2012 IEEE/RSJ International Conference on
Intelligent Robots and Systems
October 7-12, 2012. Vilamoura, Algarve, Portugal

Anytime Safe Interval Path Planning for Dynamic Environments

Venkatraman Narayanan, Mike Phillips, and Maxim Likhachev
Robotics Institute, Carnegie Mellon University, Pittsburgh, PA 15213

Abstract— **Path planning in dynamic environments is significantly more difficult than navigation in static spaces due to the increased dimensionality of the problem, as well as the importance of returning good paths under time constraints. Anytime planners are ideal for these types of problems as they find an initial solution quickly and then improve it as time allows. In this paper, we develop an anytime planner that builds off of *Safe Interval Path Planning* (SIPP), which is a fast A*-variant for planning in dynamic environments that uses intervals instead of timesteps to represent the time dimension of the problem. In addition, we introduce an optional time-horizon after which the planner drops time as a dimension. On the theoretical side, we show that in the absence of time-horizon our planner can provide guarantees on completeness as well as bounds on the sub-optimality of the solution with respect to the original space-time graph. We also provide simulation experiments for planning for a UAV among 50 dynamic obstacles, where we can provide safe paths for the next 15 seconds of execution within 0.05 seconds. Our results provide a strong evidence for our planner working under real-time constraints.**

I. INTRODUCTION

Whether it be autonomously driving a vehicle or flying a UAV, almost all robots assume the ability to safely navigate from place to place in the presence of moving objects such as people, pets, cars, etc. In order to accomplish this, robots need to be able to generate short collision-free paths with respect to where dynamic obstacles will be in the near future. Robots also need to be able to plan these paths very quickly, since predictions for dynamic obstacle trajectories constantly change. Providing good paths under hard time constraints is critical in dynamic environments.

Dynamic environments pose real time constraints on planning times. If a planner takes too long to return a new path, then a collision can occur with a moving obstacle. Anytime planners find an initial solution quickly and improve the solution as time allows. This type of planner is ideal for dynamic environments and this paper presents such an approach. Our anytime planner builds off of SIPP (Safe Interval Path Planning) which is a fast, optimal, A*-variant for planning in dynamic environments [10]. Planning with time as an explicit dimension in the state space is slow since there are an unbounded number of timesteps for each spatial location. SIPP compresses timesteps into indices of contiguous safe time intervals and therefore only has a single state for each safe time interval, for each spatial location. For example, if only one dynamic obstacle passes through a particular location, then there are only two safe intervals (before and after the obstacle passes) and therefore, only two states for that location. SIPP runs faster by having almost as few states as planning in a static environment.

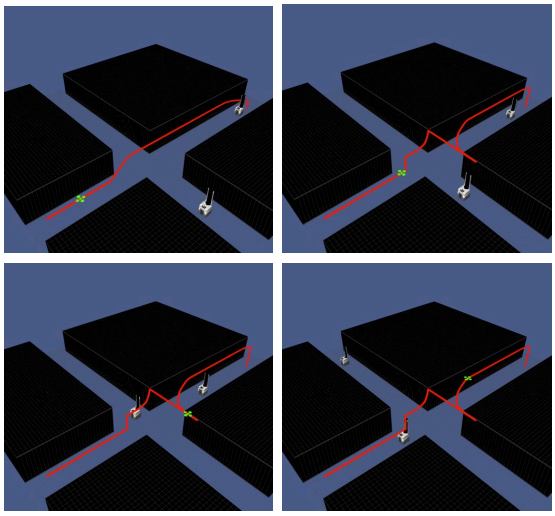

Fig. 1: An example scenario showing a quadrotor running Anytime SIPP to reach the other end of the corridor. Left to right, top to bottom: For a time horizon of 5s, the quadrotor initially sees only the ground robot in the adjacent corridor as a potential threat and plans a path (shown in red) around it. As the quadrotor executes its plan, the trajectory of the ground robot at the far end appears within the time horizon, and a new path is generated that accounts for both the dynamic obstacles.

In this paper, we extend SIPP to anytime search by combining ARA* (an anytime A* planner) and the ability to run weighted A* search with SIPP [7], [9]. On top of the anytime SIPP planner, we leverage the time horizon from the *Time-Bounded Lattice* idea [5]. According to it, the planner only plans with the time dimension for states whose time value is earlier than the time horizon, whereas states after the time horizon are planned for only with their spatial coordinates. Planning in this representation is significantly faster in large scale environments since the time dimension only exists near the robot temporally. However, it can only guarantee safety during execution up until the time horizon. In practice, this is reasonable since trajectory predictions tend to be less accurate the farther out they go anyway. Figure 1 illustrates the use of the time horizon.

In addition to presenting an anytime planner, we provide theoretical guarantees on completeness as well as a bound

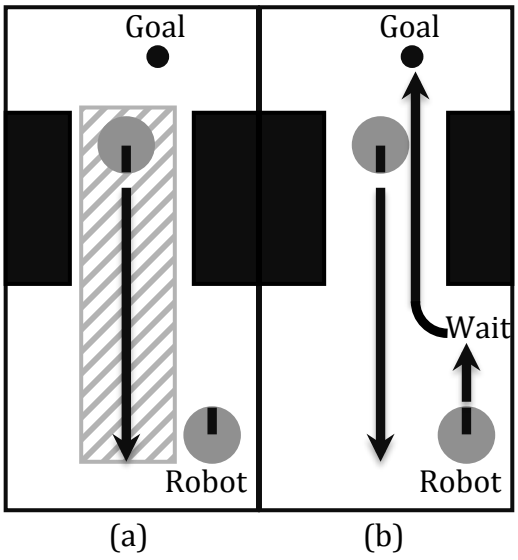

Fig. 2: (a) Treating the dynamic obstacle as a static one results in no solution, (b) Planning with time finds a solution by waiting for the obstacle to pass and then proceeding

on the sub-optimality of the solution cost. Experimentally, we demonstrate the real-time capabilities of the planner in a UAV domain. We plan time parameterized (x, y, z, θ) paths on large maps with 50 dynamic obstacles. We show that we can get initial solutions up to the time horizon in as little as 0.05s and given more time we are able to improve the solution toward optimal cost.

II. RELATED WORK

Most of the approaches to dealing with dynamic obstacles model them as static obstacles with a short window of high cost around the beginning of their projected trajectories [6], [13]. The approach is efficient since it doesn't have to consider the time dimension and it ensures that the plan generated will not be allowed to collide with the dynamic obstacles in the near future. However, this approach suffers from sub-optimality in cases where the robot could have crossed a trajectory without being hit, or just waited until the obstacle passed and then crossed. Instead it takes a long path around the trajectory of the obstacle. There are even cases when this approach will fail to find a solution at all, such as when a dynamic obstacle's trajectory goes through or crosses in front of a doorway that the robot must also use in order to reach its goal, as shown in Figure 2(a). On the other hand, Figure 2(b) shows that by considering the time dimension, the goal can be reached by waiting for the obstacle to pass.

Another common approach is to consider dynamic obstacles (still treating them as static obstacles) only while executing the path, using local obstacle avoidance methods [2]. This method can get stuck in local minima and is not globally optimal. Another alternative is for the local planner to use velocity obstacles, which determine the controls that lead to collision with moving obstacles [17]. While this is more accurate, it still can lead to local minima as it greedily minimizes the difference between the desired control without dynamic obstacles, and the set of feasible controls that are not in velocity obstacles.

Some approaches plan in the full space-time search space [14]. Silver's HCA* algorithm is designed for planning for multiple robots, but in the paper, he points out that it can be applied to planning in dynamic environments. This approach finds the optimal solution but even on problems with only a few dynamic obstacles, it can take much longer than real-time as shown in the experimental results of the SIPP paper.

The approach [16] is similar to SIPP in that it uses time intervals instead of timesteps. However, SIPP has been extended to weighted A* search which we will be exploiting in our algorithm to make an anytime search [9].

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 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], [15]. Our proposed approach differs in that we aim for computing paths that have bounds on the sub-optimality of the solution with respect to shortest time. Some other approaches use RRT-variants to plan quickly in higher-dimensional search spaces that can handle the kinodynamic constraints of more complex robots [8], [1]. However, these sampling-based approaches cannot provide the guarantees on optimality that we strive for. RRT* is a recent anytime sampling-based approach, which finds the optimal solution in the limit [4]. However, when the planning time expires, this algorithm cannot provide any information about how good its solution is. On the other hand, with each solution our anytime search finds, it has a numerical upper bound on how sub-optimal it is compared to the optimal solution.

III. ALGORITHM

A. Dynamic Obstacle Representation

Our algorithm assumes that there is another system that tracks dynamic obstacles in the environment, predicts their future trajectories, and formats them into a general representation we define. We are given a list of dynamic obstacles, where each obstacle, treated as a sphere, has a radius and a trajectory. A trajectory is a list of points, where each point has state variables, specifying its configuration and time. The points in the trajectory list are ordered from earliest time to latest time, so by reading the points in order, it can be seen how the obstacle is predicted to move in the near future.

B. Notations and Assumptions

We will now introduce some notation used to explain the algorithm. We assume that the planning problem is represented with a graph. As such, each state s in the graph is defined by two elements: a spatial configuration vector of the robot denoted by $x(s)$ (for example, (x, y, z, θ)) and a scalar safe time interval index denoted by $interval(s)$.

Indices of contiguous safe time intervals represent the indices of time intervals during which a particular location is free of dynamic obstacles. For example, if only one dynamic obstacle passes through a particular location $x(s)$, then there are only two safe intervals (before and after the obstacle passes). This means that there are only two states that have $x(s)$, the state with $interval(s) = 0$ and the state with $interval(s) = 1$. This is substantially less than having as many copies of states for a given $x(s)$ as there are timesteps. The edges between the states are short motions connecting the corresponding configurations. Depending on the representation of the configuration space, these could be kinodynamically-feasible motion primitives [11], [6] or simple segments of curves with constant curvature.

During the search, each state s has a variable, $g(s)$, which is the cost of the best known path from the start state to s. The heuristic function $h(s)$ is an estimate of the cost from s to the goal state. We will be assuming that the heuristic function is consistent, meaning that it never overestimates the cost to the goal and it satisfies the triangle inequality. The cost of a transition or edge from s to one of its successors s' is defined by $c(s, s')$.

Our algorithm makes the same assumptions as SIPP:

- $c(s, s')$ = time to execute the action from s to s'. In other words, the goal of the planner is to find a time-minimal trajectory.
- The robot is capable of waiting in place (this assumption would not be true of a motorcycle for example).
- Inertial constraints (acceleration/deceleration) are negligible. The planner assumes that the robot can stop and accelerate instantaneously.

C. Anytime SIPP

Our algorithm extends SIPP to anytime planning by combining it with ARA* (Anytime Repairing A*). ARA* performs anytime planning by running a series of weighted A* searches with decreasing values of ε [7]. Weighted A* works by inflating the heuristic by $\varepsilon > 1$. This causes the search to be more goal focused and in practice, it finds solutions significantly faster than A* [12]. Weighted A* no longer guarantees the optimal solution, but it has been shown that even without re-expanding states, the found solution has a cost no greater than ε times the optimal solution [7]. ARA* starts by running a weighted A* search with a high ε in order to find an initial solution very quickly. As time allows, ARA* then decreases the value of ε and re-runs weighted A* while reusing computation from previous searches. Given enough time, ARA* will reach $\varepsilon = 1$ and return the optimal solution.

Unfortunately, SIPP in its original form only works as an optimal planner. If $\varepsilon > 1$ there is no guaranteed sub-optimality bound and in fact it can be shown that it is not even complete. This is because SIPP works under the assumption that states are expanded at the earliest possible timestep for each location and each time interval in order to guarantee the maximum set of successors for the state. The sub-optimality of weighted search breaks this assumption. In order to overcome this, we leverage CFDA-A* (Cost Function Dependent Action A*), an extension to SIPP which introduces two instances of each state (an optimal and sub-optimal version) [9]. sub-optimal states are used when possible to produce the fast planning times expected of weighted A*, while just enough optimal states are expanded in order to maintain guarantees on completeness and bounds on the sub-optimality of the solution.

Now that SIPP has the guarantees on completeness and sub-optimality bounds, we can apply ARA* directly in order to get an anytime planner in dynamic environments using safe intervals.

The *Main* function (Algorithm 1) shows the main loop of ARA*. The lines 1-7 show how it runs the first search (*ImprovePath*) with the initial ε provided by the user. The loop from lines 8-16 decreases the value of ε (where the decrement can be chosen by the user) and reruns the search until the optimal solution is reached ($\varepsilon = 1$). During each search, the *INCONS* list is filled with states that a cheaper path was found for, but they had already been closed and therefore were not re-expanded. These states are inconsistent because the improvement in cost to these nodes have not been propagated to their successors and therefore, they need to be re-expanded in the next search iteration. Line 10 shows how these states are put into the *OPEN* list to have the chance to be expanded in the next search iteration. At lines 6,14 we compute ε' which gives us a sub-optimality bound on the currently found solution.

Algorithm 1

Main()

1: $g(s_{goal}) = \infty$
2: $g(s_{start}) = 0$
3: $OPEN = CLOSED = INCONS = \emptyset$
4: insert s_{start} into *OPEN* with $\varepsilon * h(s_{start})$
5: ImprovePath()
6: $\varepsilon' = \min(\varepsilon,$
$g(s_{goal}) / \min_{s \in OPEN \cup INCONS}(g(s) + h(s)))$
7: publish current ε'-sub-optimal solution
8: **while** $\varepsilon' > 1$ **do**
9: decrease ε
10: Move states from *INCONS* into *OPEN*
11: Update the priorities for all $s \in$ *OPEN* according to $f(s)$
12: $CLOSED = \emptyset$
13: ImprovePath()
14: $\varepsilon' = \min(\varepsilon,$
$g(s_{goal}) / \min_{s \in OPEN \cup INCONS}(g(s) + h(s)))$
15: publish current ε'-sub-optimal solution
16: **end while**

The *ImprovePath* (Algorithm 2) function shows how an individual weighted A* search is done with SIPP. As usual in weighted A*, the state in the *OPEN* list with the minimum f-value is expanded until the goal has the cheapest f-value among the states in *OPEN*. The f-value of a state s is a function of $g(s)$ and $h(s)$ that serves as a priority for expanding s, when s is in the *OPEN* list. A state expansion

starts on line 4, where it iterates over the possible motion primitives (actions) that can be applied without collision with the static environment. The variable x' is the spatial configuration the robot ends up at after applying the motion m to the spatial configuration of state s, $x(s)$. Lines 6-8 then create the minimum and maximum times the robot could arrive in x' based only on the earliest and latest times it can leave the safe interval of s. On lines 9-16, the algorithm tries to put one successor at the earliest possible time in each of the safe intervals at location x' between t_{start} and t_{end}. It does this by applying a wait operation and then the motion m (the wait time can be 0 so that it applies the motion immediately).

Figure 3 demonstrates how this works. Suppose the search is currently expanding state s_1. If $g(s_1)$ is less than 4 as it is in Figure 3(b) then two safe intervals can be reached with the action that takes the robot to location x'. The first can be reached by applying the motion immediately, which results in state s_2. To reach the second interval, a wait is applied such that after applying the motion, s_3 is reached at time 6. Therefore, only one state is put in each safe interval and they are generated with the earliest possible time. An implementation detail: if the motion m is large enough that it passes through several configurations, it may not be possible to arrive at some intervals at their earliest timestep if one of the states it passes through has dynamic obstacles as well. In these cases, the planner tries all waiting timesteps in increasing order until it finds the earliest that is collision free.

In order to guarantee completeness and the sub-optimality bound when $\varepsilon > 1$, the algorithm uses two versions of each state, optimal and sub-optimal. If the expanded state is of type optimal, then it generates two identical versions of each successor (s' and s'') except that one has the optimal variable $O(s')$ set to true, while the other has $O(s'')$ set to false. If the expanded state is sub-optimal, then it can only generate more sub-optimal states. The sub-optimal states are put into the *OPEN* list with the usual $f(s') = g(s') + \varepsilon * h(s')$. Optimal states are put into the *OPEN* list with $f(s') = \varepsilon * (g(s') + h(s'))$ which preserves the optimal expansion order among the optimal states (since f is just multiplied by a scalar) but tends to put optimal states later in the *OPEN* list relative to sub-optimal states since both terms are inflated instead of just one. This means that most of the time, only sub-optimal states are expanded. It turns out that just enough optimal states are expanded in order to preserve the theoretical guarantees (completeness and bounds on sub-optimality) expected of weighted A* search.

Note that lines 7 and 27 stress the assumption that the cost function being minimized for SIPP is time. Therefore, $g(s)$ can be treated as the time value for state s. Finally, notice that if a state is in the *CLOSED* list, it has already been expanded (line 3) and if a cheaper path is found to such a state (line 27-29), the g-value is updated, but it is put into the *INCONS* list instead of *OPEN*. This ensures that each state is only expanded once, but the search keeps track of the states that need to propagate their cheaper costs in the next search iterations of ARA*.

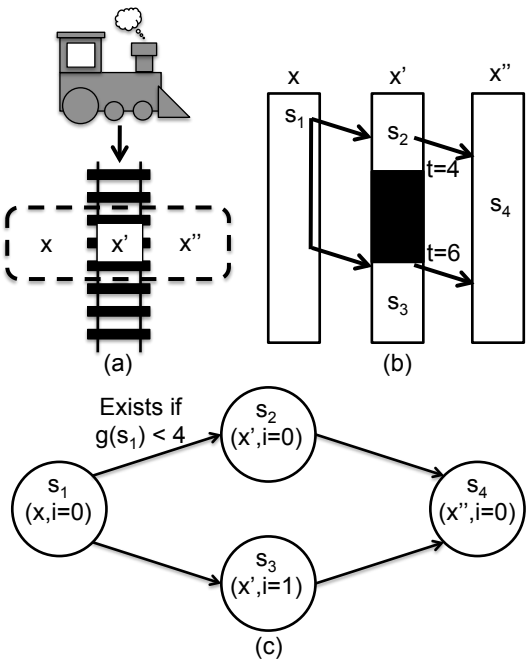

Fig. 3: (a) A scenario with three locations (x, x', x'') and a train which arrives at x' at time 4 and leaves at time 6. (b) The intervals shown for the three locations. Since x and x'' are free for all time, they each have only one interval (therefore one state). Location x' has a safe interval before the trains passes $(t < 4)$ and after $(t > 6)$, resulting in two states for this location (s_2, s_3). (c) The intervals converted into a graph with one motion that depends on the g-value (time) of state s_1. The edge (s_1, s_2) only exists if $g(s_1) < 4$. The edge going to s_3 always exists due to the robot's ability to wait in place before executing an action.

D. Theoretical Analysis

Here we sketch the proofs for completeness and optimality of our algorithm.

Theorem 1: Anytime SIPP is complete. The algorithm terminates, and when it does, it returns a safe path to the goal with respect to static and dynamic obstacles, provided such a path exists for the given graph representation of the planning problem.

The first iteration of Anytime SIPP is just a weighted A* search with SIPP as shown in CFDA-A*. Therefore, since CFDA-A* is complete, the first solution will be valid with respect to the given dynamic obstacle trajectories. Any improvements to the path that ARA* makes as time allows do not harm the correctness of the solution.

Theorem 2: When the planning time expires, the solution returned by Anytime SIPP has a cost no greater than ε' times the cost of the optimal solution.

When running weighted A* with SIPP (as done in CFDA-A*), the solution is guaranteed to be no greater than ε' times the optimal solution. Since weighted SIPP satisfies this requirement, applying ARA* provides this bound.

Algorithm 2

ImprovePath()

```
1: while f(s_goal) > min_{s∈OPEN}(f(s)) do
2:   remove s with the smallest f(s) from OPEN
3:   CLOSED = CLOSED ∪ {s}
4:   for all motion m ∈ M(s) do
5:     x' = configuration of m applied to x(s)
6:     t_m = time to execute m
7:     t_start = g(s) + t_m
8:     t_end = endTime(interval(s)) + t_m
9:     for all safe interval i ∈ x' do
10:      if startTime(i) > t_end or endTime(i) < t_start
         then
11:        continue
12:      end if
13:      t = earliest arrival time at x' in interval i with
         no collisions
14:      if t does not exist then
15:        continue
16:      end if
17:      if O(s) = true then
18:        Opt = {true, false}
19:      else
20:        Opt = {false}
21:      end if
22:      for all o ∈ Opt do
23:        s' = {x', i, o}
24:        if s' was not visited before then
25:          f(s') = g(s') = ∞
26:        end if
27:        if g(s') > t then
28:          g(s') = t
29:          if s' ∉ CLOSED then
30:            if O(s') then
31:              f(s') = ε * (g(s') + h(s'))
32:            else
33:              f(s') = g(s') + ε * h(s')
34:            end if
35:            insert s' into OPEN with f(s')
36:          else
37:            insert s' into INCONS
38:          end if
39:        end if
40:      end for
41:    end for
42:  end for
43: end while
```

E. Time-bounded Graph Representation

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 long-term prediction is likely to be incorrect as it is nearly impossible to predict the precise long-term behavior of dynamic obstacles. Based on this observation, the time-bounded lattice representation [5] exploits this observation and constructs a graph that consists of two types of states: the states defined by both configuration and timestep (higher-dimensional states) and states defined purely by spatial configuration (lower-dimensional states). The former states appear near the robot. These are all and only states reachable from the robot state within the given time horizon. In the time-bounded lattice graph, the states whose timestep is equal to the time horizon have transitions to states that do not include time (lower-dimensional states). Thus, once the timestep reaches the given time horizon, the states become "timeless".

It is even more trivial to incorporate the time-bounded representation into the safe interval-based graph representation within SIPP and Anytime SIPP. Limiting the time horizon in SIPP corresponds to truncating the duration of predicted trajectories of dynamic obstacles to be no more than the time horizon. (In fact, a different time horizon can be applied to different dynamic obstacles depending on certainty in prediction.) Since SIPP already compresses timesteps into safe intervals, the states whose g-value (time value) is larger than the given time horizon will all have the same safe interval, namely the last safe interval (the interval that goes into infinity). Thus, with SIPP and Anytime SIPP, the implementation of finite time horizon is done purely by truncating the durations of the predicted trajectories of dynamic obstacles. Note that using the time horizon might make the solution incomplete with respect to the full dynamic obstacle trajectories that are given.

IV. EXPERIMENTAL RESULTS

To demonstrate the advantages of our anytime algorithm, we ran simulations on randomly generated environments, for several planner configurations. In addition, we provide some guidelines for choosing planner parameters, based on our observations.

A. Planner Implementation

Our test domain is planning for a UAV in the (x, y, z, θ, t) 5D space. The actions used to get successors for states are a set of "motion primitives," which are short kinematically feasible motions sequences [6] used in a lattice-type planner shown in Figure 4. For our A* heuristic, we project the environment to a 2D plane, where a cell (x_i, y_i) is marked as an obstacle iff (x_i, y_i, z) is occupied for *all* z. We then run a 16-connected 2D Dijkstra search from the goal to all the (x, y) cells in this projection, assuming that the robot is circular, with a radius equal to the actual robot's inscribed sphere. Since our lattice planner also has its orientation dimension discretized into 16 directions, the heuristic value in each cell under-estimates the cost to the goal. This stems

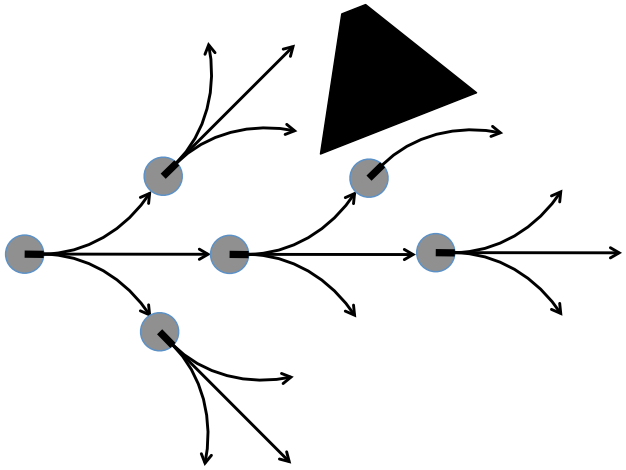

Fig. 4: An example of a lattice type graph

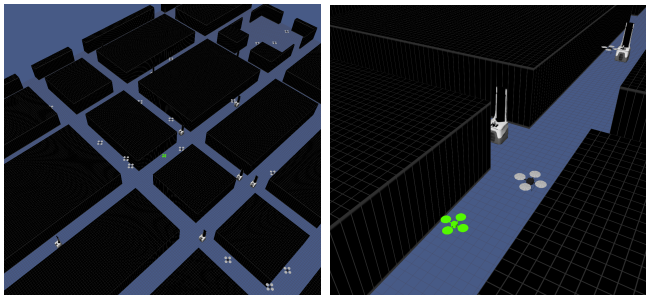

Fig. 5: Left: Aerial view of an example environment used in our experiments. Right: Close up of the environment showing the quadrotor (in green) that we are planning for, and dynamic obstacles that include other quadrotors and ground robots.

from the fact that the 2D search essentially assumes that the robot can turn in place at no cost and its 2D collision model is a circle inscribed into the projected footprint of the UAV. This heuristic is computed quickly, relative to our search, because it has much lower (2D) dimensionality. However, it is also much more informative than the common Euclidean distance heuristic, since it takes static obstacle information into account. This heuristic is especially useful in indoor environments since walls that occupy entire z columns would be projected down to the 2D heuristic grid.

B. Experiment Design

We performed experiments to demonstrate that our algorithm can be used in real-time on large environments with many dynamic obstacles and investigated the trade off between planning times and the sub-optimality bound, as well as the effect of the time horizon. We tested our algorithm on 40 randomly generated experiments to simulate indoor type environments, where all environments were 300 by 300 by 10 cells, with θ being discretized into 16 directions. The time dimension had a resolution of 0.1s. The robot's start and goal were chosen randomly for each map. For each environment, 50 dynamic obstacles were generated,

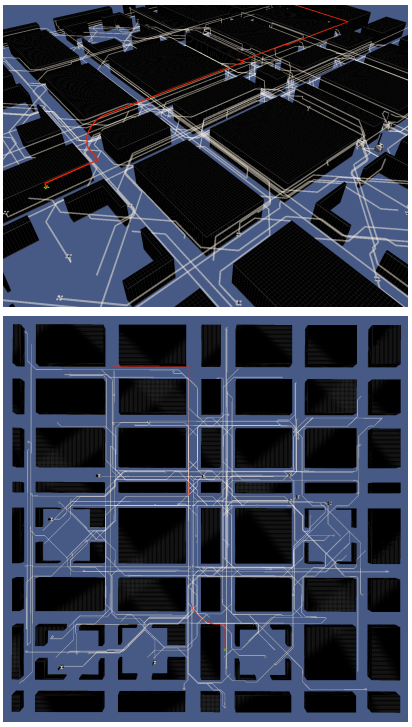

Fig. 6: An instance of the experiment, showing the paths of all dynamic obstacles (in white) and the path of the robot (in red) returned by the planner. Top: Perspective view. Bottom: Overhead view

where each dynamic obstacle was either a smaller UAV, which could be avoided by going under or above it, or a larger ground robot, that had to be circumvented. Each dynamic obstacle was started at a random configuration in the environment, and to generate a trajectory for a dynamic obstacle, random goals were chosen and A* was used to find a path to follow between the terminal points (2D for ground robots and 3D for UAVs). The environments (Figure 5) are composed of a series of randomly placed narrow hallways and rooms. Figure 6, based on the accompanying video, shows an instance of the experiment, with the paths of all dynamic obstacles and the path of the robot returned by the planner superimposed on the environment. All experiments were run on a PC with a 3.4 GHz Intel Core i7-2600QC processor with 8 GB of RAM.

C. Results

An important characteristic of an anytime planner is its ability to get a first solution very fast. Figure 7 shows the amount of time it takes for our algorithm to get a first solution as we vary the initial ε and the time horizon. The results are averaged over 40 trials. In these experiments the trajectory of any dynamic obstacle is at most 35s long and therefore, the plots would stay constant after a time horizon of 35s. The plot indicates that in under 0.05s, the planner can find a solution with a sub-optimality bound of 3, that doesn't collide with any dynamic obstacles for the next 10s. If only a 5s horizon is needed, then the planning times would

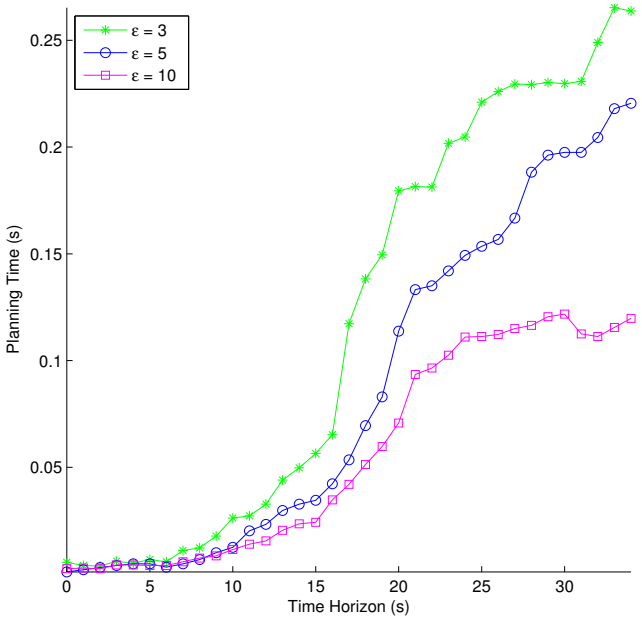

Fig. 7: Average planning time to find the first solution as a function of time horizon. The plot for $\varepsilon = 1$ is not pictured, but it ranged from 5 to 8 seconds, which is far too high for real-time scenarios.

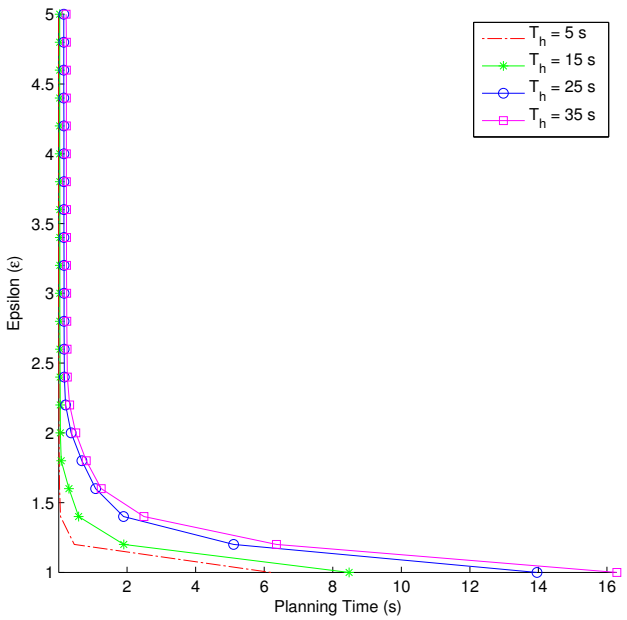

Fig. 8: The average planning time required to reach sub-optimality bound ε. The different lines show the results for various time horizon values.

TABLE I: Planning time distributions for SIPP and Anytime SIPP (ASIPP)

Plan time to first solution	SIPP	ASIPP		
		$\varepsilon = 3$	$\varepsilon = 5$	$\varepsilon = 10$
<20 ms	0%	60.92%	66.57%	68.07%
20-40 ms	0%	10.71%	10.28%	8%
40-60 ms	0%	5.07%	4.28%	6.35%
60-80 ms	0%	2.35%	1.71%	2.71%
80-100 ms	0%	4.35%	0.78%	1.78%
0.1-1 s	7.42%	13.57%	14.14%	12.14%
1-3 s	23.35%	2.35%	2.21%	0.92%
3-5 s	22.64%	0.64%	0%	0%
5-7 s	8.42%	0%	0%	0%
7-9 s	11.42%	0%	0%	0%
>9 s	26.71%	0%	0%	0%

be closer to 0.01s. These results indicate that the planner is capable of returning some solution almost right away. Table I puts these numbers into perspective by comparing them with the time required to get a first solution from optimal ($\varepsilon = 1$) SIPP. The entries in the table are distributions of planning times for SIPP and Anytime SIPP (ASIPP) for 4 different time horizons, computed over 40 experiments. It is evident from the table that Anytime SIPP provides a planning time that is at least two orders of magnitude less than optimal SIPP. These results indicate that a weighted search is needed to get some solution quickly. While left out, the number of state expansions is reflective of planning times (just off by a scalar multiple).

If the planner is given more time than this though, our approach will improve the solution quality using ARA*. Figure 8 shows the quality of the solution obtained for different planning times. For this experiment, the sub-optimality bound ε was decremented by 0.2 between successive iterations of ARA*. Once ε has reached 1, the solution is provably optimal. The plot shows the results for a few values of the time horizon and, as expected, the planning times are higher when we have longer time horizons or when we require close to optimal solutions. The fact that planning times do not scale linearly with sub-optimality bounds is one of the reasons behind Anytime SIPP's marked improvement over the optimal version. Moreover, in practice, the sub-optimal solution costs are higher than the optimal costs by a factor which is much less than the theoretical sub-optimality bound. For instance, Figure 9 shows that the solution cost for $\varepsilon = 1.4$ is greater than the optimal solution cost only by 1.77s, for a time horizon of 15s. In theory, the upper bound on this difference would be 11.124s, which is very conservative.

V. DESIGN CONSIDERATIONS

The major parameters that our algorithm depends on are the time horizon, the epsilon used to initialize the planner, and the maximum time allocated to the planner. Values for these parameters can be chosen based on the application at hand. For instance, if we have a constraint on the maximum time, T_{plan}, that the planner can take, and a requirement that the path be safe for at least time T_{safe}, we would simply lookup Figure 7 and choose an ε that minimizes $T_{plan} - time(T_{safe})$, subject to $T_{plan} > time(T_{safe})$. On the other hand, if our application imposes a strong constraint on the quality of the path returned ($\varepsilon_{desired}$), we would refer to Figure 8 and choose a time horizon T_h that minimizes $\varepsilon_{desired} - \varepsilon(T_{plan})$, subject to $\varepsilon_{desired} > \varepsilon(T_{plan})$. Here, $time(t)$ is the average planning time for time horizon t and $\varepsilon(t)$ is the epsilon that is typically reached by given a planning time t.

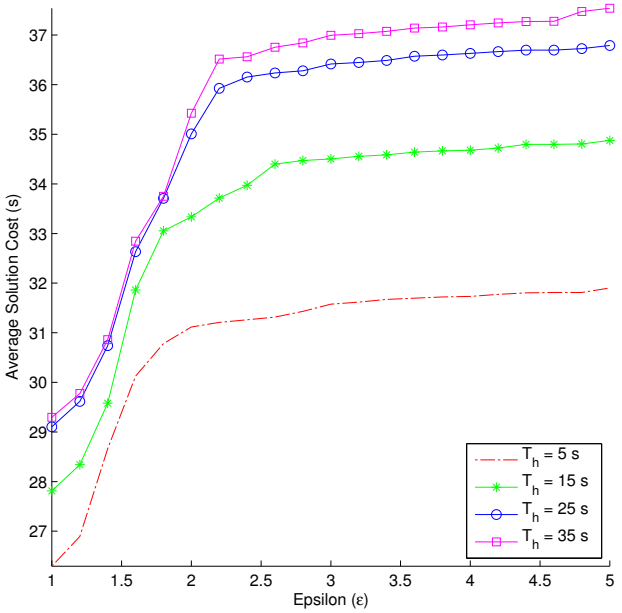

Fig. 9: The average solution cost as a function of sub-optimality bounds (ε) used in the successive iterations of ARA*. The different curves show the results for various time horizon values.

As a general observation, shorter horizons are always better as they result in quicker planning times for any ε. But in practice, how small our horizon can be depends on the environment, the number of dynamic obstacles, and their velocities. In the experiments we ran, we observed that a time horizon of 5s provides planning times on the order of 0.004 to 0.007s, whereas a time horizon of 15s provides planning times on the order of 0.024 to 0.056s, for values of ε ranging from 3 to 10. Smaller time horizons can be used at the cost of increased sub-optimality for rapidly changing environments such as crowded museums and shopping malls, where safety of the people and robot are more important. However, for environments such as a workplace, where dynamic obstacles trajectories can be predicted with some confidence, we might prefer to use longer time horizons for close to optimal paths.

VI. CONCLUSIONS

In this paper, we have developed an anytime planner for planning in dynamic environments. We built off of *Safe Interval Path Planning* which efficiently finds solutions to this problem by representing time as safe intervals instead of timesteps. We extended this to anytime search by combining it with ARA*, an anytime planner which provides bounds on the sub-optimality of the solution it returns. We further accentuate the anytime nature of this planner by using a time horizon after which the planner drops to the more efficient planning on the static map.

Our experimental results on a simulated UAV show that our approach is capable of running in real-time by providing solutions up to a 15s time horizon in under 0.05s on large environments with many dynamic obstacles. Additionally, our approach provides theoretical guarantees on completeness and solution quality.

In future work, we would like to have the algorithm running on a physical robot. Also, an interesting direction for future work would be to add an incremental component to the algorithm, that can re-use data when navigating partially-known environments or environments in which the predicted trajectories of dynamic obstacles change frequently. In addition, we are interested in planning for situations where dynamic obstacles have multiple possible future trajectories and in generating efficient policies to handle this uncertainty.

VII. ACKNOWLEDGMENTS

This research was partially sponsored by the DARPA Computer Science Study Group (CSSG) grant D11AP00275, ONR DR-IRIS MURI grant N00014-09-1-1052, and DARPA contract W31P4Q10C0202. We also thank Stefan Kohlbrecher and Johannes Meyer for making their URDF model of a quadrotor open-source and available for visualization purposes.

REFERENCES

[1] K. Bekris and L. Kavraki. Greedy but safe replanning under kinodynamic constraints. *In IEEE International Conference on Robotics and Automation*, 2007.

[2] D. Fox, W. Burgard, and S. Thrun. The dynamic window approach to collision avoidance. *IEEE Robotics and Automation*, 4(1), 1997.

[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] S. Karaman and E. Frazzoli. Incremental sampling-based algorithms for optimal motion planning. *CoRR*, abs/1005.0416, 2010.

[5] A. Kushleyev and M. Likhachev. Time-bounded lattice for efficient planning in dynamic environments. *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2009.

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

[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] 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.

[9] M. Phillips and M. Likhachev. Planning in domains with cost function dependent actions. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, 2011.

[10] M. Phillips and M. Likhachev. Sipp: Safe interval path planning for dynamic environments. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2011.

[11] M. Pivtoraiko and A. Kelly. Generating near minimal spanning control sets for constrained motion planning in discrete state spaces. In *Proceedings of the International Conference on Intelligent Robots and Systems (IROS)*, 2005.

[12] I. Pohl. First results on the effect of error in heuristic search. *Machine Intelligence*, 5:219–236, 1970.

[13] M. Rufli, D. Ferguson, and R. Siegwart. Smooth path planning in constrained environments. In *Proc. of The IEEE International Conference on Robotics and Automation (ICRA)*, 2009.

[14] D. Silver. Collaborative pathfinding. *In Proceedings of AIIDE*, 2005.

[15] 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.

[16] J. van den Berg and M. Overmars. Roadmap-based motion planning in dynamic environments. *IEEE Transactions on Robotics*, 21(5):885–897, 2005.

[17] D. Wilkie, J. van den Berg, and D. Manocha. Generalized velocity obstacles. In *IROS*, pages 5573–5578, 2009.