Intl. Conf. on Robotics and Automation, IEEE press, Rome, Italy, p. 704–710, 2007

Greedy but Safe Replanning under Kinodynamic Constraints

Kostas E. Bekris Lydia E. Kavraki

***Abstract*— We consider motion planning problems for a vehicle with kinodynamic constraints, where there is partial knowledge about the environment and replanning is required. We present a new tree-based planner that explicitly deals with kinodynamic constraints and addresses the safety issues when planning under finite computation times, meaning that the vehicle avoids collisions in its evolving configuration space. In order to achieve good performance we incrementally update a tree data-structure by retaining information from previous steps and we bias the search of the planner with a greedy, yet probabilistically complete state space exploration strategy. Moreover, the number of collision checks required to guarantee safety is kept to a minimum. We compare our technique with alternative approaches as a standalone planner and show that it achieves favorable performance when planning with dynamics. We have applied the planner to solve a challenging replanning problem involving the mapping of an unknown workspace with a non-holonomic platform.**

I. INTRODUCTION

As automobiles and other mobile platforms achieve a higher degree of autonomy, generating safe and effective motions for autonomous vehicles emerges as a great application for motion planning. In realistic tasks, however, vehicles have only partial information about their environment. Solving such problems requires interleaving sensing, planning and execution, where a planner is called frequently and has finite time to replan a trajectory [1]–[3]. Moreover, vehicles exhibit kinodynamic constraints that restrict their motion, which must be accounted for at the motion planning stage so that a planned trajectory can be followed. In this paper, a tree-based planner is presented that respects such constraints and generates safe paths under finite computation times. Safety means that the vehicle does not collide with obstacles even when the configuration space of the robot changes. This is an important consideration when replanning for real vehicles as a collision-free trajectory may bring the system close to an obstacle with high-velocity and no maneuver to avoid collisions [4], [5].

For replanning applications, the computational performance of the planner is of primary importance. A slow planner delays taking into account new sensing information and the vehicle does not react on time to changes in the workspace. In the proposed framework, the planner itself selects the duration of the planning cycle, which allows the implementation of a lazy evaluation approach to reduce the overhead of collision checking. It also uses computations executed during previous planning periods and an effective state space exploration scheme to bias its search. This paper also describes the application of the planner to a task that requires replanning: mapping of unknown environments. Initially the vehicle knows only a small part of the space but it must eventually cover the entire workspace. Fig. 1 shows an example from our simulations with a car-like robot with bounded acceleration. We have experimented with various mobile platforms on different scene types and compared the approach as a standalone planner against alternative planners. We present results that demonstrate the algorithm's favorable performance and its effectiveness in replanning under kinodynamic constraints, limited computation time and a dynamic representation of the workspace.

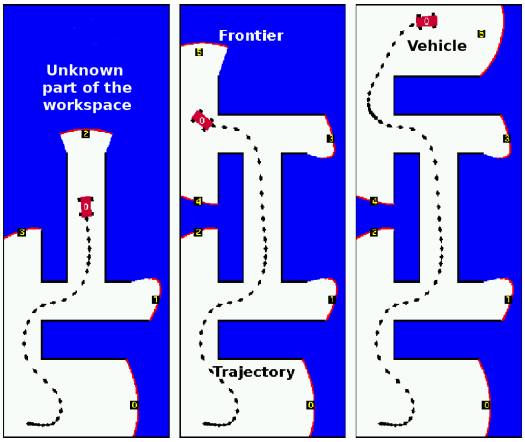


Fig. 1. Mapping an unknown space with an acceleration controlled car.

A. Related Literature

Sampling-based planners have proved effective in dealing with high-dimensional problems [6] and, in particular, tree-like planners [7], [8], have been successful dealing with dynamic constraints. Especially, the Path-Directed Subdivision Tree (PDST) planner [9] has shown good performance in such planning instances and allows biasing the search of the algorithm while providing probabilistic completeness.

Typical tree-based planners assume complete workspace knowledge. Hsu et al. [8] have experimented with a real robot that navigates among moving obstacles by replanning from scratch. Bruce and Veloso [1] describe ways of using previous calls to the RRT algorithm to bias the exploration of a new tree. Ferguson et al. [2] repair RRTs while replanning by creating a probabilistic analog to the D* family of deterministic replanning methods [10]. Van den Berg et al. [3] create a roadmap that covers the planning space and later

Work on this paper has been supported in part by NSF 0308237, NSF0205671, and a Sloan Fellowship to LEK. Experiments reported in this paper has been obtained on equipment supported by NSF CNS 0454333, NSF CNS 0421109 in partnership with Rice University, AMD, and Cray.

Kostas E. Bekris and Lydia E. Kavraki are with the Computer Science Department, Rice University, Houston, TX, 77005, USA {bekris,kavraki}@rice.edu

replan using this graph. Van den Verg and Overmars [11] provide algorithms for computing shortest safe paths amidst moving obstacles. Safety issues related to replanning and kinodynamic constraints arise when collision-free states are Inevitable Collision States (ICS). The notion of ICS has been employed in the partial motion planning framework of Fraichard et al. [4], [12]. Frazolli et al. [5] introduced the notion of τ-safety and used RRTs that guaranteed states along the produced tree were τ-safe.

B. Contribution

Our goal is to combine the above contributions with new ideas to define a single, efficient framework for replanning under kinodynamic constraints. In particular:

(a) We follow an incremental approach similar to methods that repair RRTs [1], [2], but we deal explicitly with kinodynamic constraints and safety issues.

(b) The planner uses the ICS formalization [4] and provides safety guarantees similar to τ-safety [5]. It reduces, however, the cost of achieving safety by controlling the duration of the planning cycle and employing lazy evaluation [13]. In this way, the number of states that must be checked for safety are reduced, resulting in considerable speedups as our experiments show.

(c) A state space exploration scheme similar to PDST [9] is applied, but the search is biased using information easily extracted directly from sensor data. Our experiments show that the proposed algorithm performs favorably against the "Voronoi-bias" selection strategies of RRTs given the same metrics [7].

In the following section we formally describe the problem that the proposed motion planner solves. A detailed description of the planer, called GRIP for GReedy, Incremental, Path-directed planner, is provided in section III. Section IV illustrates the application of the planner to a mapping task with non-holonomic robots. The setup for our experiments and our results are presented in sections V and VI.

II. PROBLEM OVERVIEW

Assume a drift-less non-holonomic robot whose motion is governed by: $\dot{q} = f(q, u)$ and $g(q, \dot{q}) \leq 0$, where $q \in Q$ is a state, u is a control and f, g are smooth. Q is the state space of the robot with a metric $\rho(q_1, q_2)$, where $q_1, q_2 \in Q$. The robot is in a workspace $\mathcal{W}$, equipped with a sensor of limited range. The robot uses its sensors to update an evolving workspace representation $s(\mathcal{W}, t)$. A state $q \in Q_{free}$ is considered collision-free at time t if it places the robot chassis in the known collision-free part of $s(\mathcal{W}, t)$.

Our framework is applicable to multiple tasks where the workspace changes dynamically. We focus on the case the robot does not know anything about the workspace and must cover it in order to build a map. A similar dynamic task not covered here due to space limitations is planning among dynamic obstacles. Our approach towards these high-level tasks is to break them into a sequence of smaller planning problems. To achieve this, we synchronize the mapping unit, the planner and the motion controller as shown by Fig. 2.

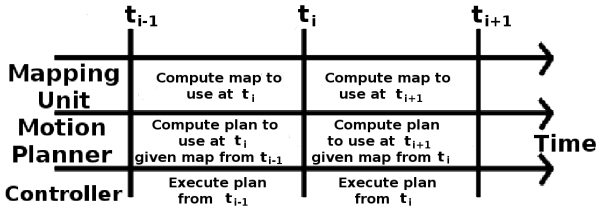


Fig. 2. The robot's synchronization scheme.

Then for the planning cycle $(t_{i-1} : t_i)$, the following steps are executed:

- A representation $s(\mathcal{W}, t_{i-1})$ built in the previous cycle.
- A target $F_{t_{i-1}} \subset Q$ is defined for the current cycle.
- A motion planner computes a plan.
- The plan will be executed during cycle $(t_i : t_{i+1})$.

A plan is a time-sequence of controls: $p(dt) = \{(u^1, dt^1), \ldots, (u^n, dt^n)\}$ where $dt = \sum_i dt^i$. When a plan $p(dt)$ of duration dt is executed at state q, it defines a trajectory $tr(q, p(dt))$, which is the sequence of states propagated according to $\dot{q} = f(q, u)$. A trajectory that respects $g(q, \dot{q}) \leq 0$ is called feasible, and if it lies in Q_{free} is collision free. The motion planner solves the following problem:

Replanning under Kinodynamic Constraints: Given:

- the latest representation $s(\mathcal{W}, t_{i-1})$
- a set of target states $F_{t_{i-1}} \in Q$
- the initial robot state q at time t_i
- the tree structure from the previous cycle $(t_{i-2} : t_{i-1})$.

Compute plan $p(t)$ to execute during $(t_i : t_{i+1})$ that produces $tr(q, p(t))$ which is: (a) collision-free, (b) leads to $F_{t_{i-1}}$, (c) minimizes the distance traveled by the robot and (d) does not lead to Inevitable Collision States [4] past t_{i+1}.

III. THE PLANNER: GRIP

The GRIP (Greedy, Incremental, Path-directed) planner expands greedily and incrementally a tree data structure in the state-time space and return safe paths for kinodynamic systems. The planner contains: **1)** A retainment step that reuses part of the previous tree (III-A). **2)** An exploration strategy that achieves probabilistic completeness but allows greedy search (III-B). **3)** Safety checking so that the selected plans do not lead to inevitable collisions (III-C).

A. Tree Retainment

The algorithm uses a planning horizon longer than the duration of the next planning cycle, e.g. $(t_i : t_{i+1})$. This implies that a large part of the tree constructed during the previous planning cycle may still be valid and it can be used to accelerate the search for a new path given new sensor data similarly to Dynamic RRTs [2]. Note, however, that kinodynamic constraints add an additional limitation, since it is not possible to go backwards along the tree. The subtree of the initial robot state q for the new planning cycle is valid for planning and everything above it is unreachable and must be discarded. Trimming other parts of the tree that are invalid due to unexpected collisions can similarly be executed. Moreover, if a path to the desired target exists in the remaining tree, every path that does not lead to the

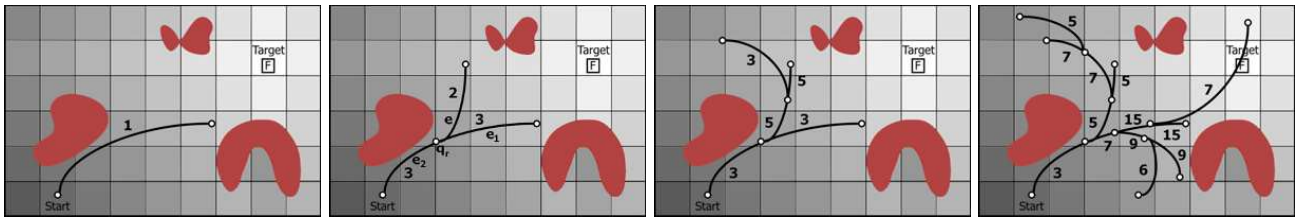


Fig. 3. An illustration of GRIP's exploration. A discrete potential function is used for $\rho(e, F)$. The numbers denote edge priorities $p(e)$. The second figure shows an edge broken into: e'_1 and e'_2, while a new edge e is propagated from state q^r. The last figure shows the final tree that reaches the target.

target can also be pruned. In this way, when a path has been found, the technique focuses on obtaining plans of increasing quality. There are also computations during tree retainment that are related to state-space exploration and plan safety, which will be described in the following sections.

B. Selection/Propagation: Completeness with Bias

We follow the selection/propagation scheme for exploring the state-space, typical for tree-based planners [7]. A state from the tree is selected for propagating a trajectory forward in time. Our approach follows the PDST algorithm [9], which deterministically biases the search, so as to search interesting parts of the state space faster. Although the search is biased, the approach is still probabilistically complete.

The basic sampling module is a path, which corresponds to an edge of the tree data structure. In this way, all the states along the tree are candidates for propagation. The algorithm first deterministically selects an edge e' to expand from. The actual state q^r used for propagation is randomly sampled along the edge e'. Edge e' is then broken into two parts e'_1 and e'_2 by creating a node at the selected state q^r. For the new edge e, a random plan $p(t_{max})$ of maximum duration t_{max} produces a trajectory $tr(q^r, p(t_{max}))$. This trajectory is checked for collisions so as to calculate t_{free}, the last point in time that the trajectory is collision-free. Figure 3 provides an illustration of the selection/propagation procedure.

The selection of edge e' is based on a combination of metric information to greedily bias the search towards the target and a priority scheme that guarantees that eventually all the edges will be selected for propagation. For a new edge e, states along e are sampled. For each state, we compute the distance to the target set F, so as to get a distance estimate between the edge and the target:

$$\rho(e, F) = \{min(\rho(q, f)),\ \forall q \in e \text{ and } \forall f \in F\}. \quad (1)$$

Edges also have priority counters. A newly expanded edge e has priority equal to the iteration counter: $p(e) = i$, where i is the number of edges propagated before e in the current cycle. The two parts of the old edge e'_1 and e'_2 get a lower priority that decreases exponentially when the same sample is selected for propagation, e.g.:

$$p(e'_1) = p(e'_2) = 2 \cdot p(e') + 1. \quad (2)$$

The overall score of an edge is:

$$s(e) = \alpha^{p(e)} \cdot \rho(e, F). \quad (3)$$

where parameter α is greater than 1: $\alpha > 1$ and controls the importance of the bias vs. the priority counters. The edges are stored in a heap that returns the minimum score edge.

The scoring function must also be recomputed during tree retainment. The distance estimate $\rho(e, F)$ is updated for all retained edges given the newly acquired representation: $s(\mathcal{W}, t_{i-1})$. An advantage of the approach, is that with the use of simple metrics instead of path subdivision as in PDST [9], this update is inexpensive. The priority counters of retained edges, which are children of the initial robot state, are set to 1 and the priority counters along the remaining tree are recursively set by Eq. 2. This choice tends to produce increasingly smoother paths towards the goal during consecutive planning cycles, when a solution path has already been found.

Notice that instead of random controls, propagation could be achieved with controls selected from an appropriate set specifically constructed for the system (e.g., maneuver automata [14], Reeds and Shepp curves [15] etc.).

C. Efficient Safety Checking

The fact that $tr(q^r, p(t_{free}))$ is collision-free does not guarantee safety, since $tr(q^r, p(t_{free}))$ may lead to an Inevitable Collision State (ICS) [4]. Computing whether a state is ICS is difficult, since it requires to consider the set of all possible future trajectories. Taking a conservative approach, however, a superset of ICS can be computed much faster by using only a small set of "contingency" plans Γ and define a state q to be **unsafe** iff:

$$\nexists\ \gamma \in \Gamma \text{ s.t.: } tr(q, \gamma(t)) \text{ is collision free.} \quad (4)$$

Examples of contingency plans for static workspaces are breaking maneuvers that bring the robot to a complete stop. The time it takes to execute the contingency plan is unrelated to the duration of the planning cycle. Note that trajectories leading to safe states are also safe. Fraichard et al. [12] and Frazolli et al. [5] used this property to reduce the overhead of safety checking. We can use the fact that the planner can select the duration of the consecutive planning cycle to further reduce the overhead of safety checking. In this way, only states that are reached at the end of the next planning cycle have to be checked for safety.

Theorem: Assume a drift-less non-holonomic system executing a replanning task with GRIP in a static environment. The planner selects duration T for the next planning cycle. It is then sufficient to guarantee collision avoidance to produce trajectories during the current cycle that are safe only for time

T, given contingency plans that are breaking maneuvers.

Proof Sketch: Assume the vehicle is safe at time t_{i-1} and has a plan $p(t)$ of duration $t > T$ that is safe at least for T. The question is whether reaching `ICS` can be avoided while replanning. There are two cases: (a) The planner will either succeed in producing a new safe plan for the next planning period and at time t_i the robot will again have a collision-free plan to follow. (b) The planner fails to compute a plan for the next period. However, the previous plan $p(t)$ was safe for at least T, which means that there is at least one plan $\gamma(t) \in \Gamma$ that can be executed after $p(t)$, which is collision-free and brings the robot to a complete stop in a collision-free configuration. So, in every case there is a collision-free plan. □

The contingency plans do not have to be breaking maneuvers. Looping maneuvers would work as well, given the assumptions. If we provide τ-safety guarantees in the case of moving obstacles, which means that collision avoidance is guaranteed only for a time period τ after the failure of the algorithm, then we can use any plan of duration τ.

Consequently, the planner does not check for safety past time T. During propagation of $tr(q^r, p(t_{free}))$, we compute whether it intersects time T at a state q'. If it does not or if q' is safe given set Γ, then the complete $tr(q^r, p(t_{free}))$ is added as a new edge. Otherwise, the part of the trajectory past q' is pruned, so that all the trajectories stored in the tree are safe at least for time T. A possible drawback of the approach is that the robot might start following a path that is not safe past time T. This is not a safety risk since the unsafe part of the trajectory will be pruned during tree retainment when it will fall within the planning period T. It could be an undesired effect, however, since it could result in often selection of contingency plans. But we can check for safety only the solution trajectories and accept them as solutions only if they are completely safe, following the lazy evaluation approach, where a path is checked for validity only after it is considered as a good candidate for a solution [13].

D. `GRIP`

Algorithm 1 provides an overview of the complete `GRIP` algorithm. The main loop selects greedily the edge that has the maximum score and expands the tree from a state along the selected edge.

An implementation choice is related to the metric $\rho(q_1, q_2)$. A possible solution could be the cost-to-go function in an obstacle free space, which can be computed with dynamic programming as described by Frazzoli et al. [5]. Lavalle and Kuffner [7] used a weighted Euclidean metric. In our implementation, we use an occupancy grid to represent the workspace and define the metric $\rho_{A^*}(q_1, q_2)$ using the A* holonomic distance between the coordinates of the two states q_1 and q_2. Although $\rho_{A^*}(\cdot)$ carries poor distance information for high-dimensional problems, the planner achieves good performance for the dynamically constrained platforms we have experimented with and outperforms `RRTs` when the same metric is used.

Algorithm 1 `GRIP` ($tree$, q, $s(\mathcal{W}, t_{i-1})$, F)

TREE RETAINMENT

`(initialization)`

Compute duration T of consecutive planning cycle
Set $tree \rightarrow root = q$ and prune everything before it
`(safety check for retained tree)`
for each edge $e \in tree$ **do**
  if e contains state q' reachable from q a time T **then**
    extend contingency trajectory $tr(q', \gamma(t))$
    if $tr(q', \gamma(t))$ is not collision-free **then**
      Prune everything below q'
    end if
  end if
end for
`(focus on smoothing if path exists)`
if a path to F exists in $tree$ **then**
  Prune every path on the tree that does not lead to F
end if
`(update scores for existing edges)`
Set the priority of edges that are children of q to 1.
Recursively update the priorities of edges as in Eq. 2.
Initialize an empty $heap$
for all the edges $e \in tree$ **do**
  compute $\rho(e, F)$ for the new target F given Eq. 1
  update $s(e)$ according to Eq. 3
  add e in $heap$
end for

Main loop: SELECTION-PROPAGATION

while time is less than t_i **do**
  `(deterministic selection)`
  Remove from $heap$ edge $e' = argmin(s(e))\ \forall\ e \in heap$
  `(random propagation)`
  Select random state q^r along e'
  Break edge e' at state q^r into edges e'_1 and e'_2
  Select random plan $p(t_{max})$ for a planning duration t_{max}
  Create trajectory $tr(q^r, p(t_{max}))$
  Compute duration t_{free} that $tr(q^r, p(t_{max}))$ is free
  `(safety check)`
  if $tr(q^r, p(t_{free}))$ contains q' reachable from q a time T **then**
    extend contingency trajectory $tr(q', \gamma(t))$
    if $tr(q', \gamma(t))$ is not collision-free **then**
      Prune everything below q' and adjust t_{free}
    end if
  end if
  `(priority scheme)`
  if t_{free} is above a minimum threshold **then**
    Create edge e from the remaining part of $tr(q^r, p(t_{free}))$
    set priorities: $p(e) = i{+}{+}$ and $p(e'_1) = p(e'_2)$ as in Eq. 2.
    compute $\rho(\cdot, F)$ and scores $s(\cdot)$ for e, e'_1, e'_2
    add e, e'_1, e'_2 in $heap$
  end if
end while
return $tree$

IV. APPLICATION TO MAPPING

This section describes how GRIP can solve the task of mapping a workspace while respecting dynamics.

1) Workspace representation and collisions: An occupancy grid is used that has 3 values: explored free space, obstacle and unexplored space. Given a state, the robot is positioned on the map and if the chassis intersects an obstacle or an unexplored space cell, it is in collision. Figure 1 shows an example of mapping in our simulator.

2) Selection of target F for the planner: We follow a frontier-based approach to select a target for the motion planner at each cycle. A frontier is the boundary between the free explored space and the unexplored one. Neighboring frontier cells are grouped by applying a flooding operation on the grid and the search is biased towards a particular frontier during each planning period. There are many alternative heuristic approaches for selecting target frontiers [16]. In our implementation, we select frontiers that are: (a) close to the initial robot state given ρ_{A*} and (b) small in size, to avoid returning to small unexplored regions after covering large distances. Figure 4 shows the A* distance on the grid map from a frontier and a tree expanded towards the frontier.

3) Path selection given a GRIP *tree:* The objective in path selection is to maximize visibility of the unexplored space and minimize the length of the trajectory. Candidate trajectories are all those that initiate from the root to a node of the constructed tree. A weight $w(tr)$ for every candidate trajectory tr is defined as:

$$w(tr) = e^{-(d(tr)+\lambda \cdot l(tr))}$$

where $d(tr)$ describes how close the trajectory is to the selected frontier and $l(tr)$ expresses the trajectory length. Parameter λ expresses the importance of the path length over the distance to the frontier. There are two distinct cases, however, for the distance parameter $d(tr)$ depending on whether the algorithm has managed to produce states that can sense the frontier or not. In the first case, for a trajectory tr and a cell c in the target frontier group we define the distance as:

$$d(tr,c) = \begin{cases} \rho_{A*}(q,c) & \text{if } \exists\ q \in tr \text{ s.t. } c \text{ is visible from } q \\ d_{max} & \text{otherwise} \end{cases}$$

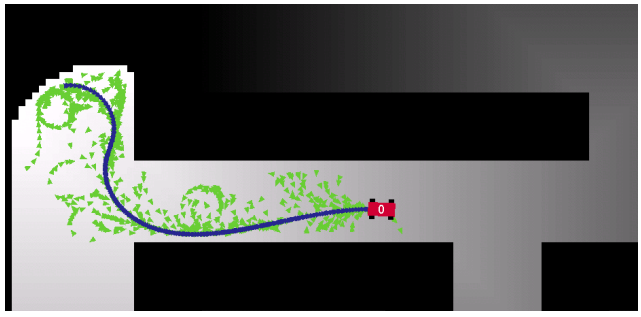


Fig. 4. Biased tree expansion. The A* distance used to bias the search is shown in the background. The light colored triangles correspond to vehicle configurations. The darker trajectory is the selected path.

Differential Drive

$$\begin{pmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \\ \dot{V}_L \\ \dot{V}_R \end{pmatrix} = \begin{pmatrix} \cos\theta \cdot \frac{R}{2} \cdot (V_L + V_R) \\ \sin\theta \cdot \frac{R}{2} \cdot (V_L + V_R) \\ \frac{R}{2 \cdot L} \cdot (V_R - V_L) \\ \alpha_L \\ \alpha_R \end{pmatrix} \quad \begin{array}{l} |V| \leq 3\frac{m}{s} \\ |\dot{\theta}| \leq 20\frac{deg}{s} \\ |\alpha| \leq 0.6\frac{m^2}{s} \\ |\ddot{\theta}| \leq 3\frac{deg^2}{s} \end{array}$$

Car-like

$$\begin{pmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \\ \dot{V} \\ \dot{s} \end{pmatrix} = \begin{pmatrix} \cos\theta \cdot \cos s \cdot V \\ \sin\theta \cdot \cos s \cdot V \\ \sin s \cdot v \\ \alpha \\ t \end{pmatrix} \quad \begin{array}{l} -0.5\frac{m}{s} \leq V \leq 3\frac{m}{s} \\ |s| \leq 4\frac{deg}{s} \\ \alpha \leq 0.6\frac{m^2}{s} \\ |t| \leq 1\frac{deg^2}{s} \end{array}$$

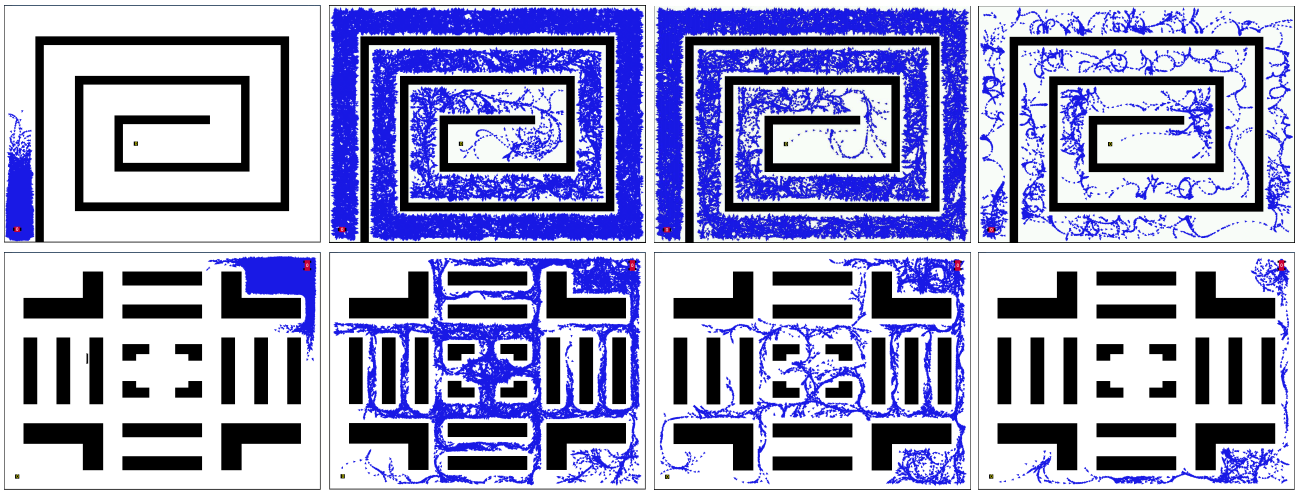
TABLE I
STATE UPDATE EQUATIONS AND CONTROL LIMITS.

where d_{max} is the sensing radius of the robot. Then $d(tr) = \sum_{\forall c} d(tr,c)$. In this way, trajectories that see a large number of frontier cells and which are closer to them have a smaller distance parameter. If there is no state along the tree able to sense the frontier, we define $d(tr)$ as the minimum distance between the end state of the trajectory and a frontier cell according to ρ_{A*}. Both $l(tr)$ and $d(tr)$ can be computed recursively during the tree construction. The trajectory of maximum weight that has duration at least T is finally returned. If no such trajectory exists, a collision-free contingency plan is guaranteed to exist by the theorem.

V. EXPERIMENTAL SETUP

We have experimented with three systems: **(1)** A differential drive robot (DD-robot) with velocity controls V_L, V_R. This platform is reducible to a simpler holonomic robot, since we can retain the entire tree at each time step and the contingency plan is trivial: $V_L = V_R = 0$. **(2)** A DD-robot with acceleration controls α_L, α_R. The contingency plan is selected so that the wheel with the largest velocity magnitude is assigned maximum de-acceleration. The second wheel's acceleration is set so that its velocity reaches zero as the same time the first wheel stops. **(3)** A car-like robot that moves backwards and forwards with acceleration control α and steering velocity t. The contingency plan sets the de-acceleration parameter to its maximum value so as to reach a configuration with zero forward and steering velocities. Table I provides the state update equations for the last two systems and the bounds we have used for the controls. Parameters R and L are the radius of the wheel and the distance between a wheel and the robot's center. The three scenes we have used for our experiments can be seen in Figure 8. The total area that the robot must sense in all of the scenes is comparable. The sensing radius of the robot is equal to one tenth the width of the scenes.

The simulation component of our program is responsible for updating the map and transmitting it over socket communication to the planner. GRIP is executed on a different processor than the simulator and after the computation of a plan the planner communicates a sequence of controls back to the simulator. The planner was tested on an Athlon 1900MPs with one gigabyte of RAM.



(e) (f) (g) (h)

Fig. 5. Goal finding in (top row) scene meandros with a 2nd order DD-robot and (bottom row) scene labyrinth with an acceleration-bounded car-like robot: (a-e) a trivial random tree does not find the target after 100,000 iterations, (b-f) an `RRT-EXTEND` selection strategy finds the target after (top) 48,410 iterations and (bottom) 51,245 iterations (c-g) `RRT-EXTEND-BIAS`, where 20% of the time the target is the attractor, finds the target after (top) 42,855 iterations and (bottom) 17,212 iterations (d-h) `GRIP` reaches the target after (top) 13,774 edges and (bottom) 4,363 respectively.

VI. RESULTS

This sections summarizes experiments conducted with `GRIP` for the three non-holonomic platform.

A. `GRIP` *as a stand-alone planner*

The first set of experiments corresponds to typical motion planning problems with dynamics that do not make use of replanning, so as to compare against the "Voronoi-bias" selection strategies of `RRT`. In these experiments, the same programming infrastructure and parameters have been used but different selection strategies are tested. Figure 5 displays the resulting trees for different selection strategies. Figure 6 provides averages over 10 experiments in these two scenes. All strategies are using the same metric $\rho_{A^*}(\cdot)$. A trivial random selection policy fails to produce any path after 100,000 edges have been added to the tree. In order to implement the Voronoi-bias approach, we randomly sample points in the free part of the workspace and use ρ_{A^*} to select the closest edge to them. The `RRT` approach offers good coverage of the state space but it is slow in reaching the target configuration. We have experimented with a version of `RRT` that is biased to promote exploration towards the target. In this version, 20% of the time the state that is used to select the closest edge is a state in the target set. The value 20% gave the best results over different scenes. Although there is an improvement compared to the strictly exploring version of the `RRT` algorithm, the approach is still slow in reaching the goal configuration. On the other hand, the `GRIP` algorithm manages to aggressively search the state space towards the goal configuration. Considering the poor quality of the metric used, this is a positive result. This behavior was consistent across all experiments.

Figure 7 shows the benefits of replanning with a selected duration for the next planning cycle. We have compared our algorithm that tests for safety only states that are T away from the root node of the tree with an approach that

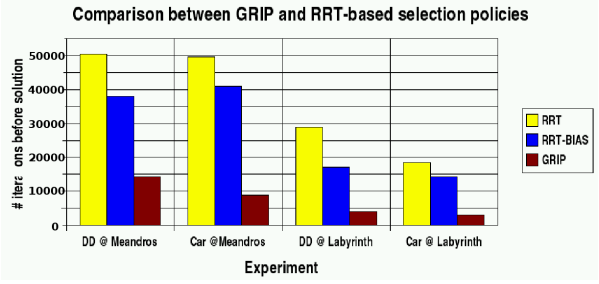


Fig. 6. Comparison between the `GRIP` selection/propagation scheme and Voronoi biased selections.

Time	DD-velocity	DD-acceleration	Car-like
Average Time in secs.	0.39	0.63	0.66
Maximum Time in secs.	0.90	1.57	1.19

TABLE II

AVER. COST IN SECONDS TO PRODUCE 250 EDGES.

produces a tree where all the leaf nodes are safe. If the two approaches are provided with the same planning period, then `GRIP` produces a much bigger tree, which allows the planner to better search the state-time space. The difference in the tree size is mainly due to the additional collision checking necessary to provide safety in the second case.

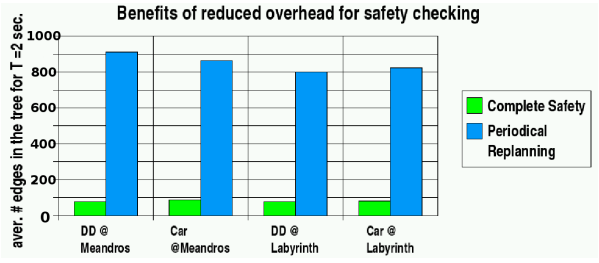


Fig. 7. Replanning with a known duration reduces the overhead of guaranteeing safety. For the same planning period `GRIP` builds bigger trees.

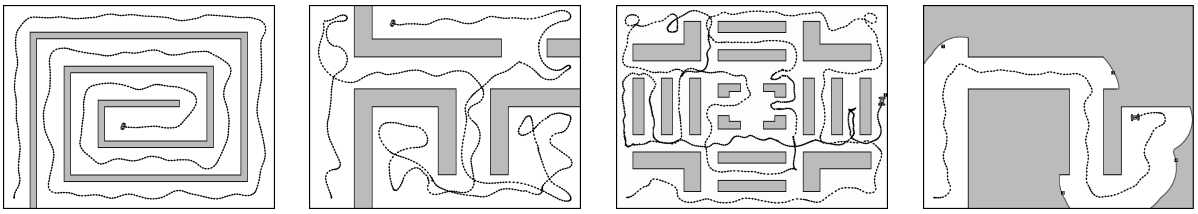


Fig. 8. Exploration of scenes (from left to right) “meandros”, “rooms” with a DD-robot, “labyrinth” and “rooms” again with a car-like robot.

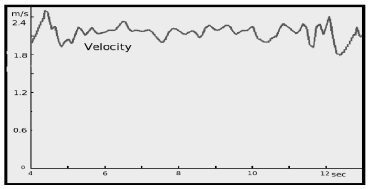


Fig. 9. The velocity profile for the car exploring “rooms” in Figure 8(d).

B. Performance for high-level tasks: mapping

Figure 8 provides a qualitative evaluation of the exploration paths produced by `GRIP`. The robot is initially positioned at the bottom left corner of a scene and knows only the part of the environment that it can sense. No collision was observed during our experiments. If the `ICS` avoidance step is removed from the planner, however, then the vehicle collides within a few seconds of execution. The paths appear smooth and the robots do not unnecessarily revisit parts of the space that are already covered. Figure 9 displays a velocity profile for an exploration procedure. The robot velocity remains for a large duration of the exploration procedure close to its maximum value and does not fluctuate considerably. Table II presents computational performance statistics. We ran 10 experiments for each robot and scene type and measure the time it takes for the planner to compute trees with 250 edges. We present: (a) the average time, (b) and the maximum time, that the planner requires to produce the tree. As expected systems with bounded acceleration are more difficult to plan for.

VII. DISCUSSION AND EXTENSIONS

This paper describes a tree-based planner for replanning under dynamic constraints for tasks with partial observability. The algorithm provides safety guarantees for collision avoidance even under limited computation time. This work provides a general framework for such applications by extending previous work on kinodynamic planning [2], [4], [5], [8], [9], [17] and uses new ideas to achieve good performance. An efficient selection/propagation strategy manages to bias state space exploration with the aid of simple metrics. The planner reduces the amount of collision checking necessary for providing safety guarantees and reuses computations from previous cycles. The simulated experiments suggest favorable computational performance against popular alternatives and returns smooth, safe paths.

An important extension is to take into account sensor noise and positioning error. One of the problems in this case is that the robot’s initial state q during each cycle does not necessarily lie on the previous tree. To achieve tree retainment, the initial robot state must properly be reconnected with the existing tree. Moreover, the notion of collision safety is no longer deterministic but each state has an associated probability of whether it collides with an obstacle. We are actively researching extensions of the framework to address planning under uncertainty without sacrificing the nice properties of high-speed safe motion.

REFERENCES

[1] J. Bruce and M. Veloso, “Safe multi-robot navigation within dynamic constraints,” *Proc. of the IEEE*, vol. 94(7), pp. 1398–1411, 2006.

[2] D. Ferguson, N. Kalra, and A. Stentz, “Replanning with rrts,” in *IEEE ICRA*, May 2006, pp. 1243–1248.

[3] J. v. d. Berg, D. Ferguson, and J. Kuffner, “Anytime path planning and replanning in dynamic environments,” in *IEEE ICRA*, May 2006, pp. 2366–2371.

[4] T. Fraichard and H. Asama, “Inevitable collision states - a step towards safer robots?” *Advanced Robotics*, vol. 18(10), pp. 1001–1024, 2004.

[5] E. Frazzoli, M. A. Dahleh, and E. Feron, “Real-time motion planning for agile autonomous vehicles,” *Journal of Guidance, Control and Dynamics*, vol. 25, no. 1, pp. 116–129, 2002.

[6] L. E. Kavraki, P. Svestka, J.-C. Latombe, and M. Overmars, “Probabilistic roadmaps for path planning in high-dimensional configuration spaces,” *IEEE TRA*, vol. 12, no. 4, pp. 566–580, Aug. 1996.

[7] S. M. LaValle and J. J. Kuffner, “Randomized kinodynamic planning,” *IJRR*, vol. 20, no. 5, pp. 378–400, May 2001.

[8] D. Hsu, R. Kindel, J.-C. Latombe, and S. Rock, “Randomized kinodynamic motion planning with moving obstacles,” *IJRR*, vol. 21, no. 3, pp. 233–255, 2002.

[9] A. M. Ladd and L. E. Kavraki, “Fast tree-based exploration of state space for robots with dynamics,” in *WAFR*, 2005, pp. 297–312.

[10] A. Stentz, “The focussed d* algorithm for real-time replanning,” in *IJCAI*, August 1995, pp. 1652–1659.

[11] J. v. d. Berg and M. Overmars, “Planning the shortest safe path amidst unpredictably moving obstacles,” in *WAFR*, July 2006.

[12] S. Petti and T. Fraichard, “Partial motion planning framework for reactive planning within dynamic environments,” in *AAAI Intl. Conf. ICAR*, Barcelona, Spain, September 2005.

[13] R. Bohlin and L. E. Kavraki, “Path planning using lazy prm,” in *IEEE ICRA*, San Fransisco, CA, April 2000, pp. 521–528.

[14] E. Frazzoli, M. A. Dahleh, and E. Feron, “Maneuver-based motion planning for nonlinear systems with symmetries,” *IEEE TR*, vol. 21, no. 6, pp. 1077–1091, December 2005.

[15] J. A. Reeds and L. A. Shepp, “Optimal paths for a car that goes both forwards and backwards,” *Pacific J. Math.*, vol. 145, no. 2, pp. 367–393, 1990.

[16] W. Burgard, M. Moors, C. Stachniss, and F. Schneider, “Coordinated multi-robot exploration,” *IEEE TR*, vol. 21, no. 3, 2005.

[17] S. LaValle and J. Kuffner, “Rapidly exploring random trees: Progress and prospects,” in *WAFR*, 2001, pp. 293–308.