2009 IEEE International Conference on Robotics and Automation
Kobe International Conference Center
Kobe, Japan, May 12-17, 2009

Time-bounded Lattice for Efficient Planning in Dynamic Environments

Aleksandr Kushleyev
Electrical and Systems Engineering
University of Pennsylvania
Philadelphia, PA 19104
akushley@seas.upenn.edu

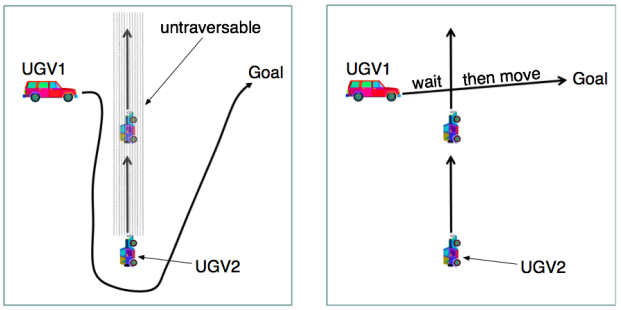
Maxim Likhachev
Computer and Information Science
University of Pennsylvania
Philadelphia, PA 19104
maximl@seas.upenn.edu

***Abstract*— For vehicles navigating initially unknown cluttered environments, current state-of-the-art planning algorithms are able to plan and re-plan dynamically-feasible paths efficiently and robustly. It is still a challenge, however, to deal well with the surroundings that are both cluttered and highly dynamic. Planning under these conditions is more difficult for two reasons. First, tracking and predicting the trajectories of moving objects (i.e., cars, humans) is very noisy. Second, the planning process is computationally more expensive because of the increased dimensionality of the state-space, with time as an additional variable. Moreover, re-planning needs to be invoked more often since the trajectories of moving obstacles need to be constantly re-estimated.**

In this paper, we develop a path planning algorithm that addresses these challenges. First, we choose a representation of dynamic obstacles that efficiently models their predicted trajectories and the uncertainty associated with the predictions. Second, to provide real-time guarantees on the performance of planning with dynamic obstacles, we propose to utilize a novel data structure for planning - a time-bounded lattice - that merges together short-term planning in time with long-term planning without time. We demonstrate the effectiveness of the approach in both simulations with up to 30 dynamic obstacles and on real robots.

I. INTRODUCTION

The most common implementations of path planning algorithms for unmanned ground vehicles (UGVs) utilize cost maps (or, in other words, 2D grid worlds) to represent the surrounding environment. This simple approach, however, lacks the appropriate framework for robustly dealing with dynamic obstacles. Consider, for example, a scenario depicted in Figure 1(a), where the vehicle UGV1 needs to reach its goal, but a dynamic obstacle, marked UGV2, poses a collision threat. It should be clear that the straight-line path from UGV1 to its desired location intersects the anticipated path of UGV2. A common solution to this planning problem is to augment a 2D cost map by marking the initial segment of the predicted obstacle's path (highlighted with stripes), as "untraversable" and treating the environment as if it were static. To UGV1, the obstacle would then appear as a wall, extending for some distance ahead of UGV2, forcing the resulting trajectory of UGV1 to be long and inefficient, as shown. A more optimal plan, on the other hand, would have been for UGV1 to quickly cut in front of UGV2 or simply wait for it to pass and then proceed, depending on the acceleration capabilities of UGV1 (Figure 1(b)). Most importantly, this approach can sometimes fail to find a solution, even if one exists.



(a) planning w/o time (b) planning with time

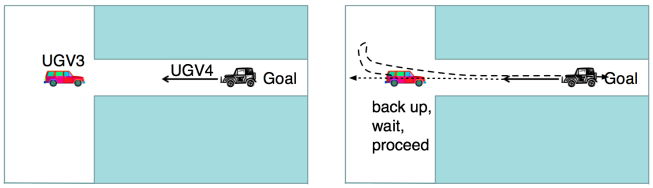
Fig. 1. Example illustrating the need for time-parameterized planning

For example, consider the scenario in Figure 2(a). In this case, a collision-free path to the goal does not exist in 2D, and a planner, incapable of finding a time-parameterized trajectory, would not be able to generate a feasible solution for UGV3. This may result in a collision or a deadlock, depending on the behavior of UGV4. In contrast, a more complex planner would make UGV3 back up, let the dynamic obstacle pass, and then proceed to the goal (Figure 2(b)).

There are several major challenges in computing collision-free time-parameterized paths such as the ones shown in Figures 1(b) and 2(b). First, in most real-world scenarios, it is nearly impossible to estimate the trajectories of dynamic obstacles with high certainty. In addition to sensor accuracy limitations, the quality of such estimates depends greatly on the size, speed, and distance of these entities from the main vehicle. To account for this uncertainty in the predicted motion of the dynamic obstacles, we propose to use an extended representation of space-time trajectories. In particular, they can be modeled as time-parameterized sequences of two-dimensional Gaussian distributions in space. The covariance matrices at each time step are computed based on the uncertainty of the past measurements as well as the uncertainty in the future actions of the dynamic obstacles.

The second challenge is that the actual search for a collision-free time-parameterized path is computationally expensive. Let us formally define a time-parameterized trajectory as an ordered set of points: $X_0, \ldots, X_n$. Each point X_i is at least a three-dimensional point (x, y, t), where x, y is the position of the center (or some other reference point) of the vehicle at time t. Most of the unmanned vehicles however, are non-holonomic and have inertial constraints. Consequently, in order to reason effectively about collisions,

Authorized licensed use limited to: Fondren Library Rice University. Downloaded on April 21,2022 at 13:07:54 UTC from IEEE Xplore. Restrictions apply.



(a) planning w/o time (b) planning with time

Fig. 2. Example illustrating the need for time-parameterized planning

each point X_i on the path needs to incorporate additional variables such as orientation θ, translational velocity v, and rotational velocity w, making X_i a six-dimensional point. It is hard to guarantee good quality real-time performance when planning six-dimensional trajectories. In particular, time parameterization causes the state-space to grow without bound.

The algorithm presented in this paper addresses these issues. In particular, when planning in dynamic and uncertain environments, it often does not make sense to rely on the predicted obstacle behavior too far into the future - the uncertainty becomes too great for the estimates to be of any use. Based on this observation, we propose to use a novel graph for planning - a time-bounded lattice - which merges together dynamically-feasible six-dimensional planning in time (x, y, θ, v, w, t) and fast kinematic planning in (x, y). The algorithm adaptively controls the extent of the six-dimensional planning based on the uncertainty in the obstacle behavior. The resulting implementation of our planner is capable of generating real-time trajectories that go all the way to the goal but, at the same time, avoid dynamic obstacles that can possibly collide within a short period of time. The success of the planner is demonstrated in both simulations and on a hardware platform. The experiments show that the planner can consistently re-plan paths within tens of milliseconds in the environments filled with up to 30 dynamic obstacles.

II. RELATED WORK

Most of the approaches to dealing with moving obstacles model them as static obstacles with a short window of high cost around the beginning of their projected trajectories [5]. While efficient, these approaches suffer from potential high suboptimality and even incompleteness as described in section I.

There has also been quite a bit of work on planning *time-parameterized* paths in dynamic environments. The work can be grouped along different dimensions. For example, some approaches assume completely known trajectories of moving objects [1], [12], [2], while others try to model the uncertainty in the future trajectories of obstacles [4], [9], [13]. Our proposed approach also models the uncertainty but is not restricted to a particular noise model and can even accommodate multiple predictions of the trajectories of dynamic obstacles.

Planning with time, required for dealing with dynamic obstacles, is hard to perform on-line since constant demand for re-planning enforces tight constraints on the duration of execution cycle. To address the real-time constraints, a number of approaches have been proposed that sacrifice near-optimality guarantees for the sake of efficiency [3], [10], [12]. Our approach differs in that we aim for computing and re-computing paths that are optimal or nearly-optimal. To achieve this, we propose a time-bounded lattice data structure and search it for a solution with a provable suboptimality.

III. ALGORITHM

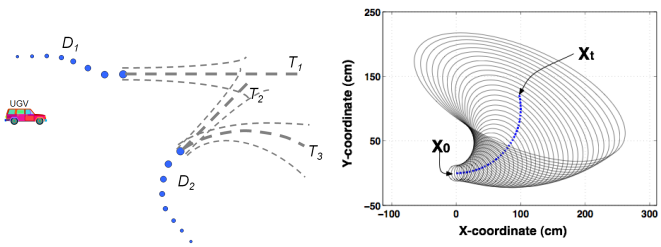
At the beginning of each planning cycle, our algorithm estimates the representation of dynamic obstacles by computing their time-parameterized trajectories as a series of Gaussian distributions evolving through time. This is done using the latest obstacle position and velocity information, extracted by a perception module before planning is done.

During a planning cycle, our algorithm constructs a graph and searches it for a collision-free path (the construction of the graph is interleaved with the search itself so as to avoid the construction of the whole graph). The graph is based on a time-bounded lattice, a structure with two different types of states: six-dimensional (x, y, θ, v, w, t) states, which are used to create the time-parameterized portion of the trajectory, and two-dimensional (x, y) states for the fast 2D search. Each transition in this graph is a short-term motion/path between the corresponding pair of these states. The associated transition costs may incorporate a variety of optimization criteria - in our experiments, we optimize for expected travel time. Since we have both static and moving obstacles, both types contribute to the transition cost: the static map is used to simply look up costs of traversing particular 2D cells; for the dynamic obstacles, high cost is assigned to transitions that are expected to collide with moving obstacles. The following sections provide a more detailed description for each of these steps.

A. Representation of Dynamic Obstacles

Our representation of any particular moving obstacle is a small set of predicted time-parameterized trajectories, each associated with a confidence and a continuous uncertainty distribution. Figure 3(a) shows an example. In this figure, circular dots correspond to the past locations of obstacles D_1 and D_2, as observed by the robot (marked as UGV). The thick dashed curves represent the predicted trajectories. Each trajectory comes with the continuous uncertainty associated with it, illustrated roughly with thin dashed curves. This uncertainty reflects the fact that past sensory information about the dynamic obstacles is noisy and therefore its motion cannot be predicted accurately. Any dynamic obstacle may potentially have more than one estimated trajectory. For example, in Figure 3(a), D_2 has two: T_2 and T_3. T_2 represents the obstacle's motion in the direction of its current heading and T_3 corresponds to the possible continuation of the turn. Such multiple alternatives must each have expected probabilities of occurrence (confidence), which sum up to one.

Authorized licensed use limited to: Fondren Library Rice University. Downloaded on April 21,2022 at 13:07:54 UTC from IEEE Xplore. Restrictions apply.



(a) multiple dynamic obstacles (b) trajectory estimate

Fig. 3. Representation of dynamic obstacles. (a) gives an artificial example illustrating how dynamic obstacles (D_1 and D_2) are represented. Note how D_2 has two estimated trajectories. (b) gives a specific trajectory estimate with 95% error ellipses corresponding to Gaussian distributions for each time step of the estimated trajectory.

To compute the trajectory together with its uncertainty for some dynamic obstacle D_i, we first assume that perception or some other module on the robot first estimates (a) the current state of the D_i given by the state X_0^i consisting of the expected position (x_0^i, y_0^i) and orientation θ_0^i together with the 3x3 covariance matrix Σ_0^i and (b) its current controls given by linear and (optionally) angular velocities (v^i, w^i) and the variances V_v^i and V_w^i of each. This form of an estimate is general and can incorporate various assumptions the perception module makes. The source of these initial estimates could be any on-board sensors or a remote localization system.

We can then forward simulate the motion of the dynamic obstacle by feeding the estimate of its initial state X_0^i, its uncertainty Σ_0^i and controls v^i, w^i into the prediction step of Extended Kalman Filter [11]. Using some obstacle dynamics function $g(X, v, w)$, the standard equations are as follows:

$$\hat{X}_t^i = \begin{pmatrix} \hat{x}_t^i \\ \hat{y}_t^i \\ \hat{\theta}_t^i \end{pmatrix} = g(\hat{X}_{t-1}^i, v^i, w^i),\ \Sigma_t^i = G_t^i \Sigma_{t-1}^i (G_t^i)^T + R_t^i$$

$$\text{where } G_t^i = \frac{\partial g(\hat{X}_{t-1}^i, v^i, w^i)}{\partial \hat{X}_{t-1}^i},\ R_t = V_t^i M_{t-1}^i (V_t^i)^T,$$

$$\text{and } V_t^i = \frac{\partial g(\hat{X}_{t-1}^i, v^i, w^i)}{\partial (v^i, w^i)},\ M_{t-1}^i = \begin{pmatrix} V_v^i & 0 \\ 0 & V_w^i \end{pmatrix}$$

The resulting distribution $p(X_t^i) = \mathcal{N}(X_t^i; \hat{X}_t^i, \Sigma_t^i)$ is an estimate of future obstacle position and uncertainty as a function of time. The assumption, of course, is that the obstacle maintains constant controls at all future times. Our representation, however, allows for more than one estimate of the dynamic obstacle trajectory. Thus, it can model different control values (i.e., stopping a turn by setting $w^i = 0$ as well as continuing a turn as was shown in Figure 3(a)).

Figure 3(b) shows a sample obstacle trajectory estimate along with 95% error ellipses, representing the uncertainty in the distribution at each future time step. This uncertainty will only grow with time and, at some point, the distribution will be so wide, that the probability of a robot colliding with the dynamic obstacle D_i can be considered negligible. This property is used to find a bounding time T_b^i, after which, the planner can ignore the obstacle completely because its position distribution will be too spread out in space. This allows the planner to perform a 2D search (i.e., without time) as soon as all obstacles can be ignored.

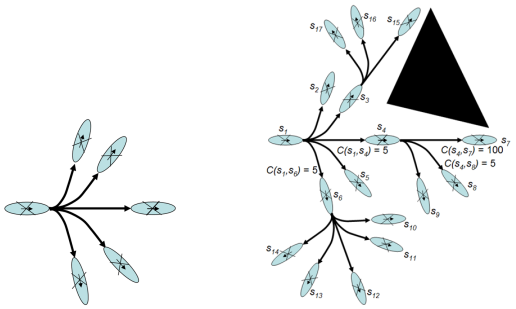
For each dynamic obstacle D_i, the value T_b^i needs to be computed once before the each planning episode (i.e., before each re-planning), as follows. We iterate over the trajectory T_i. Since for each time step, it is represented as a unimodal Gaussian, the most likely position of the obstacle is at the mean of the distribution. Thus, the highest probability mass of the robot colliding with obstacle D_i at time t is when the robot is placed at the mean of the distribution that corresponds to time t. This probability is computed by integrating the Gaussian distribution of the obstacle position at time t over a window centered at its mean. The size of the window is given by the sum of the radii of the robot and obstacle D_i. The time, when this probability drops below a certain small threshold, for example, 1%, can be used as the value for T_b^i.

B. Time-bounded Lattice

Once T_b^i are computed for all dynamic obstacles D_i, the planner can compute a single bound for how long the planning should be done in time: $T_b^{max} = \max_i T_b^i$. The bound may also be limited from above by a hard-coded limit on maximum planning in time T_b^{MAX}. The computed bound T_b^{max} is then used to construct a graph G_{tbL}, called time-bounded lattice, which is a combination of a lattice-based graph G_L and an eight-connected grid G_{2D}. In a lattice-based graph G_L all states are six-dimensional: (x, y, θ, v, w, t) and the costs of the transitions in between these states take into account dynamic obstacles. In the eight-connected grid G_{2D} each state is represented by just x, y, and the costs of transitions in between neighboring cells take into account only static obstacles.

Our version of the lattice-based graph G_L is an extension of lattice-based graphs used by planners to produce paths that can be executed smoothly and at a high speed [5]. Figure 4 shows a simple example of how graph G_L is constructed. It is built from a database of very short dynamically-feasible motion segments (a.k.a. primitives). A set of motion primitives is shown in Figure 4(a). Each of these motion primitives corresponds to a short-term (e.g., 100 msecs) action that moves the vehicle according to some pre-defined sequence of controls (for example, a constant acceleration and constant rotational velocity). As a result, each action moves the vehicle from its initial state (x, y, θ, v, w, t) into a new state. These states are shown as ovals in the figure. Figure 4(b) shows how these motion primitives are being stitched together to construct a full lattice-based graph G_L. An edge in the graph G_L corresponds to a motion primitive that connects the corresponding poses. Also, any edge (motion primitive) that intersects a known static obstacle is pruned. This can be seen in Figure 4(b), where no edges of the graph intersect the obstacle shown as a black polygon. Thus, any path in the graph G_L is a collision-free dynamically-feasible path.

Authorized licensed use limited to: Fondren Library Rice University. Downloaded on April 21,2022 at 13:07:54 UTC from IEEE Xplore. Restrictions apply.



(a) subset of motion primitives (b) lattice-based graph

Fig. 4. The construction of a lattice graph G. The planner constructs this graph on the fly as needed by the search and finds a path in it that corresponds to a dynamically feasible path that minimizes costs

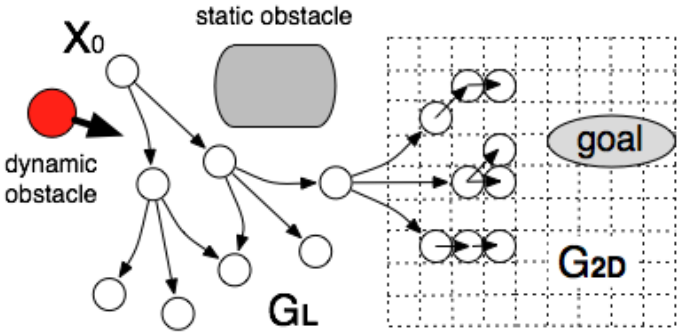


Fig. 5. Time-bounded lattice

The graph G_L is grown online on as-needed basis during the search (which is explained in the next section).

The task of the planner however, is not to find any feasible path. Instead, it should find a path that minimizes the sum of the costs of the edges that make up this path. Thus, as shown in Figure 4(b) there is a cost $c(s, s')$, associated with each edge, which is proportional to the following quantities: time it takes to traverse the corresponding motion primitive, proximity to known static obstacles, and the probability of colliding with any of the dynamic obstacles. The cost based on static obstacles is computed in a standard fashion: it is a simple look-up of a single location in an expanded 2D map. We assume that the robot is circular and expand all the static obstacles by its radius for efficient collision detection.

The dynamic collision cost is computed in a different way. In order to prevent plans that "jump over" obstacles when motion primitives have a long duration, the transitions must be broken up into n smaller segments. There will be a total of n poses, associated with the transition, and we will run collision checks on each one. The exact value for n is not critical and may be even chosen adaptively, based on the current relative speeds of the robot and obstacles. The main idea, though, is to have the collision checks done frequently enough, so that there is no chance of an obstacle intersecting robot's path without being detected.

The actual collision detection is performed at each of n time steps of duration dt, for each obstacle, by integrating its instantaneous 2D Gaussian position pdf over the expanded area of the robot (the size of the robot is expanded and circular dynamic obstacles become point objects for efficiency). This means that we need to compute the trajectory of each obstacle at dt resolution in time. Thus, if we have k obstacles, $n \cdot k$ collision checks will be performed for each transition. In order to combine these results into a single probability, $P(col)$, we assume that the events of not colliding are independent. Then,

$$P(col) = 1 - P(\overline{col}) = 1 - \prod_{i=0}^{k} \prod_{j=0}^{n} (1 - P(col)_j^i)$$

where $P(col)_j^i$ is the probability of colliding with ith obstacle at time $j \cdot dt$ from the start of the transition. In addition, since the 2D integral of a Gaussian distribution over an arbitrary area does not have a closed form solution, the approximations can be pre-computed for a finite number of shapes, relative locations and orientations and used while planning, which is much faster than doing the calculations online. In other words, the value of $P(col)_j^i$ is looked up in a table. Finally, the dynamic collision cost of a transition is computed as the cost of a collision times $P(col)$.

The construction of the graph G_{tbL} uses graph G_L only until the time associated with a state is below T_b^{max}. As soon as state s has a variable time $t >= T_b^{max}$, the state s is projected onto a 2D grid (graph G_{2D}) and starts following the grid transitions. This is shown in figure 5. The costs of transitions in the gridworld are proportional to their lengths and can also incorporate other costs associated with each cell in the grid (i.e., traversability, risk, etc.).

Since the time bound T_b^{max} is re-evaluated online based on the certainty in the dynamic obstacle trajectory predictions, graph G_{tbL} dynamically adapts. For example, if the environment has no dynamic obstacles, then graph G_{tbL} automatically reduces to a 2D gridworld.

C. Searching Time-bounded Lattice

To find a good quality path in the constructed time-bounded lattice, we use weighted A* search with an additional restriction that no state is expanded more than once. Weighted A* search is A* search with inflated heuristics (actual heuristic values are multiplied by an inflation factor $\epsilon > 1$). It proves to be fast for many domains and, in particular, for robot navigation tasks [5]. It also provides a bound on the sub-optimality, namely, the ϵ by which the heuristics are inflated. The same bound on sub-optimality continues to hold when we introduce the additional restriction that no state is expanded more than once [7] (without the restriction, weighted A* can re-expand the same state many times).

The heuristics are estimates of the cost-to-goal. In order for weighted A* to provide suboptimality guarantees, the heuristics (before inflation) must be consistent (i.e., satisfy a triangle inequality). That is, for any state $s \in G_{tbL}$, $h(s) \leq c(s, s') + h(s')$ for any successor s' of s if $s \neq s_{\text{goal}}$ and $h(s) = 0$ if $s = s_{\text{goal}}$. Here, $c(s, s')$ denotes the cost of a transition from s to s' and has to be positive. At the same time, the heuristics need to be as informative as possible

Authorized licensed use limited to: Fondren Library Rice University. Downloaded on April 21,2022 at 13:07:54 UTC from IEEE Xplore. Restrictions apply.

in order to guide the search well. To obtain such heuristic function, we run online a 2D Dijkstra's search to compute costs-to-goal for each cell in the gridworld taking into account static obstacles (similarly to how it is done in [5]). The search takes few tens of msecs and can therefore be invoked every time the map is updated. It can be shown that the resulting heuristic function is consistent, and weighted A* can therefore provide ϵ bound on the suboptimality of its solution.

IV. EXPERIMENTAL ANALYSIS

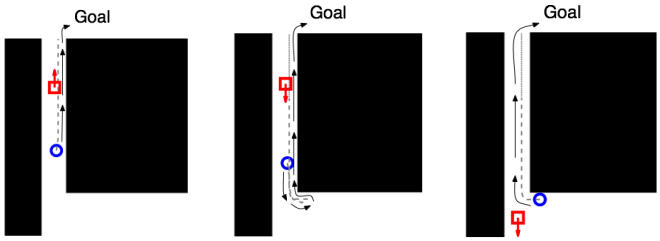
A. Experimental Setup

We have implemented and tested our planner both in simulation and on real robots. The goal was to show that actual planning times are short and allow for responsive behavior in dynamic environments. Two scenarios were simulated, one of which was set up as a real experiment. Gazebo and Player software was used as primary simulation and robot interfacing tools. The platform that we have chosen for testing was a custom built differential drive wheeled robot with velocity control. Its circular body, 30cm in diameter, has been previously modeled in Gazebo along with appropriate dynamics [8] - therefore the expectation was to see little difference in between the simulation and actual runs.

A single robot, running our planner, was tested in environments with varying static and dynamic obstacles. The role of the latter was fulfilled by other robots of the same type, controlled either by a human or a computer, depending on the specific experiment. The appropriate static obstacle map was provided to all robots before the start of each experiment. In simulation, the poses of the dynamic obstacles as well as the pose of the robot itself, were read directly from the simulator and Gaussian noise was added to simulate the sensor uncertainty. In real experiments, an overhead tracking system, consisting of an array of monocular cameras, provided approximate robot locations [8]. Obstacle velocity and heading were estimated from pose changes over time with an additional assumption that the expected obstacle paths were straight ($w = 0$).

Setup of experiment I The first simple experiment was set up specifically to demonstrate the advantage of planning in time - it demonstrated the collision avoidance and following an obstacle if the obstacle was moving in the direction of the goal. The idea was for the robot to drive from start to goal position with only a narrow corridor connecting the two points. The diagram of the environment is shown in Figure 6. The square with an arrow represents a dynamic obstacle and the circle is the robot. The dynamic obstacle robot, controlled by a human, is temporarily blocking the corridor and moving in different directions (toward or away from the goal), while the robot has to decide whether to follow the dynamic obstacle towards the goal or back up and get out of its way in order to avoid a collision.

Setup of experiment II The second experiment was set up in a virtual 15x15m environment with 20 static obstacles (circular and rectangular) and 30 dynamic obstacles. The latter were assigned random individual goals, used standard A*



(a) robot goes to goal (b) robot backs up (b) robot goes to goal

Fig. 6. Generated plans for $T_b^{MAX} = 20$ secs. Dashed curves are six-dimensional portions of the planned trajectories, while dotted curves are 2-dimensional portions.

for planning 2D paths and simple state feedback linearization controllers for tracking these 2D trajectories. The obstacle map for this experiment is shown in Figure 7. The goal of this experiment was to see how effectively the planner could control the robot towards the goals, while keeping track of 30 dynamic obstacles. The simulation ran for 30 minutes continuously, with goals randomly reassigned upon successful arrival at the destination.

B. Simulation Results

Experiment I To better see the planned trajectory of the robot when planning with time, we first ran our planner on experiment I using $T_b^{MAX} = 20$ secs and assuming high certainty in the trajectories of the dynamic obstacles. The generated plans are shown in Figure 6(a-c). Figure 6(a) shows that in case where the dynamic obstacle is moving towards the goal, the plan generated by the robot makes it follow the obstacle and expects to reach the goal in under 20 seconds (dashed lines represent the time-parameterized portion of the trajectory). Figures 6(b) and (c) show the plan generated in cases the obstacle is moving away from the goal. The robot first chooses to back up and let the obstacle pass through and then re-enters the corridor. 20 seconds is no longer enough to reach the goal and therefore the planned trajectory involves a 2D trajectory, which completes the path (shown with a dotted line).

These results are just a proof of concept, since planning with $T_b^{MAX} = 20$ secs was too expensive to compute - it took several seconds, which is not acceptable for dynamic control. In practice, robust collision avoidance behavior is achieved by limiting T_b^{MAX} to a lower case, 4 seconds in our case. This bound allows the planner to re-generate its plans quickly and thus to react fast to the most recent changes in the environment. The series of screenshots shown in Figure 10 (top row) show how the actual execution happens in simulation for the same environment but with $T_b^{MAX} = 4$ and some uncertainty about the position of the dynamic obstacles. The figures show that the sequence of events is nearly identical - the robot first follows the dynamic obstacle, and then (when the dynamic obstacle goes back) the robot turns around, backs up, lets the dynamic obstacle pass through and then proceeds towards the goal.

Experiment II In the second experiment we were interested in obtaining statistics for average planning times, number of completed goals and number of collisions with

Authorized licensed use limited to: Fondren Library Rice University. Downloaded on April 21,2022 at 13:07:54 UTC from IEEE Xplore. Restrictions apply.

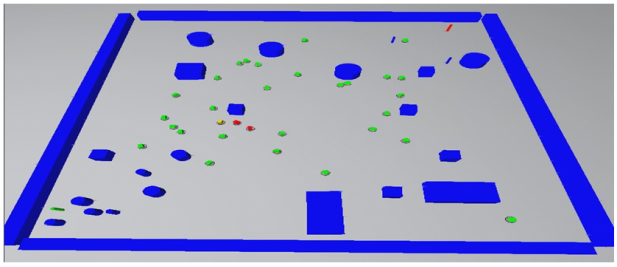


Fig. 7. Experiment II in simulation. A robot (shown in yellow) can navigate an environment full of dynamic obstacles (shown in green and red) and static obstacles (large blue obstacles). The dynamic obstacles, i.e., the potential collisions which are considered by the time-bounded lattice, are shown in red (the two next to the robot). The collisions with the rest of the dynamic obstacles are not considered by the time-bounded lattice. This figure should be viewed in color.

$t(secs)$	time-bounded lattice ($T_b^{MAX} = 4$ secs)	full 6D
0-0.5	99.19%	97.6%
0.5-1	0.67%	0.8%
1-2	0.11%	0.4%
2-5	0.03%	0.5%
5-10	0%	0.2%
10+	0%	0.5%

Fig. 8. Planning time distributions for time-bounded lattice and a full six-dimensional planning with time

dynamic obstacles. This experiment also served as a means of comparing our time-bounded lattice approach (with $T_b^{MAX} = 4$ secs) to a full 6D planning in time.

In case of planning with time-bounded lattice, the robot was able to achieve 69 goals with the mean planning time of only 34 msecs (or 125 state expansions). The total number of collisions during the experiment was equal to 12. Some of these collisions were unavoidable, since the obstacle robots did not try to avoid collisions.

In case of full six-dimensional planning, only 25 goals were achieved, with the mean planning time of 230 msecs (or 742 state expansions). Even though the average planning times are still acceptable, a closer look at the plan times reveals that the fast planning times result from scenarios, which agree closely with the pre-computed (2D Dijkstra's) heuristics. If the dynamic obstacles cause the optimal path to shift away from the heuristics, on the other hand, then planning times increase dramatically in the six-dimensional search. The consequence of this is that the robot essentially looses the ability to avoid obstacles due to large delays in planning. The table in Figure 8 demonstrates this by showing that the proportion of planning times for a full six-dimensional planning that exceeded 5 and even 10 seconds was too high.

C. Real Robot Experiments

The first experiment was also set up on the real robot platforms in order to make sure that our approach is robust enough to handle the extra uncertainty, not modeled by the

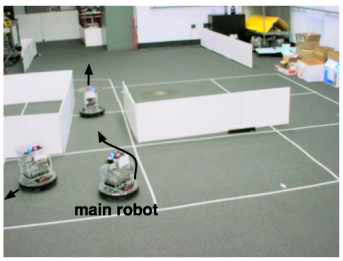


Fig. 9. Real robot experiment with two dynamic obstacles.

simulation. We ran this experiment with 1 dynamic obstacle (Figure 10(bottom row)), same as in simulation, and we also ran this experiment with 2 dynamic obstacles (Figure 9). In both cases, the planner was able to react fast and avoid collisions. The behavior in the first case was very similar to the one in simulation - the robot was able to react promptly to the changing direction of motion of the dynamic obstacle and to re-plan accordingly. The series of screenshots shown in Figure 10(bottom row) show that the robot first follows the dynamic obstacle, then turns back as soon as the dynamic obstacle turns back, backs up to let the dynamic obstacle go through, and finally proceeds to its goal. Similar behavior of the robot was observed with 2 dynamic obstacles (a movie demonstrating this behavior can be found at `http://www.seas.upenn.edu/~akushley/movies/TimeBoundedLattice.mp4`).

V. DISCUSSION

The time-bounded lattice we proposed limits the amount of planning in 6D (with time) based on the uncertainty in the predictions of the dynamic obstacle trajectories. In addition, to guarantee the real-time performance, it uses the hard threshold T_b^{MAX} as the maximum possible amount of planning. This hard threshold makes the planner "not see" a dynamic obstacle if it is beyond T_b^{MAX} secs (based on the velocity with which the robot and the obstacle approach each other) and "see" it when it gets closer. This phenomenon makes the robot actively avoid the dynamic obstacle only when it is within T_b^{MAX} seconds. This implies that T_b^{MAX} must be large enough for the robot to have enough time to react. Otherwise it may fail to avoid collisions due to its dynamic constraints.

The parameter T_b^{MAX} does not make the robot oscillate (or get stuck) unless the dynamic obstacle remains stationary (or moves back and forth all the time). If the dynamic obstacle does remain stationary, then the robot may get stuck in a hallway-like environment. In such environment, the robot would get to the obstacle within T_b^{MAX} secs, and would remain there forever, waiting for the obstacle to move. This is because the time-parameterized path would tell the robot not to move and wait until T_b^{MAX} expires, at which point all dynamic obstacles "disappear" and a 2D path can cut through the dynamic obstacles. Our assumption is that all dynamic obstacles that are actually stationary can (and should) be resolved at a higher level of the robot architecture. In particular, they can eventually be declared as static.

Authorized licensed use limited to: Fondren Library Rice University. Downloaded on April 21,2022 at 13:07:54 UTC from IEEE Xplore. Restrictions apply.

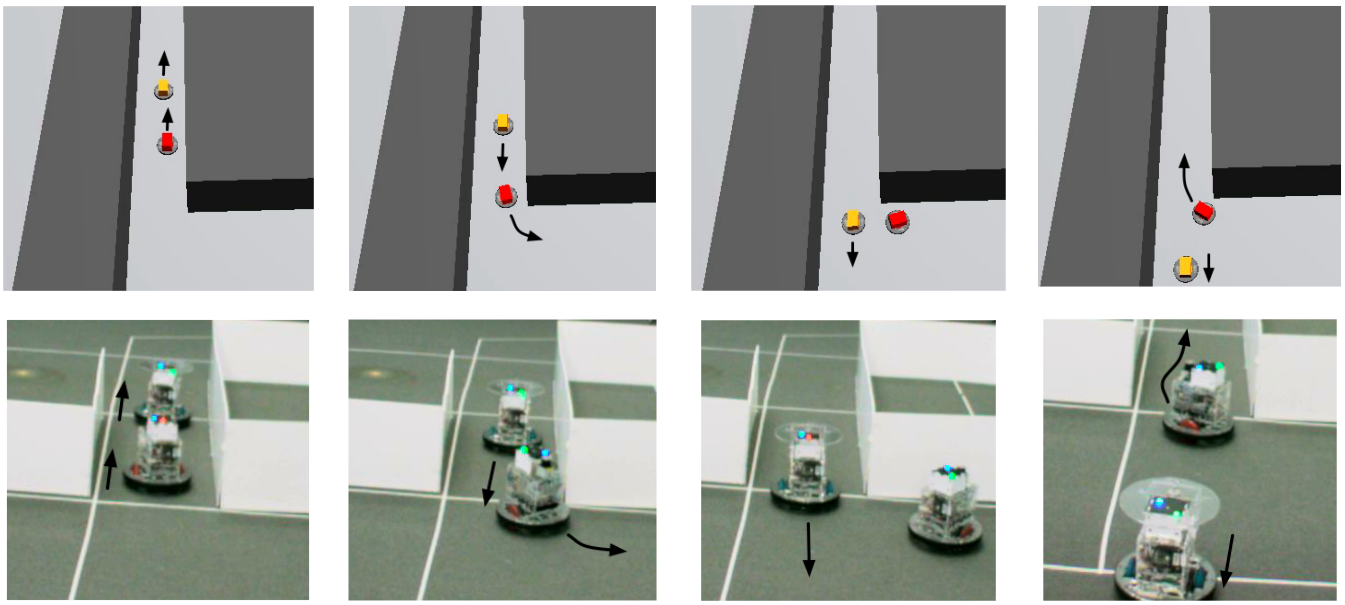


Fig. 10. Experiment I in simulation (top row) and on real robots (bottom row)

In the future, we intend on extending planning with time-bounded lattice graphs to incremental planning [6], which would allow us to speed up the planner when re-planning in response to map updates. In addition, we would like to find a principled approach to resolving the case of a stationary dynamic obstacle explained above. We would like to find an approach that can always guarantee reaching the goal state in finite amount of time whenever an infinite time-horizon 6D planning can.

Also, the obstacle model we use can be improved. Currently, the trajectory prediction does not take into account the surrounding static map, which does have an effect on the possible actions. For example, in a narrow hallway, the obstacle cannot drive past the walls. This implies that its position uncertainty distribution should be adjusted accordingly to fit most of the probability density within the reachable area.

VI. CONCLUSIONS

In this paper we have introduced a novel graph structure called time-bounded lattice useful for planning with dynamic obstacles. This graph merges together short-term planning in time with long-term planning without time. The amount of planning with time is adjusted automatically based on the uncertainty in the prediction of dynamic obstacle trajectories as well as their presence. For example, when no dynamic obstacles are present or they are too far to be detected with reasonable certainty, the planning automatically reduces to 2D planning. We have demonstrated the effectiveness of our approach both in simulations with up to 30 dynamic obstacles and on real robots.

VII. ACKNOWLEDGMENTS

This work was in part supported by the DARPA grant SB082-030 "Path Planning in Dynamic Environments" (subcontracted through Dragonfly Pictures, Inc.). We would also like to thank Vijay Kumar for making his software/hardware infrastructure available to us as well as Nathan Michael and Jon Fink for providing assistance with the experiments.

REFERENCES

[1] P. Fiorini and Z. Shiller. Motion planning in dynamic environment using velocity obstacles. *International Journal of Robotics Research*, 17(7):760–772, 1998.

[2] K. Fujimura and H. Samet. Planning a time-minimal motion among moving obstacles. *ALGORITHMICA*, 10, 1993.

[3] D. Hsu, R. Kindel, J.-C. Latombe, and S. Rock. Randomized kinodynamic motion planning with moving obstacles. *International Journal of Robotics Research*, 21:233–255, 2002.

[4] A. Inoue, K. Inoue, and Y. Okawa. On-line motion planning of autonomous mobile robots to avoid multiple moving obstacles based on prediction of their future trajectories. *J. of Robotics Society of Japan*, 15(2):249–260, 1997.

[5] M. Likhachev and D. Ferguson. Planning long dynamically-feasible maneuvers for autonomous vehicles. In *Proceedings of Robotics: Science and Systems (RSS)*, 2008.

[6] M. Likhachev, Dave Ferguson, Geoff Gordon, Anthony Stentz, and Sebastian Thrun. Anytime search in dynamic graphs. *Artificial Intelligence Journal*, accepted for publication.

[7] M. Likhachev, G. Gordon, and S. Thrun. ARA*: Anytime A* with provable bounds on sub-optimality. In *Advances in Neural Information Processing Systems (NIPS) 16*. Cambridge, MA: MIT Press, 2003.

[8] N. Michael, J. Fink, and V. Kumar. Experimental testbed for large multi-robot teams: Verification and validation. *IEEE Robotics and Automation Magazine*, 15(1):53–61, March 2008.

[9] J. Minura, H. Uozumi, and Y. Shirai. Mobile robot motion planning considering the motion uncertainty of moving obstacles. In *Proceedings of IEEE Int. Conf. on Systems, Man, and Cybernetics*, pages 692–698, 1999.

[10] S. Petty and T. Fraichard. Safe motion planning in dynamic environments. In *Proceedings of IEEE Int. Conf. on Intelligent Robots and Systems (IROS)*, pages 3726–3731, 2005.

[11] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic Robotics (Intelligent Robotics and Autonomous Agents)*. The MIT Press, September 2005.

[12] J. van den Berg, D. Ferguson, and J. Kuffner. Anytime path planning and replanning in dynamic environments. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 2366–2371, 2006.

[13] J. van den Berg and M. Overmars. Planning the shortest safe path amidst unpredictably moving obstacles. In *Proceedings Workshop on Algorithmic Foundations of Robotics*, 2006.

Authorized licensed use limited to: Fondren Library Rice University. Downloaded on April 21,2022 at 13:07:54 UTC from IEEE Xplore. Restrictions apply.