Adaptive Time Stepping in Real-Time Motion Planning

Kris Hauser

School of Informatics and Computing, Indiana University, hauserk@indiana.edu

Abstract: Replanning is a powerful mechanism for controlling robot motion under hard constraints and unpredictable disturbances, but it involves an inherent tradeoff between the planner's power (e.g., a planning horizon or time cutoff) and its responsiveness to disturbances. We present a real-time replanning technique that uses adaptive time stepping to learn the amount of time needed for a sample-based motion planner to make monotonic progress toward the goal. The technique is robust to the typically high variance exhibited by planning queries, and we prove that it is asymptotically complete for a deterministic environment and a static objective. For unpredictable environments, we present an adaptive time stepping contingency planning algorithm that achieves simultaneous safety-seeking and goal-seeking motion. These techniques generate responsive and safe motion in simulated scenarios across a range of difficulties, including applications to pursuit-evasion and aggressive collision-free teleoperation of an industrial robot arm in a cluttered environment.

1 Introduction

Robots must frequently adjust their motion in real-time to respond to unmodeled disturbances. A common approach to deal with nonlinear dynamics and hard state and control constraints is to reactively replan at each time step (Figure 1). This basic approach has been studied under various nomenclature (model predictive control, receding horizon control, or real-time planning) and using various underlying planners (numerical optimization, forward search, or sample-based motion planners), and is less susceptible to local minima than myopic potential field approaches. But replanning, in all its forms, faces a fundamental tradeoff based on the choice of time limit: too large, and the system loses responsiveness; too short, and the planner may fail to solve difficult problems in the allotted time, which sacrifices global convergence and safety. Empirical tuning by hand is the usual approach. But the time needed to solve a planning query can vary by orders of magnitude not only between problems, but also between different queries in the same problem, and even on the same

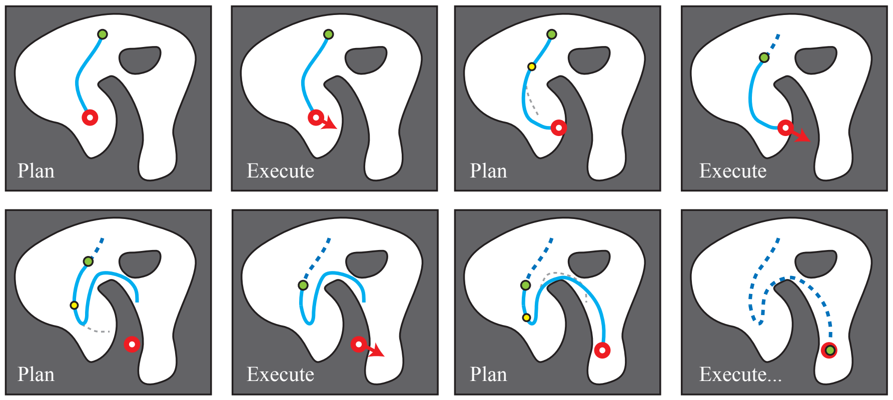

Fig. 1: A point robot (green) interleaves execution and replanning to reach an unpredictably moving target (red).

query (in the case of randomized planners). Unless variability is addressed, the safety and completeness of real-time replanning is in doubt.

This paper presents two replanning algorithms that address safety and completeness not by reducing variability in planning time, but by tolerating and adapting to it. They use a sample-based planner to build partial plans whose endpoints monotonically improve an objective function, and adaptively learn a suitable time step on-the-fly by observing whether the planner is able to make progress within the time limit. The first algorithm, described in Section 3, guarantees safe motion in deterministic, predictable environments by construction, and furthermore we prove that the state of the robot is guaranteed to globally optimize the objective function in expected finite time for a large class of systems. We apply it to real-time obstacle avoidance for a simulated 6DOF industrial robot moving dynamically in a cluttered environment. The second algorithm, described in Section 4, uses a conservative contingency planning approach to achieve a higher probability of safety in unpredictable or adversarial environments, and we apply it to a pursuit-evasion problem. Experiments suggest that adaptive time stepping is more consistent than constant time stepping across problem variations for both algorithms.

2 Related Work

Bounded Rationality in Real-Time Agents. Real-time planning architectures have a long history of study in artificial intelligence, control theory, and robotics, but few have explicitly addressed the problem of “bounded rationality”, where limited computational resources hamper an agent’s ability to produce timely, optimal plans. A notable exception is the CIRCA real-time agent architecture [12] that separates the agent’s control into high-level planning and low-level reactive control tasks. The high-level task conveys controller specifications to the low-level task whenever planning is complete. The disadvantage of this approach is that uncertainty in computation time causes uncertainty in state, leading to harder planning problems. By contrast our approach is constructed to avoid state uncertainty, at least when the system is deterministic, which makes planning more tractable.

Replanning Applications and Implementations. Model predictive control (MPC), aka receding horizon control, is a form of replanning that at each time step formulates a optimal control problem truncated at some horizon. Such techniques have been successful in robot navigation [2,16]; for example the classic dynamic windowing technique introduced for indoor mobile robot navigation is essentially MPC by another name [16]. In nonlinear systems, truncated optimal control problems are often solved using numerical optimization or dynamic programming [1,11]. In discrete state spaces, efficient implementations of replanning algorithms include the D* and Anytime A* algorithms which are based on classic heuristic search [10,17,18].

Sample-based motion planners such as randomly-exploring random trees (RRTs) and expansive space trees (ESTs) have been applied to real-time replanning for dynamic continuous systems [4,5,7,20]. RRT and EST variants have been applied to 2D helicopter navigation [5], free-floating 2D robots [7], and and car-like vehicles [13] among moving obstacles, as well as exploring an unknown environment [3]. Our algorithms also use sample-based planners.

Time Stepping in Replanning. Although many authors have proposed frameworks that can handle nonuniform time steps [3,5,13,20], few actually *exploit* this capability to adapt to the power of the underlying planner. We are aware of one paper in the model predictive control literature [14] that advances time exactly by the amount of time taken for replanning. The weakness of this approach is that if replanning is slow, the actions taken after planning are based on outdated state estimates, leading to major instability and constraint violations. Our work avoids this problem by setting planner cutoffs and projecting state estimates forward in time at the start of planning.

Safety Mechanisms. Several mechanisms have been proposed to improve the safety of replanning in dynamic environments. Feron et al introduced the notion of τ-safety, which indicates that a trajectory is safe for at least time τ [5]. Such a certificate establishes a hard deadline for replanning. Hsu et al introduced the notion of an “escape trajectory” as a contingency plan that is taken in case the planner fails to find a path that makes progress toward the goal [7]. We use a contingency planning technique for unpredictable environments that is much like a conservative escape trajectory approach, except that it always ensures the conservative path is followed in case of a planning failure.

The notion of inevitable collision states (ICS) was introduced by Petti and Fraichard to the problem of real-time planning for a car-like vehicle among moving obstacles [13]. An ICS is a state such that no possible control can recover from a collision, and considering ICS as virtual obstacles prevents unnecessary exploration of the state space. In practice, testing for ICS can only be done approximately, and the conservative test proposed in [13] may prevent the robot from passing through states that are actually safe. Our work provides similar safety guarantees without explicit testing for ICS.

Speeding up Replanning. Many approaches have sought to improve responsiveness by simply reducing average replanning time. Some common techniques are to reuse information from previous plans [4], to use precomputed coarse global plans to essentially reduce the depth of local minima [2, 8, 16, 19], or a combination [3, 20]. These approaches are mostly orthogonal to the choice of time step and can be easily combined with adaptive time stepping.

3 Replanning in Deterministic Environments

In real-time replanning the robot interleaves threads of *replanning* and *execution*, in which the robot (at high rate) executes a partial trajectory that is intermittently updated by the replanning thread (at a lower rate) without interrupting execution. The planner is given a time cutoff Δ, during which it generates a new safe trajectory originating from a state propagated in the future by time Δ. This section presents and analyzes the adaptive time-stepping technique and an application to real-time assisted teleoperation of a robot manipulator in deterministic environments.

3.1 Assumptions and Notation

The state of the robot x lies in a state space S, and its motion must obey differential constraints $\dot{x} \in U(x,t)$ (note that this is simply a more compact way of writing control constraints). We assume that the robot has a possibly imperfect model of the environment and how it evolves over time, and let $F(t) \subseteq S$ denote the subset of feasible states at time t. We say that a trajectory $y(t)$ is *τ-safe* if $\dot{y}(t) \in U(t)$ and $y(t) \in F(t)$ for all $0 \leq t \leq \tau$. If so, we say $y(t)$ is an F_τ trajectory.

In this section we will be concerned primarily with F_∞ trajectories. We will assume that F_∞ feasibility is achieved by ensuring that each trajectory terminates at a feasible stationary state. For certain systems with dynamics, such as cars and helicopters, a "braking" control can be applied. This paper will not consider systems like aircraft that cannot reach zero velocity, although terminal cycles may be considered as a relatively straightforward extension.

We address the problem of reaching a global minimum of a smooth time-invariant potential function $V(x)$ via an F_∞ trajectory $y(t)$ starting from the initial state x_0. Assume the global minimum is known and attained at $V(x) = 0$ without loss of generality. We say any trajectory that reaches $V(x) = 0$ is a *solution trajectory.* We do not consider path cost, and define the cost

```
Algorithm 1. Replanning with an Adaptive Time-Step
Initialization:
0a. $y(t)$ is set to an $F_\infty$ initial trajectory starting from $t = 0$.
0b. $\Delta_1$ is set to a positive constant.

Repeat for $k = 1, \ldots$:
1. Measure the current time $t_k$
2. Initialize a plan starting from $y(t_k + \Delta_k)$, and plan for $\Delta_k$ time
3. If $C(\hat{y}) \leq C(y) - \epsilon$ for the best trajectory $\hat{y}(t)$ generated so far, then
4.     Replace the section of the path after $t_k + \Delta_k$ with $\hat{y}$
5.     Set $\Delta_{k+1} = 2/3\Delta_k$
6. Otherwise,
7.     Set $\Delta_{k+1} = 2\Delta_k$
```

Fig. 2: Pseudocode for the replanning algorithm.

functional $C(y)$ that simply returns the value of $V(x)$ at the terminal state of the trajectory y. It is important to note that when we refer to an optimal solution, we are referring to the *optimality of the terminal point*, not the trajectory taken to reach it. We also impose the *real-time constraint* that no portion of the current trajectory that is being executed can be modified. So, if a replan is instantiated at time t and is allowed to run for time Δ, then no portion of the current trajectory before time $t + \Delta$ may be modified.

We assume that we have access to an *underlying planner* with the following "any-time" characteristics:

1. The planner iteratively generates F_∞ trajectories starting from an initial state and time $y(t_0)$ given as input.
2. Planning can be terminated at any time, at which point it returns the trajectory that attains the least value of the cost functional $C(y)$ found so far.
3. If the planner is given no time limit on any query that admits a solution trajectory, then the planner finds a solution in expected finite time.

A variety of underlying planning techniques (e.g., trajectory optimization, forward search, and sample-based motion planning) can be implemented in this fashion. All experiments in this paper are conducted with minor variants of the sampling-based planners RRT [9] and SBL [15], which grow trees using forward integration of randomly-sampled control inputs. The running time of such planners is variable across runs on a single query, and can vary by orders of magnitude with the presence of narrow passages in the feasible space.

3.2 Adaptive Time-Stepping With Exponential Backoff

Here we describe our variable-time step replanning algorithm and a simple but effective exponential backoff strategy for learning an appropriate time

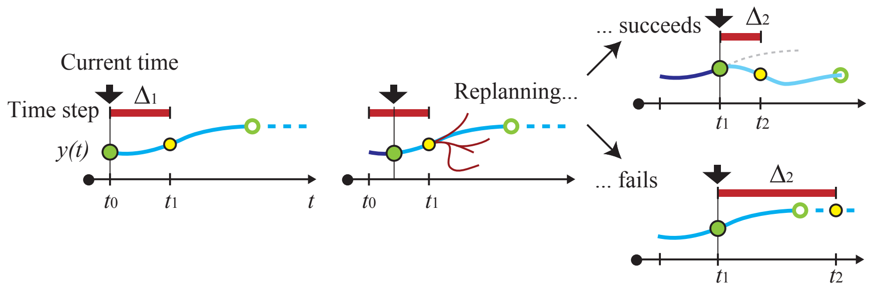

Fig. 3: Each replanning iteration chooses a time step (left), initiates a plan starting from the predicted future state (center), and either succeeds or fails. Upon success, the robot progresses on the new plan and the time step is contracted. Upon failure, the robot retains the original plan and the time step is increased.

step. Pseudocode is listed in Algorithm 1 in Figure 2. The replanning thread takes time steps $\Delta_1, \Delta_2, \ldots$. In each time step, the planner is initialized from the state on the current trajectory at time $t_k + \Delta_k$, and plans until Δ_k time has elapsed (Line 2). If the planner finds a trajectory with lower cost than the current trajectory (Line 3) then the new trajectory is spliced into current trajectory at the junction $t_k + \Delta_k$ (Line 4). Otherwise, the current trajectory is left unaltered and replanning repeats.

Note that the condition in Line 3 requires a decrease by some small constant $\epsilon > 0$. (To allow the planner to reach the global minimum exactly, an exception can be made on the final step when $C(\hat{y}) = 0$ is attained.) This simplifies later analysis by preventing the theoretical occurrence of an infinite number of infinitesimal cost improvements.

Lines 5 and 7 implement a simple exponential backoff strategy for choosing the time cutoff. This permits recovery from a local minimum of $V(x)$ in case several planning failures are encountered in sequence. Such strategies are widely used in protocols for handling network congestion, and there is a rough analogy between uncertainty in planning time and uncertainty in message delivery over an unreliable network. The idea is simple: if the planner fails, double the time step (Line 7). If it succeeds, contract the time step (Line 5). The constant 2/3 that we use in the contraction strategy does not need to be chosen particularly carefully; resetting Δ_{k+1} to a small value works well too. Figure 3 illustrates one iteration of the protocol.

3.3 Completeness and Competitiveness

We can now state a basic theorem that guarantees that Algorithm 1 is probabilistically complete for static goals as long as the robot never reaches a state where the goal becomes unreachable.

Theorem 1. *If the environment is deterministic and perfectly modeled, and the goal is reachable from any state that is reachable from the start, then Algorithm 1 will find a solution trajectory in expected finite time.*

Proof. Let $\mathcal{R}$ be the set of states reachable from the start, and let $T(x)$ be the expected planning time for finding a solution trajectory starting at $x \in \mathcal{R}$. Because of the assumption in Section 3.1, $T(x)$ is finite, and so is the maximum of $T(x)$ over all $\mathcal{R}$, which we denote to be T_{max}. First we will show that the time until a plan update has a finite expected value.

Suppose Algorithm 1 has its first plan update on the k'th iteration after $k-1$ failed iterations (the possibility that no such k exists is vanishingly small). Because $k-1$ iterations have passed without an update, the planning cutoff on the k'th iteration is $2^k \Delta_0$. So the total time spent T over all the k iterations is a geometric series with sum $T = (2^{k+1} - 1)\Delta_0$. Let T_p be the random variable denoting the planning time necessary to find a global solution starting at $(y(t_k), t_k)$. If k were known, then T_p would be lie in the range $(2^{k-1}\Delta_0, 2^k \Delta_0]$, so that the inequality $T < 4T_p$ holds. But the inequality $E[T_p] \leq T_{max}$ holds for all k, x_k, and t_k, so $E[T] < E[4T_p] \leq 4T_{max}$ unconditionally.

The number N of plan updates needed to reach a global minimum is finite since the initial trajectory has finite cost, and each plan update reduces cost by a significant amount. So, the total expected running time of Algorithm 1 is bounded by $4NT_{max}$, which is finite. □

We remark that the bound $4NT_{max}$ is extremely loose, and seemingly poor compared to the performance bound T_{max} of simply planning from the initial state until a solution is found. In practice, most problems contain few planning queries of extremely high difficulty corresponding to escaping deep local minima of C, and the running time of Algorithm 1 will tend to be dominated by those queries. Smaller, greedy advances in $C(y)$ are often much quicker to plan.

By construction Algorithm 1 will never drive the robot to an inevitable collision state (ICS) as defined in [13], so it is equivalently "safe". But in which systems is it asymptotically complete? The key assumption of Theorem 1 is that the goal can be reached by all states in $\mathcal{R}$ (this can be slightly weakened to take $\mathcal{R}$ as those states actually reached by the robot during execution). For example, it holds in reversible systems. In general, however, Algorithm 1 might inadvertently drive the robot into a dead end from which it cannot escape — a condition we might call an *inevitable failure state.* In fact, non-reversible systems seem to prove quite troublesome to all replanning techniques because detecting dead ends requires sufficient global foresight that might not be practical to achieve with limited computation time. These are fruitful directions for future work.

As a final note, we remark that Algorithm 1 is not necessarily complete in problems with time-varying potential V. For example, if the global minima of V might alternate quickly between two locally easy but globally difficult

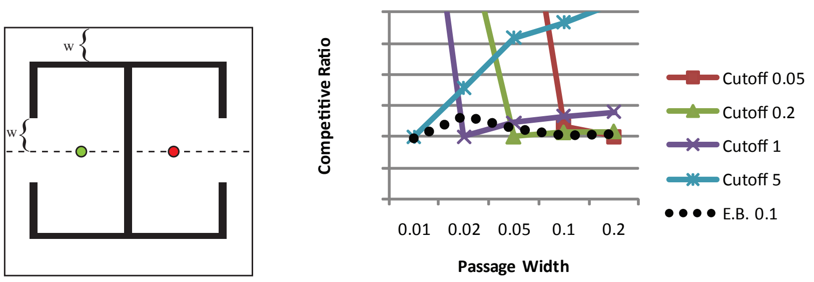

Fig. 4: (a) Our 2D benchmark problem. Passage width, and hence, difficulty, is parameterized by w. (b) The performance of constant cutoff strategies varies greatly across passage widths, and short cutoffs (0.05, 0.1, and 0.2) fail completely on difficult problems. The adaptive exponential backoff strategy (E.B.) achieves consistent performance across problem variations. Performs is measured by solution time, normalized by the time of the best constant cutoff for that problem.

problems, then the algorithm will forever be able to make local progress and will thereby keep the time step short.

3.4 Completeness and Sensitivity Experiments

We evaluated the performance of the adaptive strategy against constant time stepping strategies on a static 2D benchmark across varying problem difficulties. Consider a unit square state space S where the state is subject to velocity constraints $||\dot{x}|| \leq 1$. Obstacles partition the feasible space $F(t)$ into two “rooms” with opposite-facing doorways, which are connected by hallways (see Figure 4). The state must travel from $(0.3, 0.5)$ to $(0.6, 0.5)$, and the potential function $V(x)$ simply measures the distance to the goal. For replanning we use a unidirectional RRT planner [9], which, like other sample-based planners, is sensitive to the presence of narrow passages in the feasible space. We control the difficulty of escaping local minima by varying a parameter w, and set the hallway widths to w and the doorway widths to $2w$.

We measure performance as the overall time taken by the robot to reach the goal, averaged over 10 runs with different random seeds. If it cannot reach the goal after 120 s, we terminate the run and record 120 s as the running time. Experiments compared performance over varying w for constant cutoffs 0.05, 0.1, 0.2, 0.5, 1, 2, and 5 s and the exponential backoff algorithm starting with $\Delta_1 = 0.1$ (performance was relatively insensitive to choice of Δ_1).

Figure 4 plots the performance ratios of several strategies. Performance ratio is measured relative to the best constant time step for that problem. Shorter cutoffs are unreliable on hard problems because the planner is unable to construct paths that escape the local minimum of the initial room. On the other hand, longer cutoffs waste time on easier problems. The adaptive strategy delivers consistent performance, performing no worse than 1.4 times that of the best constant cutoff across all problems.

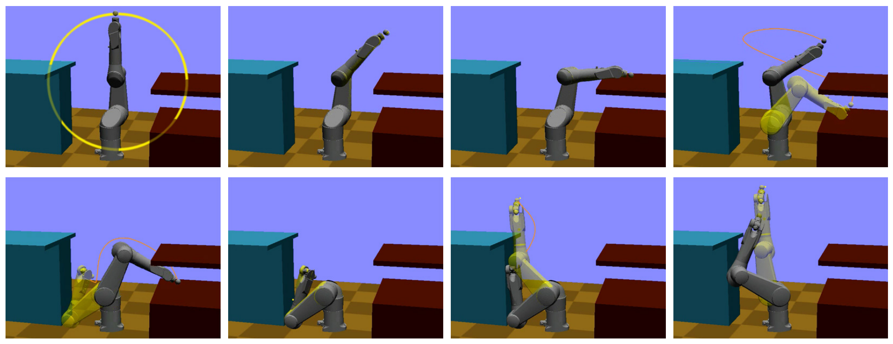

Fig. 5: A Staübli TX90L manipulator is commanded in real time to move its end effector in a clockwise circle in a cluttered environment. The robot responds reactively to the target's motion. Along the upper semicircle, rapid replanning with a short time step allows the target to be followed closely. When obstacles are encountered on the lower semicircle, planning becomes more difficult. Adaptive time stepping gives the planner sufficient time to enter and escape deep narrow passages. The current plan is drawn in orange, and its destination configuration is drawn transparently.

3.5 Assisted Teleoperation Experiments on a 6DOF Manipulator

Replanning interleaves planning and execution, so motion appears more fluid than a pre-planning approach. This is advantageous in human-robot interaction and assisted teleoperation applications where delays in the onset of motion may be viewed as unnatural. We implemented a teleoperation system for a dynamically simulated 6DOF Staübli TX90L manipulator that uses replanning for real-time obstacle avoidance in assisted control. The robot is able to reject infeasible commands, follow commands closely while near obstacles, and does not get stuck in local minima like potential field approaches.

In this system, an operator controls a 3D target point (for example, using a joystick or a laser pointer), and the robot is instructed to reach the point using its end effector. The robot's state space consists of configuration $\times$ velocity, and its acceleration and velocity are bounded. Its configuration are subject to joint limit and collision constraints. The objective function for the planner is an unpredictably time-varying function $V(x,t)$ which measures the distance from the end effector to the target point.

Our underlying planner is a unidirectional variant of the SBL motion planner [15] that is adapted to produce dynamically feasible paths. We made the following adjustments to the basic algorithm:

- We extend the search tree by sampling extensions to stationary configurations sampled at random. The local planner constructs dynamically feasible trajectories that are optimal in obstacle free environments (a similar strategy was used in [5]). To do so, we use analytically computed trajectories that are time-optimal under the assumption of box-bounds on velocity and acceleration [6].

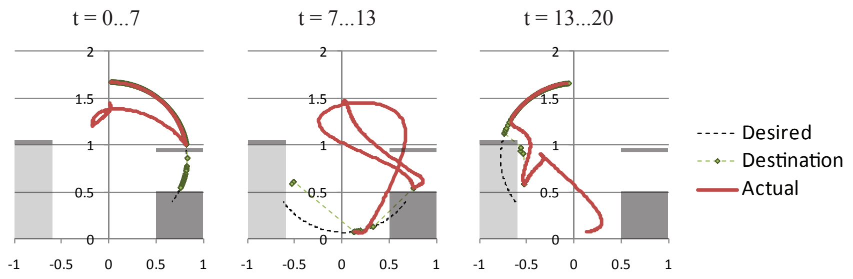

Fig. 6: Traces of the end effector's desired position (Desired), the position at the current plan's destination configuration (Destination), and actual position as executed by the robot (Actual) for the experiment in Figure 5.

- For every randomly generated sample, we generate a second configuration using an inverse kinematics solver in order to get closer to the target.
- SBL uses a lazy collision checking mechanism that improves planning time by delaying edge feasibility checks, usually until a path to the goal is found. We delay edge checks until the planner finds a path that improves $C(y)$.
- To improve the fluidity of motion, we devote 20% of each time step to trajectory smoothing. We used the shortcutting heuristic described in [6] that repeatedly picks two random states on the trajectory, constructs a time-optimal segment between them, and replaces the intermediate portion of the trajectory if the segment is collision free.

The simulation environment is based on the Open Dynamics Engine rigid-body simulation package, where the robot is modeled as a series of rigid links controlled by a PID controller with feedforward gravity compensation and torque limits. The simulation does perform collision detection, but in our experiments the simulated robot did not collide with the environment.

We simulated a user commanding a target to follow a circular trajectory that passes through the robot and obstacles (Figure 5). The circle has radius 0.8 m and a period of 20 s. The upper semicircle is relatively unconstrained and can be followed exactly. Targets along the lower semicircle are significantly harder to reach; at several points they pass through obstacles, and at other points they require the robot to execute contorted maneuvers through narrow passages in the feasible space. Experiments show that replanning can reach a large portion of the lower semicircle while tracking the upper semicircle nearly perfectly (Figure 6).

4 Replanning in Unpredictable Environments

A conservative approach to uncertainty may be preferred in safety-critical applications like transportation and medical robotics. This section presents a real-time contingency planning algorithm that generates both optimistic

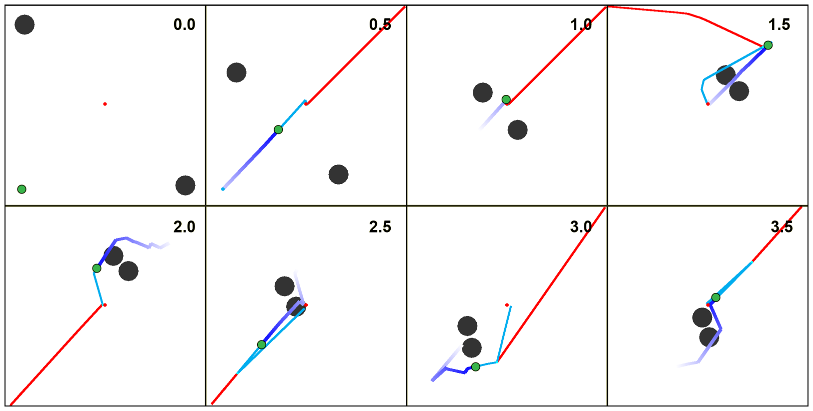

Fig. 7: Snapshots taken at half-second intervals from a pursuit-evasion experiment in the unit square. Two pursuers (grey circles) seek the evader (green) greedily at half the speed of the evader. The evader knows the pursuers' velocity bound but not their behavior. The evader replans a pessimistic path (red) to avoid the pursuers in the worst case, and replans an optimistic path (cyan) in order to reach the goal in the center of the room. Both paths share a common prefix. The trace of the robot between frames is drawn as a purple trail.

(goal seeking) and pessimistic (safety seeking) trajectories to balance safety-seeking and goal-seeking behavior. Adaptive time stepping allows for a high probability of replanning before a certain time limit — the time to potential failure, or TTPF — in which safety is guaranteed. Experiments evaluate the system in a pursuit-evasion scenario.

4.1 Conservative Replanning Framework

We assume that we have access to conservative bounds on the uncertainty of the environment, and let E_k denote the environment model estimated by the robot's sensors at k'th time step. Let $F(t; E_k)$ denote the feasible set with the current model, and let $\tilde{F}(t; E_k)$ denote the set of states that is guaranteed to be feasible at time t under the conservative uncertainty bounds. For example, if obstacle velocities are bounded, then one can consider a conservative space-time "cone" of possible obstacle positions that grows as time increases.

Consider a purely safety-seeking robot that uses the following scheme:

1. The current trajectory $y(t)$ has a *time to potential failure* (TTPF) T if it is safe for some duration T under conservative bounds on uncertainty. That is, $y(t) \in \tilde{F}(t; E_k)$ for all $t \in [t_k, t_k + T]$.
2. Replanning searches for a path $\hat{y}$ that increases the TTPF to $T + \Delta_k$ or some constant T_{min}, whichever is lower.

It is straightforward to use Algorithm 1 to implement such behavior simply by using the TTPF as an optimization criterion. The robot will remain safe unless replanning cannot improve the TTPF within the duration T (and even then, a constraint violation only happens in the worst case)[1]. A violation may occur if 1) no trajectory that improves the TTPF exists, in which case the planner can do nothing except hope that the potential hazard goes away, or 2) not enough planning time was devoted to finding a safe path.

The risk of condition (2) is somewhat mitigated by the selection of the parameter T_{min}, which governs an "acceptable" threshold for the TTPF. Below this threshold, the planner enforces that subsequent pessimistic plans must increase the TTPF. Naturally, if safety were the robot's only objective, the best approach is to set T_{min} to be infinite. In the below section, a finite T_{min} will allow it to make optimistic progress toward a target while being acceptably confident that safety will be ensured.

If the robot must also seek to optimize an objective $V(x)$, it must sacrifice some safety in order to do so. Below we describe a contingency planning framework where the robot's path has a similarly high probability of safety as the above scheme, but the planner seeks to simultaneously increase the TTPF and makes progress towards reducing $V(x)$.

4.2 A Contingency Replanning Algorithm

In our contingency planning algorithm, the robot maintains both an optimistic and a pessimistic trajectory that share a common prefix (Figure 7). The role of the pessimistic trajectory is to optimize the TTPF, while the role of the optimistic trajectory is to encourage consistent progress toward the goal.

Pseudocode is listed in Figure 8. The pessimistic trajectory $y(t)$ is maintained and followed by default. The optimistic trajectory $y^o(t)$, if it exists, is identical to $y(t)$ until the "junction" time t_j. Each iteration of the replanning loop begins by establishing time limits for the optimistic and the pessimistic planners, with sum Δ_k (Line 2). Then a top-level decision is made whether to initiate the new plan from the optimistic or the pessimistic trajectory:

- *From the optimistic trajectory* (Lines 4–9). To continue progress along y^o after time t_j, the robot must generate a pessimistic trajectory that branches out of y^o at some time after t_j. An improvement to the optimistic plan is attempted as well.
- *From the pessimistic trajectory* (Lines 11–20). To progress toward the target, the planner will attempt to branch a new pessimistic and optimistic pair out of the current pessimistic trajectory at time $t_k + \Delta_k$. The new junction time will be $t_k + 2\Delta_k$. If this fails, the planner attempts an extension to the pessimistic path.

To improve the optimistic path, the planner constructs a path in the optimistic feasible space $F(t;t_c)$ based on the current environment model. A

[1] A major benefit of sample-based replanning is that holding TTPF constant, a factor n increase in computational speed results in a sharp reduction in failure rate from p to p^n.

```
**Algorithm 2**. Contingency Replanning with Adaptive Time Steps
*Initialization*:
0a. $y(t) \leftarrow$ an initial trajectory starting from $t = 0$.
0b. $y^o(t) \leftarrow$ nil.
0c. Junction time $t_j \leftarrow 0$

*Repeat for* $k = 1, \ldots$:
1. Measure the current time $t_k$.
2. Pick a pessimistic and optimistic time limit $\Delta_k^p$ and $\Delta_k^o$. Let $\Delta_k = \Delta_k^p + \Delta_k^o$
3. If $t_j > t_k + \Delta_k$ (branch the new plan from the optimistic path)
4.    Plan an improved optimistic path starting from $y_o(t_j)$.
5.    Plan a pessimistic path $\hat{y}$ starting from $y_o(t_j + \Delta_k)$.
6.    If Line 5 is successful, then
7.       Set $y(t) \leftarrow y_o(t)$ for $t \leq t_j + \Delta_k$, and $y(t) \leftarrow \hat{y}(t)$ for $t \geq t_j + \Delta_k$.
8.       Set $t_j \leftarrow t_j + \Delta_k$.
9.    End
10. Otherwise, (branch the new plan from the pessimistic path)
11.    Plan an optimistic path starting from $y(t_k + \Delta_k)$.
12.    If successful, then
13.       Plan a pessimistic path $\hat{y}$ starting from $y_o(t_k + 2\Delta_k)$.
14.       If successful, then
15.          Set $y(t) \leftarrow y_o(t)$ for $t_k + \Delta_k \leq t \leq t_k + 2\Delta_k$, and $y(t) \leftarrow \hat{y}(t)$ for
$t \geq t_k + 2\Delta_k$.
16.          Set $t_j \leftarrow t_k + 2\Delta_k$.
17.       End
18.    Otherwise,
19.       Plan a pessimistic path $\hat{y}$ starting from $y(t_k + \Delta_k)$.
20.       If successful, set $y(t) \leftarrow \hat{y}(t)$ for $t \geq t_k + \Delta_k$.
```

Fig. 8: Pseudocode for the contingency replanning algorithm.

query is deemed successful if, after time limit Δ_k^o, $C(y^o)$ is improved over the current optimistic path if it exists, or otherwise over the current pessimistic path. If the query fails, y^o is left untouched. Pessimistic queries are handled exactly as in the prior section.

To choose planning times, we again use an adaptive time stepping scheme using the exponential backoff strategy of Section 3.2. Pessimistic and optimistic planning times are learned independently. We also make adjustments in case the candidate time step exceeds the finite TTPF of our paths. First, if we find that Δ_k exceeds the TTPF T of the pessimistic path, that is, failure may occur before planning is complete, we set $\Delta_k^p = T/2$ and $\Delta_k^o = 0$. Second, if we are attempting a modification to the optimistic trajectory, and the TTPF of the optimistic trajectory T^o is less than $t_j + \Delta_k$, then we scale Δ_k^p and Δ_k^o to attempt a replan before T^o (otherwise, the pessimistic replan is guaranteed to fail).

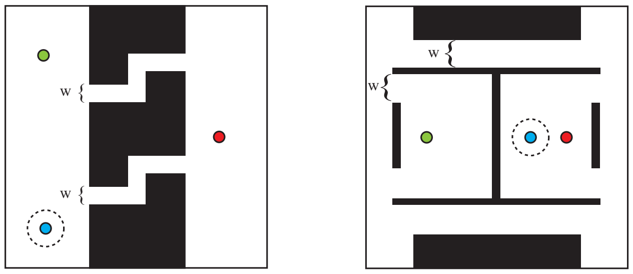

Fig. 9: Pursuit-evasion environments 1 and 2. Narrow passages, and hence, difficulty, are parameterized by w. The evader (green) must try to reach the target (red) within 10 s while avoiding the pursuer (blue), with capture radius 0.05.

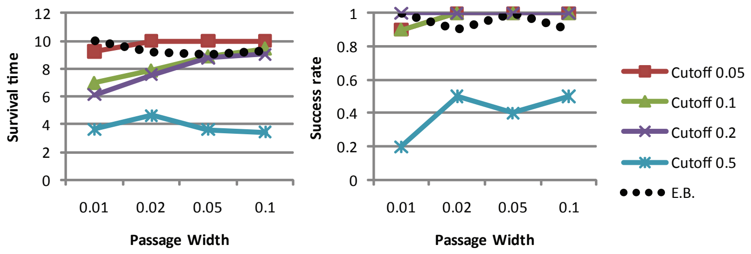

Fig. 10: (a) Survival time and (b) success rates for evader time-stepping strategies on problem 1. Results were averaged over ten trials on each passage width. The adaptive strategy (E.B.) performs as well as the best constant cutoff, and is more consistent across problem variations.

4.3 Experiments on a Pursuit-Evasion Example

Our experiments evaluate how contingency planning strategies affect an evader's performance in a planar pursuit-evasion scenario. The evader's goal is to reach a target within 10 s before being captured by a pursuer. The evader and pursuer move at maximum speeds 1 and 0.5, respectively. The evader treats the pursuer as an unpredictable obstacle with bounded velocity, and uses Algorithm 2 with $T_{min} = 1.0$. The evader's conservative model of $\tilde{F}(t, E_k)$ does not consider walls to be impediments to the pursuer's possible movement. The pursuer treats the evader as an unpredictably moving target, and uses Algorithm 1 to reach it.

Holding the pursuer's behavior constant, we varied the environment difficulty and evader's time stepping strategy on Problem 1 (Figure 9(a)). Here the pursuer begins in a room with the evader, which must escape through a narrow passage to reach the target in a second room. Figure 10 shows that narrow passage width does not affect the evader's survival much, but rather, responsiveness is more important to enable it to dance around an approaching pursuer. The adaptive time strategy appropriately finds short time steps.

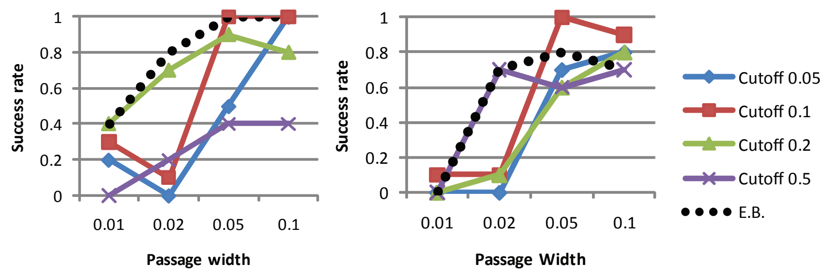

Fig. 11: Success rates for evader time-stepping strategies on problem 2 for a (a) nonadversarial and (b) adversarial pursuer behaviors. In the nonadversarial case the pursuer is allowed to pass through obstacles. A shorter time step (Cutoff 0.2) performs well in the nonadversarial case, but a longer time step (Cutoff 0.5) performs better in the adversarial case. The adaptive strategy works well in both cases.

Next, we considered a more difficult environment, Problem 2 (Figure 9(b)). Mere survival is not challenging (in all experiments it was over 90%), but reaching the target is; success requires the evader to choose a different hallway than the pursuer. We tested a nonadversarial pursuer behavior in which it "wanders" with velocity varying according to a random walk, and is allowed to pass through walls. Figure 11(a) shows that in this case, the success rate is highly dependent on problem difficulty, and no constant cutoff performs uniformly well across all width variations. Similar variations were found using an adversarial pursuer (Figure 11(b)). The adaptive strategy performed nearly as well as the best constant cutoff across all problem variations.

5 Conclusion

The runtime variance of planning queries has been a major impediment to the adoption of replanning techniques in real-time robot control. This paper addresses this problem by introducing two adaptive time-stepping algorithms – a simple one for deterministic environments, and a more complex one for nondeterministic environments – that tolerate run-time variance by learning a time step on-the-fly. Experiments on shared control for an industrial robot arm and on pursuit-evasion examples suggest that replanning may be a viable mechanism for real-time navigation and obstacle avoidance. Additional videos of our experiments can be found on the web at http://www.iu.edu/ motion/realtime.html.

References

1. F. Allgöwer and A. Zheng. *Nonlinear Model Predictive Control (Progress in Systems and Control Theory)*. Birkhäuser, Basel, 2000.
2. S. J. Anderson, S. C. Peters, K. D. Iagnemma, and T. E. Pilutti. A unified approach to semi-autonomous control of passenger vehicles in hazard avoidance

scenarios. In *Proc. IEEE Int. Conf. on Systems, Man and Cybernetics*, pages 2032–2037, San Antonio, TX, USA, 2009.

3. K. Bekris and L. Kavraki. Greedy but safe replanning under kinodynamic constraints. In *Proc. IEEE Int. Conference on Robotics and Automation (ICRA)*, pages 704–710, Rome, Italy, April 2007.
4. J. Bruce and M. Veloso. Real-time randomized path planning for robot navigation. In *IEEE International Conference on Intelligent Robots and Systems (IROS)*, Lausanne, Switzerland, October 2002.
5. E. Feron, E. Frazzoli, and M. Dahleh. Real-time motion planning for agile autonomous vehicles. In *AIAA Conference on Guidance, Navigation and Control*, Denver, USA, August 2000.
6. K. Hauser and V. Ng-Thow-Hing. Fast smoothing of manipulator trajectories using optimal bounded-acceleration shortcuts. In *Proc. IEEE Int. Conference on Robotics and Automation (ICRA)*, Anchorage, USA, 2010.
7. D. Hsu, R. Kindel, J.-C. Latombe, and S. Rock. Kinodynamic motion planning amidst moving obstacles. *Int. J. Rob. Res.*, 21(3):233–255, Mar 2002.
8. M. Kallmann and M. Mataric. Motion planning using dynamic roadmaps. In *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, Apr. 2004.
9. S. LaValle and J. Kuffner. Randomized kinodynamic planning. In *Proc. IEEE Intl. Conf. on Robotics and Automation*, pages 473–479, 1999.
10. M. Likhachev, D. Ferguson, G. Gordon, A. Stentz, and S. Thrun. Anytime dynamic a*: An anytime, replanning algorithm. In *In Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS*, 2005.
11. D. Q. Mayne, J. B. Rawlings, C. V. Rao, and P. O. M. Scokaert. Constrained model predictive control: Stability and optimality. *Automatica*, 36:789–814, 2000.
12. D. J. Musliner, E. H. Durfee, and K. G. Shin. Circa: A cooperative intelligent real-time control architecture. *IEEE Transactions on Systems, Man, and Cybernetics*, 23:1561–1574, 1993.
13. S. Petti and T. Fraichard. Safe motion planning in dynamic environments. In *IEEE International Conference on Intelligent Robots and Systems (IROS)*, pages 3726–3731, 2005.
14. I. Ross, Q. Gong, F. Fahroo, and W. Kang. Practical stabilization through real-time optimal control. In *American Control Conference*, page 6 pp., jun. 2006.
15. G. Sánchez and J.-C. Latombe. On delaying collision checking in PRM planning: Application to multi-robot coordination. *Int. J. of Rob. Res.*, 21(1):5–26, 2002.
16. C. Stachniss and W. Burgard. An integrated approach to goal-directed obstacle avoidance under dynamic constraints for dynamic environments. In *IEEE-RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, pages 508–513, 2002.
17. A. Stentz. The focussed d* algorithm for real-time replanning. In *In Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, 1995.
18. 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, May 2006.
19. J. van den Berg and M. Overmars. Roadmap-based motion planning in dynamic environments. *IEEE Trans. Robot.*, 21(5):885–897, October 2005.
20. M. Zucker, J. Kuffner, and M. Branicky. Multipartite rrts for rapid replanning in dynamic environments. In *Proc. IEEE Int. Conf. Robotics and Automation*, April 2007.