Combined Trajectory Generation and Path Planning for Mobile Robots Using Lattices with Hybrid Dimensionality

Janko Petereit, Thomas Emter, and Christian W. Frey

Fraunhofer Institute of Optronics, System Technologies
and Image Exploitation IOSB, Karlsruhe, Germany,
{janko.petereit, thomas.emter, christian.frey}@iosb.fraunhofer.de

Abstract. Safe navigation for mobile robots in unstructured and dynamic environments is still a challenging research topic. Most approaches use separate algorithms for global path planning and local obstacle avoidance. However, this generally results in globally sub-optimal navigation strategies. In this paper, we present an algorithm which combines these two navigation tasks in a single integrated approach. For this purpose, we introduce a novel search space, namely, a state $\times$ time lattice with hybrid dimensionality. We describe a procedure for generating high-quality motion primitives for a mobile robot with four-wheel steering to define the motion in this lattice. Our algorithm computes a hybrid solution for the path planning problem consisting of a trajectory (i.e., a path with time component) in the imminent future, a dynamically feasible path in the near future, and a kinematically feasible path for the remaining time to the goal. Finally, we provide some results of our algorithm in action to prove its high solution quality and real-time capability.

Keywords: mobile robot motion planning, hybrid-dimensional planning, state lattice planner

1 Introduction

A common usage scenario for autonomous mobile robots is the support of rescue teams after natural disasters or industrial accidents. Robots can assist by exploring and mapping the disaster area, acquiring important environmental data, searching for victims or simply carrying heavy loads. This task is characterized by a mostly unknown and unstructured environment, which is highly dynamic due to human rescue workers and other rescue vehicles operating in the close vicinity of the robot.

In order to efficiently accomplish the mission while at the same time moving safely between the (possibly dynamic) obstacles in such an environment, the robot needs fast and high-quality path planning as well as a strategy for reliable obstacle avoidance. In the last few years mainly two approaches for global path planning that consider the kinematic constraints of car-like robots have emerged for planning in unstructured environments. The first one combines continuous

motion primitives with a discrete search space by spanning a tree of continuous states, thus constituting a hybrid search space (Hybrid A* [1]). The second approach utilizes specifically constructed motion primitives which cause the reachable set to form a lattice structure in the robot's state space [2]. Therefore, the graph search, although using motion primitives which represent *continuous* motion, operates in a *discrete* search space.

In their original versions these global path planners focused merely on the search of *kinematically* feasible paths. However, for the application to autonomous vehicles, it is favorable to guarantee that the found solutions are also *dynamically* feasible. This naturally results in a search space with increased dimensionality, which makes the path planning even more challenging and complex. In the last two decades a lot of effort went into the development of algorithms to tackle this increased complexity. Two examples are the well-known Probabilistic Roadmaps [3] and Rapidly-exploring Random Trees (RRTs) [4]. Although both algorithms are probabilistically complete and have been applied to mobile robots in the past, they have some disadvantages for this particular application area. Probabilistic Roadmaps lose their benefit of precomputed roadmaps of complex configuration spaces in a rapidly changing environment, which is the case in the presence of dynamic obstacles. RRTs generally provide non-optimal solutions and it is very hard to incorporate additional information (like terrain quality or traversal risk) into the search. To overcome these limitations, particularly with regard to mobile robot applications, state lattices have been successfully extended to search spaces of higher dimensionality to allow for a planning of dynamically feasible maneuvers [5]. However, the algorithm proposed in [5] plans a completely dynamically feasible maneuver from the start to the goal, which is generally not necessary and thus may waste valuable computation time. For this purpose, we propose a novel algorithm that relaxes the accuracy requirements of the motion plan with increasing time while still guaranteeing dynamically feasible motions in the close future.

1.1 Problem Statement and Proposed Solution

Especially in a highly dynamic environment (e.g., in the presence of human rescue workers), it is important to include these dynamic obstacles already in the global path planning as an unaware global path planner followed by a subsequent local obstacle avoidance would generally lead to sub-optimal driving strategies.

In this paper, we present a novel algorithm, which combines the trajectory generation (i.e., planning in state $\times$ time space) with global path planning. It exploits the fact that the requirements imposed on the accuracy of the planned robot motion decrease the more it extends into the future. For this purpose, the dimensionality of the search space is successively reduced: The search starts in the full-dimensional state $\times$ time space for the immediate future (thus generating time-parametrized trajectories), then continues through still dynamically feasible maneuvers, and finally considers merely kinematically feasible paths in the far future. For distant regions an even further reduction could be made by also dropping the kinematic feasibility and reducing to a simple grid search.

However, we deliberately refrain from this option as we impose some minimum requirements on the resulting path (namely, kinematic feasibility).

1.2 Related Work

There are some recent research results closely related to our approach. For example, Ziegler and Stiller [6] used spatiotemporal lattices for planning on-road driving maneuvers. However, this algorithm is not suitable for planning in unstructured environments. In [7] a hybrid approach which considers time during planning for a specific time horizon is proposed, but after reaching this point in time, it reduces to a simple 2D grid search not considering dynamic or kinematic constraints any more. In past research, we have extended this approach to at least guarantee the kinematic feasibility of the resulting motion plan [8], and proposed a multi-resolution concept to speed-up the hybrid-dimensional planning algorithm [9]. However, all these previous research results share the drawback of an abrupt transition from full-dimensional state × time states to only kinematically feasible states. The contribution of this paper is to fill this gap by introducing a concept of successive dimensionality reduction in order to lower the fidelity of the robot's state representation in a more gradual way. This will allow for a better trade-off between planning quality and computational performance.

The general idea of path planning in a search space with adaptive dimensionality has been addressed in [10], however, it is restricted to mere *path* planning – *trajectories* are not considered.

2 Algorithm

The presented algorithm consists mainly of the following three steps, of which steps 1 and 2 can be precomputed off-line.

1. Construction of a state × time lattice L_0 with full dimensionality.
2. Repeated projection of the state × time lattice L_0 to state lattices with lower dimensionality: $L_1, L_2, \ldots, L_{\max}$
3. Graph search in the generated state lattices, starting in L_0 and weaving through the state lattices with decreasing dimensionality. Special edges connect lattices of subsequent dimensionality to allow for transitions from L_k to L_{k+1}.

Throughout this paper, we will explain the algorithm using the example of a mobile robot, whose dynamics can be described using a general nonlinear system model

$$\dot{\boldsymbol{x}} = \boldsymbol{f}(\boldsymbol{x}, \boldsymbol{u}) \tag{1}$$

where the state $\boldsymbol{x}$ consists of the robot's position x and y, its orientation θ, and its translational velocity v. The input $\boldsymbol{u}$ contains the commanded acceleration u_a and the steering angle u_β.

However, our approach is not restricted to this specific system model, which is why we do not go into detail here. In fact, it can be applied to a wide range of robotic systems which shall be empowered to act in a dynamic environment.

2.1 Motion Primitive Generation

State lattice planners are commonly based on a set of motion primitives which span the search space. A motion primitive is a short time driving strategy which connects a state s to a succeeding state s' in the robot's state space. By storing the associated inputs that are needed to drive the system from s to s' along with each motion primitive, an overall driving strategy can be reconstructed from the motion primitives that form the solution of the trajectory/path planning problem.

There is a vast variety of approaches for the construction of these motion primitives. Bicchi et al. have shown in [11] which conditions a system and its inputs must satisfy so that its reachable set forms a lattice. However, for mobile robot applications this generally has the disadvantage of non-uniform heading discretization. Rufli and Siegwart overcome this limitation by bending the state lattice towards an a priori known path [12] but this approach is not well suited for heavily unstructured environments. Pivtoraiko et al. [2] use the approach presented by Kelly and Nagy [13], which is based on curvature polynomials, which approximate the vehicle motion.

In [8] we have shown that in the case of a simplified system model of a mobile robot with four-wheel steering the integration of the system of differential equations (1) can be done analytically for constant inputs u_a and u_β. This allows for a very efficient simulation of robot trajectories which are the basis for acquiring the needed motion primitives. In this section we will recap from [8] the procedure for constructing a high dimensional state $\times$ time lattice by sampling high-dimensional motion primitives, however, we will put it on a more formal basis in order to be able to conveniently derive the gradual dimensionality reduction in the following sections and to enable the smooth integration in the multi-resolution concept that we have proposed in [9].

First, the desired quantization of each dimension of the full-dimensional state $\times$ time lattice

$$L_0 = X \times Y \times \Theta \times V \times T \tag{2}$$

has to be set. Although, in principle, these could be chosen arbitrarily, the final choice has a huge impact on the outdegree of the lattice points. Furthermore, as we require only "translational invariance" of a motion primitive with respect to the x, y, and t dimensions, the remaining θ and v dimensions may be discretized in a non-equidistant way, which is especially useful for the v dimension. The consequence of the fact that motion primitives are only "translationally invariant" with respect to the x, y, and t dimensions is that it is necessary to sample individual motion primitive sets for states that start at different θ and v configurations. Because of its shape, we call the set $B(\theta, v)$ of all motion primitives that originate from an identical start configuration a *bunch*.

For each $\theta \in \Theta$ and $v \in V$ the system model (1) is now used to run a large number of simulations of the robot's motion for randomly sampled inputs u_a and u_β in order to generate the motion primitives that constitute each $B(\theta, v)$. After each time interval Δt, which corresponds to the quantization of the t dimension, we assess the quantization error e_Q of the end point of the motion primitive. The quantization error e_Q is simply calculated as the normalized distance to the closest state in L_0. If e_Q is larger than a given threshold $e_{Q,\max}$ (e.g. 5 %), we continue simulating the robot's motion until a simulation horizon $t_{\max}$. If by then e_Q is still too large, we drop the motion primitive. If e_Q is less then $e_{Q,\max}$, we check whether another motion primitive exists in the bunch ending at an identical state. If this is the case, we score both motion primitives by a weighted sum of their length and e_Q in order to decide which one to keep.

The union of all bunches that are generated using the above-described procedure constitutes the high-dimensional motion primitive set M_0 that defines the valid state transitions for L_0.

$$M_0 = \bigcup_{\substack{\theta \in \Theta \\ v \in V}} B_0(\theta, v) \tag{3}$$

The index "0" indicates that the bunch/motion primitive set has not been projected to a lower dimensionality so far.

The presented approach is capable to generate and test several million motion primitives per second, which enables the algorithm to quickly generate a set of high-quality motion primitives with small quantization errors e_Q. Furthermore, additional constraints on the state variables (like a larger turning radius for higher velocities) can be easily integrated in our approach. However, for more complex systems a numerical integration of the system of differential equations might be necessary, but as all this is done off-line, computation time is not an issue. Furthermore, a simple stop criterion for this probabilistic sampling approach can be employed by using a performance measure like the RMS of the quantization error e_Q.

2.2 Repeated Projection of High-dimensional Lattice

The high-dimensional state lattice L_0 can guarantee a good planning quality of the robot motion, however, due to its high number of dimensions it is prone to the curse of dimensionality, which may result in a very poor planning speed. Therefore, we propose a novel concept of successive dimensionality reduction of the search space in order to gradually lower the planning fidelity with increasing time. For this purpose, we start by projecting the full-dimensional state $\times$ time lattice L_0 onto its $X \times Y \times \Theta \times V$ subspace in order to define a new state lattice

$$L_1 = X \times Y \times \Theta \times V \ . \tag{4}$$

This essentially means that the corresponding set of – still dynamically feasible – motion primitives M_1 is generated by projecting the set M_0 of motion

primitives corresponding to L_0 onto L_1. This is done for each motion primitive in each bunch separately. Although an arbitrarily chosen projection rule would be possible, we perform the projections by simply dropping the t dimension.

The projection process generally results in multiple motion primitives ending in an identical state. For these cases, we assign a cost to each of them using the same cost function as the subsequent graph search will use, and drop all motion primitives except the one with the least cost. Furthermore, the motion primitive which results from the "wait" action in M_0 (i.e, the robot does not move at all) is removed from M_1 because of its identical start and end state. Although the motion primitives of

$$M_1 = \bigcup_{\substack{\theta \in \Theta \\ v \in V}} B_1(\theta, v) \tag{5}$$

do not contain an explicit time component any more, the duration for executing a motion primitive is still stored to allow the graph search to incorporate this information into the cost function.

Transitions from L_0 to L_1 are defined by connecting all states $(x, y, \theta, v, t) \in L_0$, $t > t_0$ with the states reachable by the motion primitive $m \in M_1$ that starts at the corresponding projected state $(x, y, \theta, v) \in L_1$. The parameter t_0 determines for which time horizon dynamic obstacles should be considered during the planning. This threshold can be either a fixed value or computed adaptively depending on the robot motion and the prediction of the motion of the dynamic obstacles (cf. [7]).

In the following steps, the state lattices are repeatedly projected onto a subspace to successively reduce the order of the system of differential equations associated with the state lattice until it finally contains no more dynamic components but still maintains kinematic feasibility of the motion. For our given exemplary motion model this is already the case for one further projection of L_1, which contains (x, y, θ, v) states, onto the $X \times Y \times \Theta$ space, thus generating the state lattice L_2. The corresponding set of motion primitives M_2 is constructed in a similar manner. However, as the v dimension is dropped during the projection, the new bunches that constitute M_2 are only dependent on the starting θ.

$$M_2 = \bigcup_{\theta \in \Theta} B_2(\theta) \tag{6}$$

Again, appropriate transitions from L_1 to L_2 have to be defined, which are generated as needed during the graph search. For this purpose, each node stores the time of it being first visited during the search. If this time exceeds a given threshold t_1, a transition from L_1 to L_2 is inserted defined by the motion primitive $m \in M_2$ which starts at the projection of the corresponding state $(x, y, \theta, v) \in L_1$ onto $(x, y, \theta) \in L_2$.

At this stage, the maximum depth of projections is reached as a further projection (e.g. onto the $X \times Y$ sub-space) would discard the kinematic feasibility of the representable motion. Thus, for our example, $L_2 = L_{\max}$ follows.

Due to the repeated projections of the state ($\times$ time) spaces, it is obvious that the goal region for the subsequent graph search must be defined using the

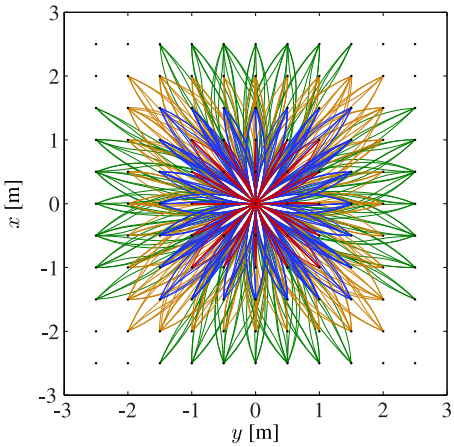

Fig. 1. Set of motion primitives $M_2 = M_{\max}$, $|M_2| = 812$, average outdegree: 17.3, average length: 2.02 m.

farthest projected state lattice $L_{\max}$. Consequently, the graph search can easily check an expanded node that belongs to any lattice $L_0, L_1, \ldots, L_{\max}$ by simply projecting its corresponding state onto $L_{\max}$ and testing if it lies in the specified goal region.

3 Graph Search

The search space is completely defined by the procedure described in the previous section. The corresponding graph is constructed during the search as needed using the motion primitive sets $M_0, M_1, \ldots M_{\max}$.

For finding the optimal trajectory/path through the sequence of lattices, standard algorithms for finding shortest paths in graphs can be employed. As on the one hand especially the first seconds of the search space are relatively high-dimensional but on the other hand a fast generation of (possibly sub-optimal) trajectories is required to allow a safe navigation in the presence of dynamic obstacles, anytime graph search algorithms are particularly well suited for this type of problem.

For our implementation of the presented algorithm, we chose to use the ARA* algorithm (Anytime Repairing A*, [14]). It first starts a weighted A* search to find a valid, but possibly ϵ-sub-optimal solution. For this purpose, it uses a heuristic, which is inflated by a factor ϵ, to determine the order of the node expansion. If a solution is found and there is still enough computation time left, the search starts again using a decreased inflation factor ϵ. This step may be repeated several times. To speed up planning, ARA* exploits intermediate results from the previous iteration. The integration of such an anytime graph search algorithm with our sequence of lattices is straightforward.

As especially in a highly dynamic environment a fast replanning is very important, it would be interesting to explore the potential of employing an explicit replanning algorithm like D* Lite [15]. However, this is a rather challenging task because all these replanning algorithms generally search backwards from the goal to the robot's current position to make use of their inherent advantages. This conflicts with our approach, which uses the accumulated time of a node since the start to determine the transition between two lattices. Of course, this time is not available when searching backwards.

4 Results

We implemented the individual components of the presented algorithm in C++ and evaluated them on an Intel® Xeon® E3-1270 CPU using both simulated and real data.

4.1 Construction of State (× Time) Lattices

The choice of an optimal quantization of the state × time lattice turned out to be a challenging task. On the one hand, a relatively fine discretization is desirable to enable planning of near optimal paths and to guarantee completeness of the search. On the other hand, the outdegree of each node increases with higher resolutions, which slows down the graph search because of the higher branching factor.

After a thorough look at the tradeoffs, we finally chose the following quantization. For the discretization of x and y we chose $\Delta x = \Delta y = 0.5\,\mathrm{m}$. To enable sufficiently smooth paths, we allow 16 discrete orientations, which corresponds to a heading resolution $\Delta\theta = 22.5°$. For the discretization of the velocity, we exploited the fact that the quantization may be done in a non-equidistant way, thus, we chose the set $\{-2\frac{\mathrm{m}}{\mathrm{s}}, 0\frac{\mathrm{m}}{\mathrm{s}}, 3\frac{\mathrm{m}}{\mathrm{s}}\}$ to represent the admissible discrete velocities. The duration of the time increment Δt is set to 0.5 s and the maximum duration $t_{\max}$ of a motion primitive is limited to $2\Delta t = 1\,\mathrm{s}$.

With this quantization setup, we are able to generate and test roughly 8 million motion primitives per second. This part of our algorithm profits enormously from parallelization as each motion primitive can be generated independently. In order to obtain a high-quality set of motion primitives, it is sufficient to run the generation process for one minute. The generated set M_0 consists of 1064 motion primitives with an average length of 1.96 m and an average outdegree of 22.6. M_1, which results from the projection of M_0 onto L_1, contains the same number of motion primitives as M_0 does. This implies that no end states of any two motion primitives from M_0 share the same (x, y, θ, v) components. This is due to the chosen quantization in conjunction with the short maximum motion primitive duration of $2\Delta t = 1\,\mathrm{s}$.

The subsequent projection of M_1 onto L_2 results in the expected reduction of the motion primitive count. A total of 814 motion primitives remains in $M_2 = M_{\max}$ (see Fig. 1). Consequently, the average outdegree decreases to 17.3, the average length increases slightly to 2.02 m.

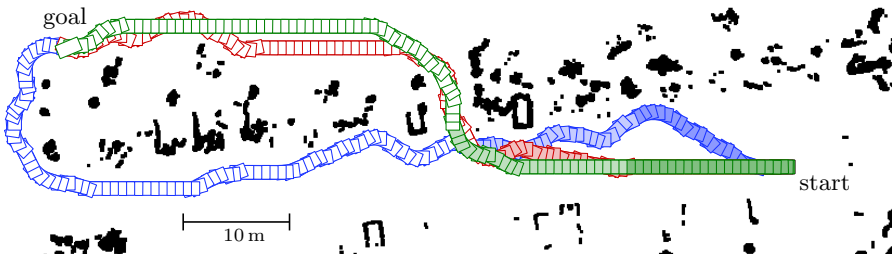

Fig. 2. Three iterations of the ARA* algorithm. First pass (blue) with $\epsilon = 2.0$ (computation time 6 ms), second pass (red) with $\epsilon = 1.3$ (computation time 47 ms), and third pass (green) with $\epsilon = 1.0$ (computation time 156 ms). The robots shape is shown every 0.25 s. The dark parts of the hybrid trajectory/path consist of states from L_0, i.e., they were planned in the $X \times Y \times \Theta \times V \times T$ space. The medium-dark patches belong to L_1 (i.e., the $X \times Y \times \Theta \times V$ space), and, finally, the light patches result from planning in $L_2 = L_{\max}$, i.e., the $X \times Y \times \Theta$ space.

4.2 ARA* Graph Search

The integration of the sequence of lattices with the ARA* algorithm is straightforward. As expected, the anytime nature of this algorithm is beneficial for this particular application. It has the ability to quickly find a hybrid trajectory/path to initiate an obstacle avoidance maneuver if it encounters a critical situation like an upcoming collision with a dynamic obstacle in the vicinity of the robot. However, this might come at the expense of the optimality of the solution. If there is some computation time left, the path can be successively improved. Fig. 2 shows three ARA* iterations for planning in an unstructured environment which has been mapped by laser scanners.

From Fig. 2 also our concept for planning in sequences of lattices with variable dimensionality becomes clear. The dark tiles of each solution represent the *trajectory* part of the solution, i.e., they consist of states from L_0, which explicitly contain a time dimension. This full-dimensional trajectory extends to the point $t_0 = 5\,\mathrm{s}$ in time for the shown example. The subsequent medium-dark tiles originate from planning in L_1, thus, still representing a dynamically feasible motion in the $X \times Y \times \Theta \times V$ space. At the time $t_1 = 10\,\mathrm{s}$ the solution transitions to a pure *path*, which comprises only (x, y, θ) states; however, it is still kinematically feasible (white tiles).

Overall, the dynamically feasible part of the solution extends to the time t_1, which results in planning a smooth trajectory in a relatively large vicinity of the robot. However, due to the quantization of the heading dimension, it might nonetheless be advisable to employ an additional trajectory smoothing for the very first seconds of the path.

To focus the graph search towards the goal, we used the following very simple heuristic.

$$h(x, y) = \|(x_{\mathrm{goal}}, y_{\mathrm{goal}}) - (x, y)\| \left(1 + \frac{\lambda_t}{v_{\max}}\right) \tag{7}$$

It is the sum of the Euclidean distance and the estimated time to reach the goal using the maximum admissible velocity. The ratio of these two components is controlled by the parameter λ_t.

To gain a sense for the computational effort of our algorithm, we simulated the path planning for the scenario depicted in Fig. 2 for different values of the transition time t_0. The time parameter t_1, i.e., the transition time from the dynamically feasible to the kinematically feasible phase, was set to $t_1 = 2t_0$. Fig. 3 shows the results. It can be seen, that according to the measured times, an optimal ($\epsilon = 1$) motion planning with 10 Hz is possible for transition times $t_0 \leq 3\,\mathrm{s}$ and $t_1 = 2t_0$.

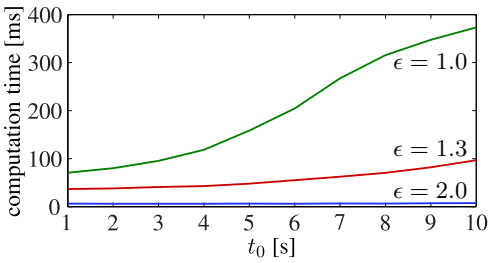

Fig. 3. Computation time for the trajectory/path planning of the scenario depicted in Fig. 2. The transition time t_1 is set to $2t_0$.

4.3 Dynamic Obstacles

The most interesting result is the behavior of our algorithm in the presence of dynamic obstacles because, after all, the safe operation in dynamic environments was our main motivation for the development of the presented algorithm on the basis of a search space with variable dimensionality. In the following, we will describe two exemplary scenarios for such a dynamic environment. The prediction of the dynamic obstacles is done using the methods described in [8], which model obstacles in a probabilistic way in order to incorporate a measure for the collision risk into the cost function during the graph search.

In the first scenario (see Fig. 4), the robot arrives at an intersection to its left, but it cannot turn left immediately because of an oncoming dynamic obstacle. Therefore, the robot plans an avoidance maneuver and positions itself right above the intersection to enter it as soon as the obstacle has passed. All this is possible because the robot explicitly considers time (and, thus, also the predicted position of dynamic obstacles) during the planning for the first t_0 seconds. In our implementation of the algorithm, we set this transition time to a fixed value of $t_0 = 5\,\mathrm{s}$, however, as already mentioned, it would be also possible to choose t_0 depending on the predicted motion of the dynamic obstacles by using

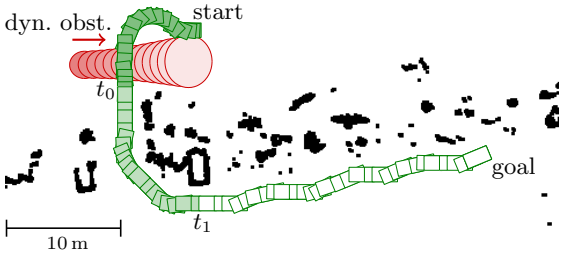

Fig. 4. Planning of a turning maneuver (green) with oncoming traffic (red). Again the dark-green tiles represent the *trajectory* part of the solution, the medium-dark tiles the dynamically feasible part, and the light tiles the kinematically feasible path. The color gradient of the dynamic obstacle directly encodes the time (lightness increases with time).

an approach similar to the one described in [7]. From this scenario, it is particularly apparent, that the local obstacle avoidance must not be decoupled from the global path planning. If this had been the case, the robot would probably have missed the intersection leading to an unnecessary detour and thus sub-optimal solution of the path planning problem. Also note, how – in addition to the initial full-dimensional trajectory till t_0 – planning a still dynamically feasible motion for the interval $[t_0, t_1]$ leads to high smoothness of the planned motion till t_1.

The second scenario (see Fig. 5) shows a more complex maneuver. Initially, the robot is oriented towards the goal on the left when it encounters an oncoming dynamic obstacle. As the passage is too narrow for both vehicles, the robot backs off, lets the obstacle pass, and then proceeds towards the goal. The whole maneuver is planned consistently only using the presented algorithm; no additional rules or logic are necessary. Nevertheless, it takes only 43 ms to compute this relatively complex maneuver (even the optimal solution with $\epsilon = 1$, and including the probabilistic prediction of the dynamic obstacle). This is only moderately more than planning a mere path (i.e., $t_0 = t_1 = 0$), which takes 12 ms; however, of course, the mere path would not be able to cope with the dynamic obstacle. On the other hand, planning a full-dimensional trajectory from the start to the goal, would take about 1500 ms, which is clearly infeasible for real-time applications. These benchmarks demonstrate the capability of our algorithm to significantly reduce the computation times while still guaranteeing high-quality solutions for the close future of the motion plan.

5 Conclusions

In this paper, we have presented a novel algorithm for the integrated global path planning and local dynamic obstacle avoidance. For this purpose, we used a state $\times$ time lattice with hybrid dimensionality. The algorithm can efficiently generate the required motion primitives using a probabilistic sampling strategy. We applied the ARA* algorithm to quickly find an initial solution for the

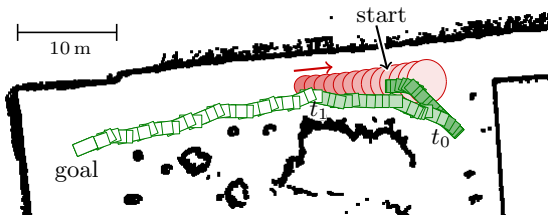

Fig. 5. Robot (green) with oncoming traffic (red). The robot backs off, lets the dynamic obstacle pass, and proceeds. Again the dark-green tiles represent the *trajectory* part of the solution, the medium-dark tiles the dynamically feasible part, and the light tiles the kinematically feasible path. The color gradient of the dynamic obstacle directly encodes the time (lightness increases with time).

planning problems in order to safely avoid collisions with moving objects. Our algorithm finds hybrid solutions consisting of a trajectory in the imminent future, a dynamically feasible path in the near future, and a kinematically feasible path for the remaining time to the goal. Even for long paths the computation time of our algorithm is quite moderate.

In future work, we will have a look at the practical application to systems with higher dimensionality and investigate the use of improved heuristics.

References

1. Dolgov, D., Thrun, S., Montemerlo, M., Diebel, J.: Path planning for autonomous vehicles in unknown semi-structured environments. The International Journal of Robotics Research **29**(5) (2010) 485–501
2. Pivtoraiko, M., Knepper, R.A., Kelly, A.: Differentially constrained mobile robot motion planning in state lattices. Journal of Field Robotics **26**(3) (2009) 308–333
3. Kavraki, L.E., Švestka, P., Latombe, J.C., Overmars, M.H.: Probabilistic roadmaps for path planning in high-dimensional configuration spaces. IEEE Transactions on Robotics and Automation **12**(4) (1996) 566–580
4. LaValle, S.M., Kuffner, Jr., J.J.: Randomized kinodynamic planning. The International Journal of Robotics Research **20**(5) (2001) 387–400
5. Likhachev, M., Ferguson, D.: Planning long dynamically feasible maneuvers for autonomous vehicles. The International Journal of Robotics Research **28**(8) (2009) 933–945
6. Ziegler, J., Stiller, C.: Spatiotemporal state lattices for fast trajectory planning in dynamic on-road driving scenarios. In: Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems. (2009)
7. Kushleyev, A., Likhachev, M.: Time-bounded lattice for efficient planning in dynamic environments. In: Proceedings of the IEEE International Conference on Robotics and Automation. (2009)
8. Petereit, J., Emter, T., Frey, C.W.: Safe mobile robot motion planning for waypoint sequences in a dynamic environment. In: Proceedings of the IEEE International Conference on Information Technology. (2013)

9. Petereit, J., Emter, T., Frey, C.W.: Mobile robot motion planning in multi-resolution lattices with hybrid dimensionality. In: Proceedings of the IFAC Intelligent Autonomous Vehicles Symposium. (2013)
10. Gochev, K., Cohen, B., Butzke, J., Safonova, A., Likhachev, M.: Path planning with adaptive dimensionality. In: Proceedings of the Symposium on Combinatorial Search. (2011)
11. Bicchi, A., Marigo, A., Piccoli, B.: On the reachability of quantized control systems. IEEE Transactions on Automatic Control **47**(4) (2002) 546–563
12. Rufli, M., Siegwart, R.: On the design of deformable input- / state-lattice graphs. In: Proceedings of the IEEE International Conference on Robotics and Automation. (2010)
13. Kelly, A., Nagy, B.: Reactive nonholonomic trajectory generation via parametric optimal control. The International Journal of Robotics Research **22**(7–8) (2003) 583–601
14. Likhachev, M., Gordon, G., Thrun, S.: ARA*: Anytime A* search with provable bounds on sub-optimality. In: Proceedings of the Conference on Neural Information Processing Systems. (2003)
15. Koenig, S., Likhachev, M.: Fast replanning for navigation in unknown terrain. IEEE Transactions on Robotics and Automation **21**(3) (2005) 354–363