75 m in a cluttered environment. Obstacles are in black, while regions in light gray are those with no location signal. The maximum-likelihood path is represented with a black line, with the robot trail in light blue. The green and red diamonds are the start and the goal points.



(a) (b) (c)

Figure 9: Planned paths for a car-like robot of 3.0×0.75 m under different uncertainty conditions. The robot trail is in blue, the maximum-likelihood path is in black, and the diamonds in green and red are the start and the goal points. (a) shows the path without location denied areas; (b) a different path which is selected when two of the corridors are location denied areas; and (c) another solution when the same happens in the corridor on the right. Here the black ellipses are $3 \cdot \sigma$ of the predicted PDFs.

sult in a collision. The chosen path depends on the PDFs along the different candidates, as the planner favors the safety of the solution rather than its traversal time. The optimal solution is shown in Fig. 9a, in which there are no location denied areas and motion uncertainty only arises from the controls. A safe plan that crosses the two narrow corridors is obtained, maneuvering only in those areas in which there is enough space to avoid collisions.

Fig. 9b shows the solution for the same environment but different uncertainty conditions. Here the probability of collision of the prior path is affected, as the motion uncertainty grows due to the location denial in the corridors of the left. Therefore, a safer alternative which avoids this area is selected despite increasing the traversal time. Having another location denied region in the right of the map, as shown in Fig. 9c, results in a variation of this path. In this case, the planner minimizes the probability of collision avoiding to maneuver just after leaving the location denied area in the corridor of the right. Instead, the planned path crosses the first narrow corridor and allows the robot to receive measurements before turning.

Fig. 9c shows in black the predicted PDFs, which explain the solution obtained. The uncertainty grows when the navigation is done without receiving measurements; thus, the distance to obstacles must be higher to maintain the safety of the path. After leaving those areas, the uncertainty shrinks due to the measurements and the use of a LQG controller, which is taken into account when predicting the distributions at planning time. Hence, the path can go through the final obstacles and safely reach the goal.

As mentioned before, Fig. 9c shows in black the uncertainty estimated at planning time. The ellipses represent $3 \cdot \sigma$ of the PDFs, a 99.7% of confidence. Moreover, $1,000$ simulations of each planned path were run to estimate the real behavior of the robot, and also to compare the predicted distributions with the real ones. These simulations allow the comparison of the probability of collision obtained from the PDFs, $\hat{p}_c$, and the one from the simulated paths, p_c, in which collisions were checked with the real shape.

In Fig. 10 the behavior of the planner in a map with high uncertainty is shown. In this experiment the robot has a significant uncertainty at the beginning of the path —$\Sigma_{x^0} = I$—, and the only area in which measurements are received is far away from the obstacles. Directly crossing the door following the shortest path, without taking into account the uncertainty conditions, is too risky and it would result in a plan with a 55% of probability of collision. Instead, our planner obtains a path which goes outside the location denied area to diminish the uncertainty and ensure that the robot safely crosses the door. In fact, our planner is conservative in this matter and considers that the robot only receives measurements when there is no overlap between the PDFs and the location denied areas.

Prior works already reported planning results in this environment, so we used it to compare the performance of our proposal with them. The approach of [17] finds a

Table 1: Planning results for the experiments in this section using the graduated fidelity lattice and the heuristic H2DMR. Robot dimensions are 3.0×0.75 m. The planner was run with $\epsilon_0 = 1.5$ and decreasing it iteratively. Column ϵ has the value for which the optimal solution was found. Columns "Iterations", "Insertions" and "Time" detail the planning efficiency —nodes expanded by AD*, nodes inserted in the OPEN queue and planning time. "Cost" is the traversal time. The estimated probability of collision ($\hat{p}_c$) and the real one (p_c), obtained from $1,000$ simulations, are 0 in all cases. All times are in seconds.

		Planning			Solution
Problem	ϵ	**Iterations**	**Insertions**	**Time (s)**	**Cost (s)**
Fig. 8	1.05	228	1,043	0.38	68.66
Fig. 9a	1.5	252	639	0.32	132.02
Fig. 9b	1.5	659	2194	0.85	157.37
Fig. 9c	1.5	799	1,872	2.26	172.28
Fig. 10	1.03	684	2,595	2.18	108.27
Fig. 11b	1.06	611	1,559	0.96	131.14

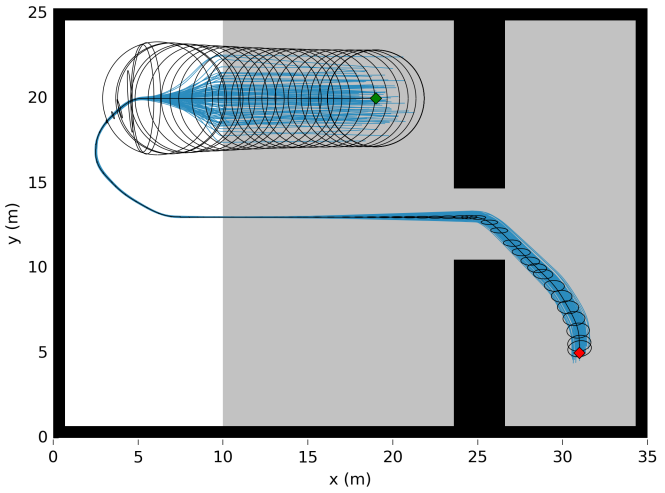


Figure 10: Planned path for a car-like robot of 3.0×0.75 m in an environment with high uncertainty. The nominal path is in black, while the simulated executions are in blue. The ellipses are $3 \cdot \sigma$ of the predicted PDFs. The robot receives measurements only when the PDF is completely outside the gray region.

valid solution in these conditions and converges in approximately $17,500$ iterations, with a planning time of 40 s. As reported in this work, the method of [16] fails in this environment due to the Voronoi-bias that prevents the expansion of the paths going through the area in which the robot can receive measurements. Our approach outperforms these results and finds the best solution in 2.18 s and 684 iterations for our dynamics model[1].

The approach of [15] finds an initial path and then obtains a locally-optimal control policy. The convergence of their method took 3.65 s, and the 93% of the simulated executions were collision free. On the contrary, for the plans obtained with our proposal the simulations showed that in all cases the robot was able to safely cross the narrow passage, also varying the starting point. Their method does not assume a fixed control gain along each section of the planned path. However, with their approach a different control policy has to be obtained for each homotopy class of trajectory in order to find the globally optimal solution. As a consequence, the planning times would be higher in more complex environments —i.e. the map of Fig. 9 already contains 4 homotopy classes for the given start and goal points.

Table 1 contains the detailed results for all the environments in this section. In all cases the planner finds the optimal solution in a few seconds.

5.1. Reliable collision check

The capability of the planner to work with different robot shapes is shown in Fig. 11. In this example the planner was run in the same environment and uncertainty conditions, but using two different shapes, a squared one with size 0.75×0.75 m, and another one with size 3.0×0.75 m. Approximating the former by a circle could be a reasonable assumption, but for the latter this cannot be done without drawbacks.

The proposed method accurately estimates the probability of collision of the path in accordance with the real shape, which leads to the obtention of different paths in the example of Fig. 11. Fig. 11a shows the motion plan obtained for a squared robot of 0.75×0.75 m. This robot can safely pass through the room at the center of the map after maneuvering in the narrow passage. However, for the robot of 3.0×0.75 m this maneuver would be too risky due to its dimensions. In fact, the robot collides with its right front corner even before crossing the door, while afterwards the collision involves all its right side. Our motion planner manages that and obtains for this robot a slightly longer path, but with the minimal probability of collision, shown in blue in Fig. 11b. Table 1 details the planning results for the 3.0×0.75 m shape.

This example shows the relevance of taking into account the real shape, which is systematically disregarded in the state of the art. In this environment, approximating the large robot by the inscribed circle —with a diameter of 0.75 m— would result in underestimating the probability of collision, which would lead to unplanned collisions, as shown in red in Fig. 11b. The same happens to chance-constraint planning only using the PDFs. On the contrary, with the outer circle the probability of collision of the paths would be highly overestimated.

[1] [15, 17] use a punctual robot with 2D dynamics, while our results are for a car-like robot.

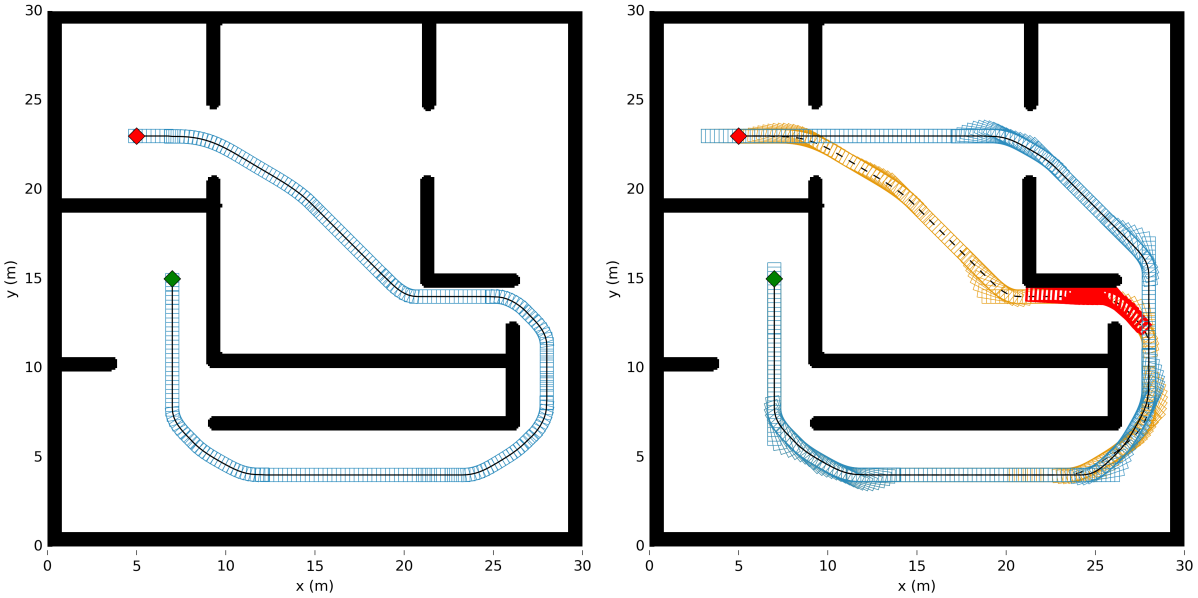


Figure 11: Influence of the robot shape in the planned paths: (a) shows the plan for a shape of 0.75×0.75 m (b) shows in blue the path for a larger robot, of 3.0×0.75 m, in orange the path that this robot would follow if its real dimensions were not taken into account, and in red the most probable collisions in that situation. The diamonds in green and red are the start and the goal points.

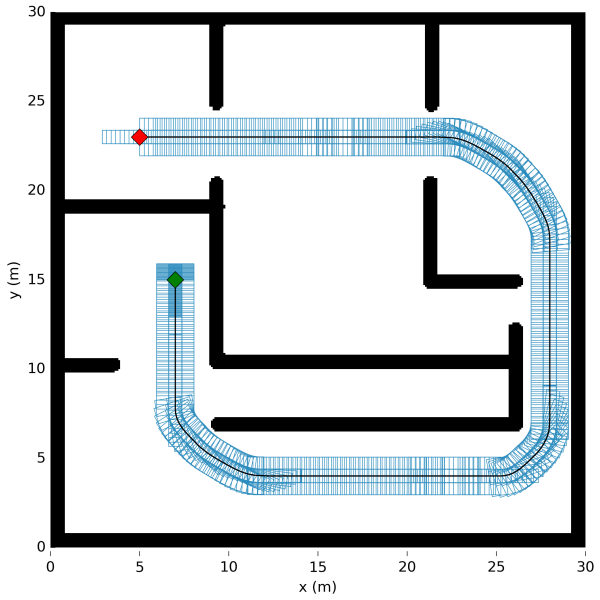


Figure 12: Planned path for a robot with "T" shape, of 3.0×2.2 m, shown in blue. The diamonds in green and red are the start and the goal points.

In Fig. 12 we show that our approach is able to reliably estimate the probability of collision and find safe paths regardless the robot shape. In this experiment the robot shape is a "T", which makes more difficult to maneuver in narrow spaces. The planned path is similar to that of Fig. 11b, since the robot cannot enter through the first door due to its dimensions. However, in this case the robot has to enter the corridor completely aligned with the walls to avoid collisions with them. Similarly, the robot has to take the curve wide before crossing the doors at the top of the map. The planner finds the optimal solution in 1.38 s — 525 iterations. The cost is 136.89 s, slightly higher than in the experiment of Fig. 11b.

Whatever the kind of environment, the uncertainty in heading clearly influences the probability of collision for those robots with non-circular shapes. This is aggravated if, due to the robot dimensions, like the one in the example, slight changes in heading cause large variations in the distance to the obstacles. This is managed by the proposed method, since the probability of collision is obtained by sampling PDFs in a way that not only takes into account deviations in position, but also in heading. The estimated probability of collision, $\hat{p}_c$, is correct regardless the robot dimensions, and it approximates well the real value, p_c, obtained from 1,000 simulations of the planned path. Thus, reliable and safe paths are found for any shape.

Fig 13 compares the estimation error of our method with that of other approaches [16, 19] which approximate the robot by its bounding circle. They estimate the probability of collision using the regularized gamma function, $\Gamma(n/2, r^2/2)$, which gives the probability that a sample is within r standard deviations for a multivariate Gaussian distribution of dimension n. A challenging situation for planning is when a non-circular robot is approaching a diagonally placed obstacle —i.e. entering a corridor or turning a corner. In these situations, the reliability of the gamma function falls due to not taking into account the uncertainty in heading and approximating the robot by a

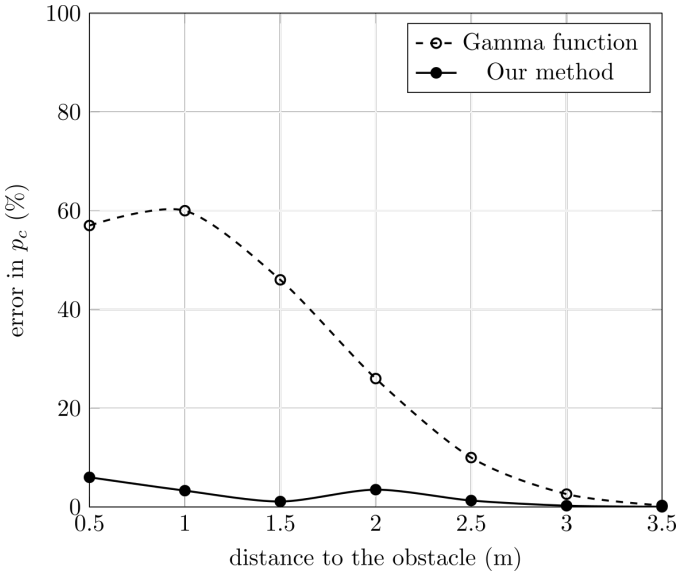


Figure 13: Error of the probability of collision estimated with our method compared with that of the gamma function. The error was measured varying the distance between a robot with shape $3.0 \times 0.75\ m$ and a single obstacle placed diagonally with it, and for $\Sigma = I$.

circle, which results in a significant error of the estimated probability of collision with respect to the real one. Instead, our method accurately approximates it with only a 1.5% of average error.

5.2. Graduated fidelity lattice

Most of the planning time, approximately a 80%, is spent in propagating the PDFs through the candidate paths and estimating their probability of collision. Diminishing the fidelity of the lattice allows to reduce the number of paths contained in the state lattice, obtaining better computation times. However, doing it in a general way affects the robustness of the system and its capability to find near-optimal solutions. On the contrary, the proposed graduated fidelity approach allows to drastically reduce the number of analyzed paths, highly increasing the efficiency of the planner while keeping nearly the same performance.

Fig. 14 shows the lattice for the motion plan in Fig. 8 and the results of the graduated fidelity approach. The highest fidelity is selected in those areas with more density of obstacles, such as the corridors. In those areas the octree cannot be compacted due to the different occupancy information of adjacent cells, and therefore cells tend to be small. This leads to the selection of the highest fidelity, f^+, using the motion primitives with the minimum length, $0.5\ m$. Similarly, higher fidelities are selected when the uncertainty affects the probability of collision of the paths —i.e. maneuvering in the vicinity of obstacles.

The opposite occurs in the uncluttered areas of the map. In those regions lower fidelities are selected, which significantly reduces the density of states in the lattice. Moreover, reducing the maneuverability in those areas has a limited influence in the cost of the solutions retrieved, while saving significant computational resources. Only near the goal an exception is made, and the whole set of actions $\mathcal{U}$ is used to facilitate maneuvering in this area regardless the density of obstacles.

Table 2 shows a comparison between the executions of the proposed planner with graduated fidelity, GF, and those of a standard state lattice planner. For the latter, the connectivity is given by the whole set of primitives.

This comparison shows a significant reduction in the number of iterations of the search algorithm —an average decrease of 83.3%—, but more importantly in the number of insertions in the OPEN queue —reduced in a 88.1%. The latter highly influences the planning time, as the insertions involve propagating the PDFs and estimating the probability of collision along the candidate paths. Thus, the planning time diminished on average by 88.5% when making use of the graduated fidelity lattice. Conversely, the cost of the solutions slightly increases —an average of 6.2%— due to the selection of longer maneuvers for turning. However, when compared with the solutions provided without the graduated fidelity approach, the differences are minimal.

5.3. Anytime search

This subsection focuses on the capabilities of combining anytime search with a graduated fidelity lattice. This feature allows obtaining sub-optimal solutions faster and refine them later, which is desirable when using the planner in real world domains.

To test this feature the planner was run with $\epsilon_0 = 1.5$, and later the parameter was iteratively decreased to refine the solution. Thus, an initial plan is retrieved in a very short time and the robot can move while the plan is improved. Table 3 details the results of these tests.

The cost of the sub-optimal solutions is bounded by the value of ϵ, so the cost of the solution decreases in accordance with the value of the parameter. Also, the higher

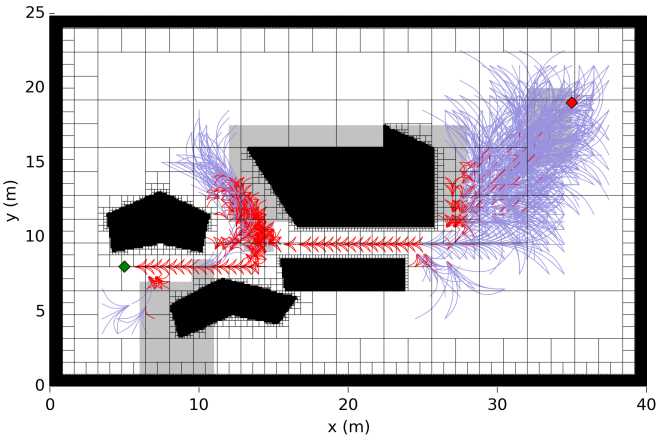


Figure 14: Graduated fidelity selection for an environment combining cluttered and non-cluttered areas. Areas making use of the highest fidelity are in red, while those with lower ones are in blue. This lattice corresponds to the motion plan of Fig. 8.

Table 2: Performance comparison between the planner using the graduated fidelity lattice (GF) and a standard one. All tests were run with $\epsilon = 1.0$. Robot shape is 3.0×0.75 m. Columns "Iterations", "Insertions" and "Time" detail the planning efficiency —nodes expanded by AD*, nodes inserted in the OPEN queue and planning time. "Cost" is the traversal time. The estimated probability of collision ($\hat{p}_c$) and the real one (p_c), obtained from 1,000 simulations, are 0 in all cases. All times are in seconds.

		Planning			Solution
Problem	**GF**	**Iterations**	**Insertions**	**Time (s)**	**Cost (s)**
Fig. 8	✓	144	586	0.59	68.66
Fig. 8	✗	272	1,446	1.58	68.04
Fig. 9a	✓	318	774	0.84	132.02
Fig. 9a	✗	1,501	6,287	5.25	127.19
Fig. 9b	✓	820	2,093	1.54	154.71
Fig. 9b	✗	2,957	12,346	12.96	146.91
Fig. 9c	✓	2,030	4,522	4.52	172.28
Fig. 9c	✗	7,277	26,942	25.90	160.55
Fig. 10	✓	1,302	2,868	2.68	108.27
Fig. 10	✗	15,952	42,594	45.38	95.96
Fig. 11b	✓	593	1,321	1.29	131.14
Fig. 11b	✗	3,196	12,183	8.53	126.27

Table 3: Anytime planning performance. Planner was run starting in $\epsilon_0 = 1.5$, decreasing it iteratively. Column ϵ has the value for which the optimal solution was found. Robot shape is 3.0×0.75 m. Columns "Iterations", "Insertions" and "Time" detail the planning efficiency —nodes expanded by AD*, nodes inserted in the OPEN queue and planning time. "Cost" is the traversal time. The estimated probability of collision ($\hat{p}_c$) and the real one (p_c), obtained from 1,000 simulations, are 0 in all cases. All times are in seconds.

		Planning			Solution
Problem	ϵ	**Iterations**	**Insertions**	**Time (s)**	**Cost (s)**
Fig. 8	1.5	92	484	0.15	68.78
Fig. 8	1.05	228	1,043	0.38	68.66
Fig. 9a	1.5	252	639	0.32	132.02
Fig. 9b	1.5	659	2194	0.85	157.37
Fig. 9b	1.15	1,032	3,152	1.40	154.71
Fig. 9c	1.5	799	1,872	2.26	172.28
Fig. 10	1.5	466	1,164	1.57	112.79
Fig. 10	1.03	684	2,595	2.18	108.27
Fig. 11b	1.5	64	225	0.12	132.98
Fig. 11b	1.06	611	1,559	0.96	131.14

its value, the lower the number of iterations and insertions, and consequently the planning time. For $\epsilon = 1.0$ the solution retrieved is guaranteed to be optimal. While this applies to the traversal time of the path, the probability of collision is always minimized —$\hat{p}_c$ is always minimal, even for anytime solutions. This is because anytime search works scaling the value of the heuristic by ϵ, which only estimates the traversal time, not the rest of the elements of the cost function, as detailed in Sec. 4.3.

It is specially noticeable the reduction in the planning times for those cases in which the heuristic is too optimistic with respect to the real cost of the path —due to the uncertainty conditions, i.e. Fig. 10. In those cases, finding the optimal solution is not trivial and anytime search avoids long waiting times. However, while increasing ϵ_0 speeds up the obtention of the initial solutions, it may lead to longer computational times if too many values of ϵ are tested. This is because each time ϵ changes, the priorities of all the nodes in *OPEN* have to be updated, which can be costly depending on its number. Therefore, if there are too many nodes in *OPEN* it might be better to obtain a new plan from scratch, as pointed out in [23].

5.4. Multi-resolution heuristic

Several tests were run to compare the performance of H2DMR, the multi-resolution heuristic, with the one with a fixed resolution, H2D. For the latter, the grid is built with the resolution given by the highest fidelity of the motion primitives, f^+. A map of an empty environment with dimensions 50×50 m was built for testing purposes, and then several octrees with the same occupancy information were obtained. For all of them, the minimum size of the cells is 0.1 m while the maximum one, C_+, ranges between 0.1 m and 25.6 m, depending on the test. Finally, both heuristics were run in each octree until they completely explored the free space —until the *OPEN* queue was empty.

Figure 15 compares the performance of H2DMR with that of H2D in terms of runtime and number of iterations. In the case of H2D, the number of iterations is constant, as the grid is built only taking into account f^+. Conversely, as H2DMR takes advantage of the octree structure to decide the best resolution for each area of the map, the num-

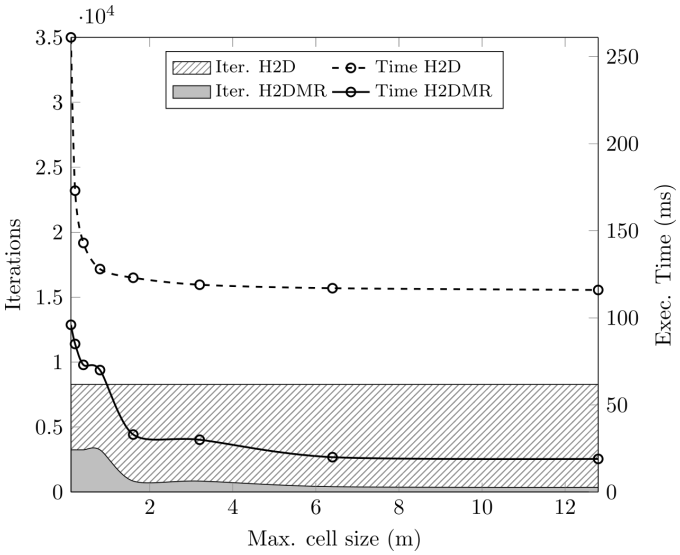


Figure 15: Performance of H2DMR compared with that of H2D. Results for the run time and the number of iterations needed to calculate heuristics over an empty 50×50 m map are shown.

Table 4: Efficiency of H2DMR compared with that of H2D. Runtimes and number of iterations to obtain the heuristics over an empty map of 50×50 m are given. The maximum resolution of the octree is 0.1 m, while $f^+ = 0.5$ m. Tests were run for several maximum cell sizes, C_+ (in meters).

	Iterations			Time (ms)		
C_+	SR	MR	Gain	SR	MR	Gain
0.1	8,281	3,250	60.8%	261.2	96.0	63.2%
0.2	8,281	3,250	60.8%	173.7	85.3	50.9%
0.4	8,281	3,250	60.8%	143.8	73.0	49.2%
0.8	8,281	3,250	60.8%	128.6	70.0	45.6%
1.6	8,281	842	89.8%	123.4	33.0	73.3%
3.2	8,281	842	89.3%	119.0	30.0	74.8%
6.4	8,281	410	95.1%	117.8	20.0	83.0%
12.8	8,281	362	95.6%	116.2	19.0	83.7%

ber of iterations is significantly lower, and so it is the obtention time. Not only H2DMR is more efficient for all resolutions, but also the divergence from H2D increases with the cell size. Consequently, the benefits of this approach are specially noticeable in large environments, where the octree cells can be larger. The reduction in the runtime of H2D is only related with the collision check operations, which are more efficient with larger octree cells. Table 4 gives full detail of these tests, also quantifying the performance gain achieved by H2DMR with respect to H2D for each maximum octree resolution.

Finally, Fig. 17 compares the grids built by H2DMR and H2D. The goal is located at the center of the map. In the case of H2DMR, shown in Fig. 17b, the resolution goes in accordance with the size of the octree cells and f^+ —0.5 m in this example. This results in a grid with a complexity considerably lower than that of H2D. However, this grid has an increased connectivity, as all points between adjacent cells have a link between them, instead of the 8-connected grid of H2D. This compensates the reduction in the number of paths due to diminishing the density of positions in the grid, specially in those areas with large octree cells. This allows retaining the quality of the estimations. In fact, the costs given by H2DMR were compared with those of H2D, showing an average reduction of a 4%. This means that H2DMR is also more precise than H2D.

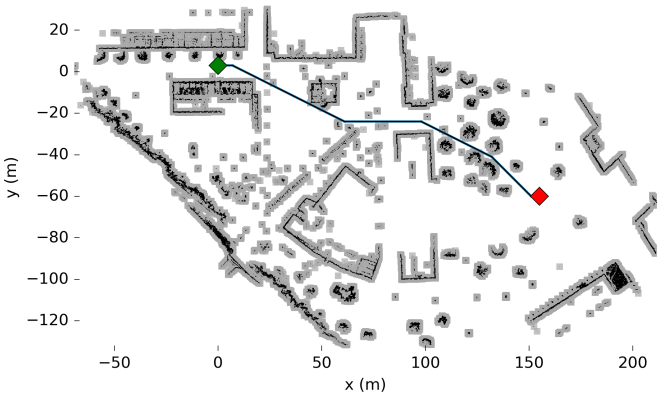


Figure 16: Planned path for an odometric robot of 0.75×0.75 m in a real environment. Obstacles are in black, while regions in light gray are those with no location signal. The robot trail is in blue and the diamonds in green and red are the start and the goal points.

5.5. Planning in a real environment

In this section we report results for a large real environment —292×167 m. It is a map of the Freiburg University campus built from 3D laser scans, and it was obtained from a public dataset[2]. The location denied areas are located around the obstacles to simulate the effect that they have on the GPS signal. Concretely, for this experiment we considered that the space of 2 m around the obstacles has high uncertainty. The motion model was that of an odometric robot. The set of motion primitives has 246 motions which connect the lattice states of distances 0, 1, 2, 4, 8 and 16. The highest fidelity, f^+, is 1.0 m.

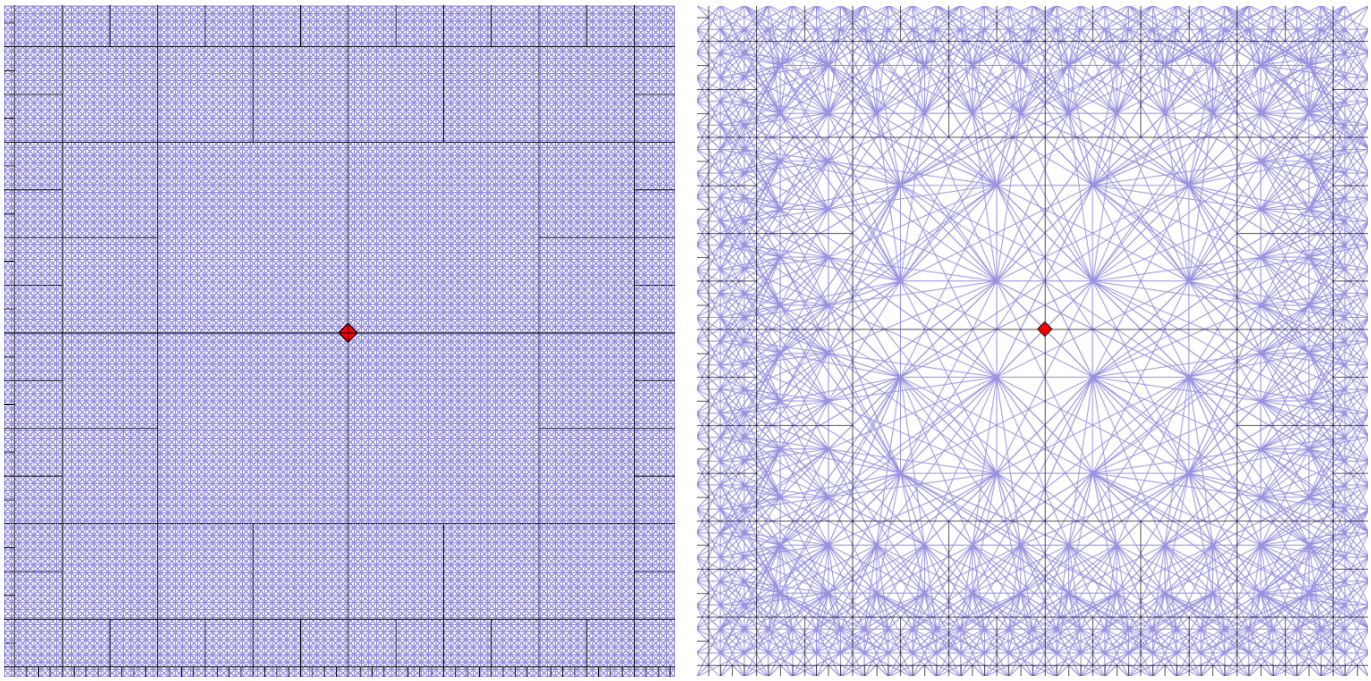
Figure 16 shows the optimal solution for this environment. Our approach finds a valid path, and the robot is able to safely maneuver between the trees of the right. Its cost was 359.28 s and it was obtained in 34 s of planning time, after 26,972 iterations, for $\epsilon = 1.05$. Due to the anytime search capabilities of AD*, a first sub-optimal solution is available after 1.1 s of planning time, in 823 iterations, with a cost of 402.20 s for $\epsilon = 1.5$.

The cluttered areas of the map do not affect the ability of the planner to find the shortest path. Nevertheless, the planner might require slightly higher runtimes due to the size of the map and the cluttered areas.

6. Conclusions and future work

We have presented a motion planner based on state lattices which manages motion and sensing uncertainty.

[2] Available at http://ais.informatik.uni-freiburg.de/projects/datasets/octomap/



(a) Single-resolution (H2D) (b) Muti-resolution (H2DMR)

Figure 17: Comparison of the grid generated by H2D with that of H2DMR. The grid is in blue, the structure of the octree is in black, and the goal is the red diamond. The environment is 50×50 m.

Our proposal introduces a reliable method to obtain the probability of collision of the paths taking into account the real shape of the robot. We have also introduced a novel graduated fidelity approach which adapts the fidelity of the lattice to the obstacles in the environment and the maneuverability of the robot, resulting in a drastic increase of the planning efficiency without affecting the performance. Moreover, we have proposed H2DMR, a multi-resolution heuristic which significantly improves the computation times required to estimate the cost to the goal taking into account the obstacles in the map.

We have validated our proposal with 21 tests in several environments, with different robot shapes, motion models and under a variety of uncertainty conditions. Results show the good performance of the motion planner. The probability of collision was always reliably estimated, even for shapes which cannot be approximated as circles. Due to the graduated fidelity lattice, planning time decreased on average by 88.5% with respect to a standard lattice, while the heuristic was obtained a 65.5% faster due to the multi-resolution grid. Experimental results show the anytime search capability of the planner, which allows obtaining safe paths regardless they are sub-optimal or not. All of the above results in an efficient and reliable approach.

Acknowledgments

This research was supported by the Spanish Ministry of Economy and Competitiveness —grants TIN2014-56633-C3-1-R and TIN2017-84796-C2-1-R—, and the Galician Ministry of Education, Culture and Universities —grants GRC2014/030 and accreditation 2016-2019, ED431G/08. These grants are co-funded by the European Regional Development Fund —ERDF/FEDER program.

[1] S. M. LaValle, Planning Algorithms, Cambridge University Press, Cambridge, U.K., 2006.
[2] S. Karaman, E. Frazzoli, Incremental sampling-based algorithms for optimal motion planning, Robotics Science and Systems VI 104.
[3] L. Janson, B. Ichter, M. Pavone, Deterministic Sampling-Based Motion Planning: Optimality, Complexity, and Performance, Vol. 2, Springer International Publishing, 2018, pp. 507–525.
[4] D. Meagher, Geometric modeling using octree encoding, Computer graphics and image processing 19 (2) (1982) 129–147.
[5] A. Hornung, K. M. Wurm, M. Bennewitz, C. Stachniss, W. Burgard, Octomap: An efficient probabilistic 3d mapping framework based on octrees, Autonomous Robots 34 (3) (2013) 189–206.
[6] S. M. LaValle, J. J. Kuffner, Randomized kinodynamic planning, The International Journal of Robotics Research 20 (5) (2001) 378–400.
[7] M. Pivtoraiko, A. Kelly, Efficient constrained path planning via search in state lattices, in: 8th International Symposium on Artificial Intelligence, Robotics and Automation (I-SAIRAS), 2005.
[8] M. Pivtoraiko, A. Kelly, Differentially constrained motion replanning using state lattices with graduated fidelity, in: IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2008, pp. 2611–2616.
[9] M. Likhachev, D. Ferguson, Planning Long Dynamically Feasible Maneuvers for Autonomous Vehicles, The International Journal of Robotics Research 28 (8) (2009) 933–945.
[10] R. Knepper, A. Kelly, High Performance State Lattice Planning Using Heuristic Look-Up Tables, IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (2006) 3375–3380.
[11] R. Alterovitz, T. Siméon, K. Goldberg, The stochastic motion

roadmap: A sampling framework for planning with Markov motion uncertainty, in: Robotics: Science and Systems, 2007, pp. 246–253.
[12] L. P. Kaelbling, M. L. Littman, A. R. Cassandra, Planning and acting in partially observable stochastic domains, Artificial intelligence 101 (1) (1998) 99–134.
[13] C. H. Papadimitriou, J. N. Tsitsiklis, The complexity of Markov decision processes, Mathematics of operations research 12 (3) (1987) 441–450.
[14] N. Roy, S. Thrun, Coastal navigation with mobile robots, in: Advances in Neural Information Processing Systems, 2000, pp. 1043–1049.
[15] J. Van den Berg, S. Patil, R. Alterovitz, Motion planning under uncertainty using iterative local optimization in belief space, The International Journal of Robotics Research 31 (11) (2012) 1263–1278.
[16] J. Van den Berg, P. Abbeel, K. Goldberg, LQG-MP: Optimized path planning for robots with motion uncertainty and imperfect state information, The International Journal of Robotics Research 30 (7) (2011) 895–913.
[17] A. Bry, N. Roy, Rapidly-exploring random belief trees for motion planning under uncertainty, in: IEEE International Conference on Robotics and Automation (ICRA), 2011, pp. 723–730.
[18] B. D. Luders, S. Karaman, J. P. How, Robust sampling-based motion planning with asymptotic optimality guarantees, in: AIAA Guidance, Navigation, and Control (GNC) Conference, 2013, p. 5097.
[19] S. U. Lee, R. Gonzalez, K. Iagnemma, Robust sampling-based motion planning for autonomous tracked vehicles in deformable high slip terrain, in: Robotics and Automation (ICRA), 2016 IEEE International Conference on, IEEE, 2016, pp. 2569–2574.
[20] T. M. Howard, A. Kelly, Optimal rough terrain trajectory generation for wheeled mobile robots, The International Journal of Robotics Research 26 (2) (2007) 141–166.
[21] C. De Boor, A Practical Guide to Splines, Vol. 27 of Applied Mathematical Sciences, Springer-Verlag New York, 1978.
[22] P. Abbeel, Apprenticeship learning and reinforcement learning with application to robotic control, Stanford University, 2008.
[23] M. Likhachev, D. Ferguson, G. Gordon, A. Stentz, S. Thrun, Anytime dynamic A*: An anytime, replanning algorithm, in: Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS), 2005, pp. 262–271.
[24] M. Pivtoraiko, R. A. Knepper, A. Kelly, Differentially constrained mobile robot motion planning in state lattices, Journal of Field Robotics 26 (3) (2009) 308–333.
[25] D. Bertsekas, Dynamic programming and optimal control, Vol. 1, Athena scientific Belmont, MA, 1995.
[26] S. J. Julier, J. K. Uhlmann, H. F. Durrant-Whyte, A new approach for filtering nonlinear systems, in: Proceedings of the American Control Conference, Vol. 3, IEEE, 1995, pp. 1628–1632.