Sampling-based Motion and Symbolic Action Planning with Geometric and Differential Constraints

Erion Plaku Gregory D. Hager
Department of Computer Science, Johns Hopkins University
3400 N. Charles Street, Baltimore, MD 21211
{erion, hager}@cs.jhu.edu

***Abstract*— To compute collision-free and dynamically-feasibile trajectories that satisfy high-level specifications given in a planning-domain definition language, this paper proposes to combine sampling-based motion planning with symbolic action planning. The proposed approach, Sampling-based Motion and Symbolic Action Planner (SMAP), leverages from sampling-based motion planning the underlying idea of searching for a solution trajectory by selectively sampling and exploring the continuous space of collision-free and dynamically-feasible motions. Drawing from AI, SMAP uses symbolic action planning to identify actions and regions of the continuous space that sampling-based motion planning can further explore to significantly advance the search. The planning layers interact with each-other through estimates on the utility of each action, which are computed based on information gathered during the search. Simulation experiments with dynamical models of vehicles carrying out tasks given by high-level STRIPS specifications provide promising initial validation, showing that SMAP efficiently solves challenging problems.**

I. Introduction

Research in robotics has focused since its inception towards increasing the ability of robots to plan and act on their own in order to complete assigned high-level tasks. Toward this goal, this paper studies the following problem:

> Given a high-level specification, plan the sequence of motions the robot needs to execute so that the resulting trajectory is dynamically feasible, avoids collisions, and satisfies the high-level specification.

There are two crucial aspects to this problem: (i) planning in the space of possible high-level actions and (ii) planning in the space of possible motions.

Action planning, which assumes a discrete world and discrete actions, has been extensively studied in AI. Throughout the years, significant progress has been made in addressing increasingly complex discrete planning problems. In fact, current methods based on symbolic reasoning have made it possible to specify high-level goals using sophisticated planning-domain languages, such as STRIPS [1], ADL [2], PDDL [3], HAL [4], and efficiently plan the sequence of discrete actions that accomplishes the specified goals [5].

In contrast, motion planning assumes a continuous world and continuous motions. The motivation comes from navigation, exploration, search-and-rescue missions, and other applications where it is essential to compute trajectories that can be followed by the robot in the physical world. As a result, the planned motions need to not only avoid collisions with obstacles but also satisfy differential constraints imposed by underlying robot dynamics. Due to the increased complexity, motion planning has generally been limited to simpler goal specifications, such as reachability, where the objective is to compute a collision-free and dynamically-feasible trajectory from an initial to a goal state [6], [7].

Researchers have generally considered action planning and motion planning separately. As a result, the problem of planning motions that satisfy high-level specifications is typically approached by first using action planning to compute a sequence of discrete actions that satisfies the goal specification. In a second step, motion planning based on controllers is used to consecutively follow the discrete actions in the continuous world [8], [9], [10], [11], [12]. In many cases, however, collision-avoidance requirements and differential constraints imposed by dynamics make it difficult or impossible to design a controller that can follow a discrete action in the continuous world. As a result, decoupled approaches have had limited success.

This paper treats the problem planning collision-free and dynamically-feasible trajectories that satisfy high-level specifications as a search problem over both the discrete space of actions and the continuous space of motions. To conduct the search efficiently, this paper develops the Sampling-based Motion and Symbolic Action Planner (SMAP), which combines sampling-based motion planning with symbolic action planning. SMAP leverages from sampling-based motion planning the underlying idea of searching for a solution trajectory by selectively sampling and exploring the continuous space of motions. Sampling-based motion planners have had significant success in solving challenging reachability motion-planning problems in high-dimensional continuous spaces, as surveyed in [6], [7]. To handle both collision avoidance and differential constraints imposed by dynamics, SMAP uses a tree-based exploration of the continuous space. The tree is rooted at the initial state and is incrementally extended with trajectories obtained by applying input controls to the states in the tree and propagating the dynamics forward in time. The success and computational efficiency depends on the ability of SMAP to effectively guide the tree-based exploration toward the goal. Drawing from AI, SMAP uses symbolic action planning to guide the tree-based exploration by identifying and selecting discrete actions and regions of

the continuous space that sampling-based motion planning can further explore to significantly advance the search for a solution trajectory. The planning layers in SMAP interact with each-other through estimates on the utility of discrete states and actions, which are computed based on information gathered during the tree-based exploration. Thus, symbolic action planning guides sampling-based motion planning, while the latter feeds back information in the form of utility estimates to improve the guide in the next iteration. As a result of this interplay, SMAP becomes increasingly successful in identifying regions whose further exploration can significantly advance the search while avoiding spending valuable computational time exploring regions that do not advance the search. Simulation experiments with dynamical models of vehicles carrying out tasks given by high-level STRIPS specifications provide promising initial validation, showing that SMAP efficiently solves challenging problems.

II. Related Work

SMAP is motivated by earlier work on manipulation planning [13], [14], [15], [16] and hybrid systems [17], [18]. The work in [14] used discrete search over the manipulation graph to guide PRM (Probabilistic RoadMap [19]) sampling-based planner in the computation of transfer and transit paths. Later work by [15], [16] led to the aSyMov planner, which extended the idea even further by combining PRM with symbolic action planning, making it possible to specify high-level goals in planning-domain definition languages.

A limitation of these approaches that rely on PRM is that they cannot take into account differential constraints imposed by dynamics. To construct a roadmap, each edge (a, b) requires connecting the state a to b via a dynamically-feasible trajectory. Exact solutions to this steering problem are available only in limited cases, while numerical solutions impose significant computational cost [20], rendering roadmap construction impractical.

In contrast, SMAP uses a tree-based exploration of the state space, which does not require any steering, but only the ability to propagate dynamics forward in time. Forward propagation is readily achieved through numerical integration, making it possible for SMAP to generate not only collision-free but also dynamically-feasible trajectories that satisfy the high-level specification. Moreover, an essential component of aSyMov is a computationally-intensive backtracking procedure that checks for collisions to ensure that a candidate action is grounded in a geometric context that has at least a collision-free path from the initial state. SMAP takes a different approach, which avoids backtracking, by validating each trajectory before adding it to the tree.

SMAP builds upon prior work [21], [22], [17], [18], which showed how to effectively combine sampling-based motion planning with discrete search to compute collision-free and dynamically-feasible trajectories that satisfy high-level specifications given by linear temporal logic. A limitation of the work in [21], [22], [17], [18], is that it relies on an explicit representation of the discrete space and the possible transitions between the discrete states. To address this limitation, SMAP integrates sampling-based motion planning with symbolic action planning, which can handle complex discrete planning problems. The integration of sampling-based motion planning with symbolic action planning also makes it possible for SMAP to handle high-level goal specifications given by planning-domain languages.

III. Preliminaries

A. Discrete Specifications by Planning-Domain Definition Languages

Drawing from research in AI, this paper uses planning-domain definition languages, such as STRIPS, to allow for sophisticated high-level planning specifications. Details can be found in standard AI books [23], [5]. For completeness, a summary follows. A discrete model is a tuple $\mathcal{M} = (\mathcal{O}, \mathcal{P}, \mathcal{Q}, q_{\text{init}}, \phi_{\text{goal}}, \mathcal{A})$, where

- $\mathcal{O}$ denotes the set of objects (or atoms).
- $\mathcal{P}$ denotes the set of predicates, which express relations among objects in $\mathcal{O}$.
- $\mathcal{Q}$ denotes the discrete state space. A discrete state is a conjunction of all positive and grounded literals that currently hold in the world. A positive literal is of the form $P(t_1, \ldots, t_m)$, where $P \in \mathcal{P}$ and each t_i is an object or an object variable. A positive literal is grounded if it does not contain any object variables.
- $q_{\text{init}} \in \mathcal{Q}$ denotes the initial discrete state of the world.
- ϕ_{goal} denotes the goal specification, which is given as a formula constructed by combining positive grounded literals with Boolean operators $\neg$, $\vee$, and $\wedge$.
- $\mathcal{A}$ denotes the set of action schemas. An action schema $A = (\text{vars}, \text{pre}, \text{post}) \in \mathcal{A}$ is defined in terms of object variables, a precondition that must hold before execution, and a postcondition that will hold after execution. A precondition is usually given as a conjunction of positive literals, while a postcondition is given as a conjunction of positive or negative literals. An action $a \in A$ is a specific instantiation of the variables in A. If the precondition is satisfied, the execution of a changes the current state according to the postcondition, i.e., adding positive literals and deleting negative literals. If the precondition is not satisfied, a has no effect.

A discrete solution consists of a finite sequence of actions $[a_i]_{i=1}^{n}$ that transforms the world from q_{init} to a discrete state that satisfies ϕ_{goal}.

B. Interpretation over Continuous Spaces

The physical world, which includes the robotic system, obstacles, and objects to be manipulated, is commonly modeled in a continuous setting. The continuous space of the world, denoted by $\mathcal{S}$, consists of a finite collection of continuous variables that can describe the world, e.g., placement of objects, joint values in a robot arm, vehicle velocity.

The continuous space $\mathcal{S}$ gives meaning to the predicates in the discrete specification. As an example, $\text{On}(\text{book}, \text{table})$ holds iff the book is actually on the table. Since a continuous state $s \in \mathcal{S}$ specifies the placement of objects, one can

determine whether or not the predicate holds at s. This interpretation of which predicates actually hold at a continuous state provides a mapping from the continuous space to the discrete space, denoted as a function $\text{map}_{\mathcal{S}\mapsto\mathcal{Q}} : \mathcal{S} \to \mathcal{Q}$.

Moreover, trajectories over $\mathcal{S}$ give meaning to the actions in the discrete specification. A trajectory over $\mathcal{S}$ is a continuous function $\zeta : [0, T] \to \mathcal{S}$, parametrized by time. As the continuous state changes according to ζ, the discrete state, obtained by the mapping $\text{map}_{\mathcal{S}\mapsto\mathcal{Q}}$, may also change. As a result, the trajectory ζ *follows* a discrete action a if (i) $\text{map}_{\mathcal{S}\mapsto\mathcal{Q}}(\zeta(0))$ satisfies a's precondition and (ii) $\text{map}_{\mathcal{S}\mapsto\mathcal{Q}}(T)$ satisfies a's postcondition.

The underlying dynamics are specified as a set of differential equations $f : \mathcal{S} \times \mathcal{U} \to \dot{\mathcal{S}}$, where $\mathcal{U}$ is a control space consisting of a finite set of input variables that can be applied to the system (e.g., a car can be controlled by setting the acceleration and the rotational velocity of the steering wheel). A dynamically-feasible trajectory $\zeta : [0, T] \to \mathcal{S}$ is obtained by computing a control function $\tilde{u} : [0, T] \to \mathcal{U}$ and propagating the dynamics forward in time through numerical integration from a given state $s \in \mathcal{S}$, i.e.,

$$\zeta(t) = s + \int_0^t f(\gamma(h), \tilde{u}(h))\, dh.$$

The dynamically-feasible trajectory $\zeta : [0, T] \to \mathcal{S}$ is considered collision free if each state along the trajectory avoids collisions with the obstacles.

IV. Methods

To effectively compute a collision-free and dynamically-feasible trajectory that starts at $s_{\text{init}} \in \mathcal{S}$ and satisfies the discrete goal specification ϕ_{goal}, SMAP conducts the search both in the continuous space $\mathcal{S}$ and in the discrete state and action spaces, $\mathcal{Q}$ and $\mathcal{A}$.

In the continuous space $\mathcal{S}$, SMAP maintains the search data structure as a tree $\mathcal{T} = (V, E)$. Each vertex $v \in \mathcal{T}[V]$ is associated with a continuous state $s \in \mathcal{S}$, written as $v.s$. An edge $(v', v'') \in \mathcal{T}[E]$ indicates that SMAP has computed a collision-free and dynamically-feasible trajectory from $v'.s$ to $v''.s$. Initially, $\mathcal{T}[V]$ contains only one vertex, v_{init}, which is associated with the initial state $s_{\text{init}} \in \mathcal{S}$, and $\mathcal{T}[E]$ is empty. As SMAP explores $\mathcal{S}$, new vertices and new edges are added to $\mathcal{T}$. The procedure consists of selecting a vertex $v \in \mathcal{T}$ for expansion and then extending the tree from v by generating a collision-free and dynamically-feasible trajectory that starts at $v.s$. A common strategy is to apply some control $u \in \mathcal{U}$ to $v.s$ and simulate the dynamics forward in time until a collision occurs, a state-constraint is violated, or a maximum number of steps is exceeded [6], [7]. The control $u \in \mathcal{U}$ is generally selected uniformly at random to allow subsequent calls to extend the tree along new directions. Intermediate states along the trajectory are also added to the tree, as suggested in [6], [7]. The search terminates successfully when a vertex v_{new} is added to $\mathcal{T}$ such that $\text{map}_{\mathcal{S}\mapsto\mathcal{Q}}(v_{\text{new}}.s)$ satisfies ϕ_{goal}. The solution trajectory is then obtained by concatenating the collision-free and dynamically-feasible trajectories associated with the edges in $\mathcal{T}[E]$ connecting v_{init} to v_{new}.

Due to challenges posed by the high dimensionality of the continuous space $\mathcal{S}$, collision-avoidance requirements, differential constraints imposed by dynamics, and the complexity of the discrete specification, the success of the search depends on the ability of SMAP to effectively and selectively sample and explore $\mathcal{S}$. SMAP employs symbolic action planning to identify regions in $\mathcal{S}$ that sampling-based motion planning can then selectively sample and explore to significantly advance the search for a collision-free and dynamically-feasible trajectory that satisfies ϕ_{goal}.

In particular, SMAP groups the tree vertices according to the mapping function $\text{map}_{\mathcal{S}\mapsto\mathcal{Q}}$. Let Γ denote the list of all such groups, where $\Gamma_q \in \Gamma$ contains all the vertices $v \in \mathcal{T}$ such that $\text{map}_{\mathcal{S}\mapsto\mathcal{Q}}(v.s) = q$, i.e., each time a vertex v is added to $\mathcal{T}$, v is also added to $\Gamma_{\text{map}_{\mathcal{S}\mapsto\mathcal{Q}}(v.s)}$.

Consider one such group $\Gamma_q \in \Gamma$ and let a be an action whose precondition is satisfied by q. Let $a(q)$ denote the discrete state obtained as a result of a's effect on q. Sampling-based motion planning in SMAP, denoted by ExploreAction$(\mathcal{T}, \Gamma, \Gamma_q, a)$, can then advance the search by extending $\mathcal{T}$ from vertices associated with Γ_q (which satisfy a's precondition) toward the region of the continuous space $\mathcal{S}$ that satisfies a's postcondition, i.e., $\text{map}^{-1}_{\mathcal{S}\mapsto\mathcal{Q}}(a(q)) = \{s : s \in \mathcal{S} \wedge \text{map}_{\mathcal{S}\mapsto\mathcal{Q}}(s) = a(q)\}$. There is, however, an associated computational cost with each invocation of ExploreAction$(\mathcal{T}, \Gamma, \Gamma_q, a)$. This raises a central issue about which group $\Gamma_q \in \Gamma$ and which action a, among the many available options, should be selected for exploration at each invocation of ExploreAction. To address this issue, SMAP maintains a running weight estimate Util(Γ_q, a) on the utility of having ExploreAction$(\mathcal{T}, \Gamma_q, a)$ spend additional time attempting to extend $\mathcal{T}$ from vertices associated with Γ_q toward the region $\text{map}^{-1}_{\mathcal{S}\mapsto\mathcal{Q}}(a(q))$. The objective of Util$(\Gamma_q, a)$, which is updated based on new information gathered by sampling-based motion planning, is three-fold:

(i) give high utility to (Γ_q, a) when the action plan from the discrete state $a(q)$ to a discrete state that satisfies ϕ_{goal} is short. This is to bias the search in the continuous space $\mathcal{S}$ so that the sampling-based motion planner follows action plans that can quickly lead to a solution.

(ii) give high utility to (Γ_q, a) when it is under-explored, since additional exploration by the sampling-based motion planner could advance the search further.

(iii) give low utility to (Γ_q, a) when it is over-explored, since over-exploration does not bring much new information and wastes valuable computational time.

The core of SMAP interleaves symbolic action planning, sampling-based motion planning, and updates to utility estimates in order to effectively compute a collision-free and dynamically-feasible trajectory that satisfies ϕ_{goal}:

- Use symbolic action planning and the utility estimates to select a group $\Gamma_q \in \Gamma$ and an action a whose precondition is satisfied by q. Bias the selection process

toward pairs (Γ_q, a) associated with high utilities.
- Use EXPLOREACTION$(\mathcal{T}, \Gamma_q, a)$ for a short period of time to extend $\mathcal{T}$ from vertices associated with Γ_q toward continuous states in $\mathrm{map}^{-1}_{\mathcal{S}\mapsto\mathcal{Q}}(q)$, which satisfy a's postcondition.
- Update the estimates UTIL(Γ_q, a) based on new information gathered from EXPLOREACTION$(\mathcal{T}, \Gamma_q, a)$.

This interplay of symbolic action planning and sampling-based motion planning through utility estimates is a crucial component of the computational efficiency of SMAP. In particular, it allows SMAP to make proper use of the computational time by selectively sampling and exploring those actions a and regions of $\mathcal{S}$ that allow SMAP to significantly advance the search for a collision-free and dynamically-feasible trajectory that satisfies ϕ_{goal}. Pseudocode is given in Algo. 1. Details of the main components in SMAP follow.

Algorithm 1 SMAP

Input: problem specification; $t_{\max}$: upper bound on time
Output: A collision-free and dynamically-feasible trajectory that satisfies high-level specification or null if no solution is found

♢ *initialize data structures*
$\mathcal{T} \leftarrow \emptyset$; $\Gamma \leftarrow \emptyset$
$v_{\text{root}} \leftarrow$ new vertex; $v_{\text{root}}.s \leftarrow s_{\text{init}}$; v_{root}.parent $\leftarrow$ null
ADDVERTEX$(v_{\text{root}}, \mathcal{T}, \Gamma)$
♢ *core loop: interplay between symbolic action planning and sampling-based motion planning through action utilities*
while ELAPSEDTIME $< t_{\max} \wedge$ no solution path **do**
  $\Gamma_q \leftarrow$ SELECTGROUP(Γ)
  $a \leftarrow \Gamma_q$.curr_action
  status $\leftarrow$ EXPLOREACTION$(\mathcal{T}, \Gamma, \Gamma_q, a)$ (for a short time)
  if status $=$ solved **then**
    $\zeta \leftarrow$ concatenate tree trajectories from root to last vertex
    return ζ
  UPDATEUTIL$(\mathcal{T}, \Gamma_q, a, \text{status})$
  Γ_q.curr_action $\leftarrow$ SELECTACTION(Γ_q)
  for each new $\Gamma_{q_{\text{new}}}$ added to Γ **do**
    $[a_i]_{i=1}^n \leftarrow$ SYMBOLICPLANNER$(\mathcal{A}, q_{\text{new}}, \phi_{\text{goal}})$
    $\Gamma_{q_{\text{new}}}$.actions $\leftarrow \{a_1\} \cup \Gamma_{q_{\text{new}}}$.actions
return null

A. Symbolic Action Planning

SYMBOLICPLANNER$(\mathcal{A}, q, \phi_{\text{goal}})$ computes an action plan $[a_i]_{i=1}^n$, which transforms the discrete state q to a discrete state that satisfies ϕ_{goal}. This is the only requirement imposed on SYMBOLICPLANNER, since SMAP uses it as a black-box. Therefore, SMAP can take advantage of research in AI and plug in efficient symbolic actions planners [24], [25], [26], [27], [28], [5]. These action planners, which reason about the discrete problem symbolically, are capable of effectively handling complex specifications in definition-domain planning languages, such as STRIPS.

B. Action Selection

Each $\Gamma_q \in \Gamma$ maintains a list of actions, Γ_q.actions, that are available for the selection process. This list is generally a small subset of all the actions whose preconditions are satisfied by q. For each $a \in \Gamma_q$.actions, Γ_q maintains a utility estimate, UTIL(Γ_q, a). An action $a \in \Gamma_q$.actions is then selected with probability proportional to its utility, i.e.,

$$\text{ProbSelect}_{\Gamma_q}(a) = \frac{\text{UTIL}(\Gamma_q a)}{\sum_{a' \in \Gamma_q.\text{actions}} \text{UTIL}(\Gamma_{q'}, a')}.$$

The selected action is kept as Γ_q.curr_action.

In this way, actions associated with high utilities are selected more often. This allows the sampling-based motion planner to quickly expand the search toward promising directions. At the same time, each available action has a non-zero probability of being selected, which is important to ensure that the search also expands along new directions.

When the group Γ_q is first created, the function SYMBOLICPLANNER$(\mathcal{A}, q, \phi_{\text{goal}})$ is invoked to compute an action plan $[a_i]_{i=1}^n$, which transforms the discrete state q to a discrete state that satisfies ϕ_{goal}. Since the overall objective is to satisfy ϕ_{goal}, if there are no action plans from q that satisfy ϕ_{goal}, then Γ_q is deleted from Γ. Otherwise, the first action of the plan, a_1, is added to Γ_q.actions. Thus, initially, Γ_q.actions contains only one action. Note that this provides an opportunity to use symbolic action planners that can efficiently compute the first action of an action plan.

As the search progresses and new information is gathered by the sampling-based motion planner, the utilities of actions in Γ_q.actions are updated to take into account this new information. When the utilities of all the actions in Γ_q.actions fall below a certain threshold, SYMBOLICPLANNER$(\mathcal{A}, q, \phi_{\text{goal}})$ is invoked again to compute a new action plan $[a_i]_{i=1}^n$, where $a_1 \notin \Gamma_q$.actions. If it succeeds, as during initialization, the first action of the plan is added to Γ_q.actions and is also made available for selection.

In this way, the search is made broader when it becomes difficult to make significant progress using the current actions to guide the sampling-based motion planner. This allows the sampling-based motion planner to explore new directions, which could lead to further progress in the search for a collision-free a dynamically-feasible trajectory that satisfies the goal specification.

C. Group Selection

The utility of a group $\Gamma_q \in \Gamma$ is defined as the utility of the action currently selected in Γ_q, i.e.,

$$\text{UTIL}(\Gamma_q) = \text{UTIL}(\Gamma_q, \Gamma_q.\text{curr\_action}).$$

Then, as during action selection, a group Γ_q is selected from Γ with probability proportional to its utility, i.e.,

$$\text{ProbSelect}_{\Gamma}(\Gamma_q) = \frac{\text{UTIL}(\Gamma_q)}{\sum_{\Gamma_{q'} \in \Gamma} \text{UTIL}(\Gamma_{q'})},$$

which aims to strike a balance between being greedy and being methodical by giving preference to groups associated with high utilities, without ignoring other groups in Γ.

D. Action Utility

Drawing from earlier work in sampling-based motion planning [29], [30], [31], [21], the utility estimates in this

paper are designed to be computationally efficient and work well in practice. Specifically, $\textsc{Util}(\Gamma_q, a)$ is computed as

$$\textsc{Util}(\Gamma_q, a) = \frac{1}{(1+\text{nplan}^2)(1+\text{nsel})(1+|\Gamma_{a(q)}|)},$$

where nplan denotes the length of the action plan $[a_i]_{i=1}^n$ (with $a = a_1$) computed by SYMBOLICPLANNER when Γ_q is first created (Algo. 1:14) and nsel denotes the number of times EXPLOREACTION has been invoked with Γ_q, a as arguments. Since the overall objective is to compute a solution as quickly as possible, this scheme assigns high utility to (Γ_q, a) when the action plan is short. To ensure that SMAP does not spend all the time exploring a particular (Γ_q, a), the utility is reduced after each invocation of $\textsc{ExploreAction}(\mathcal{T}, \Gamma, \Gamma_q, a)$. To avoid over-exploration, the utility is also reduced as more and more tree vertices are added to $\Gamma_{a(q)}$. Further improving the proposed utility estimates remains an important direction for future research.

E. Explore Action

The objective of $\textsc{ExploreAction}(\mathcal{T}, \Gamma, \Gamma_q, a)$ is to extend $\mathcal{T}$ toward the region $\text{map}^{-1}_{\mathcal{S}\mapsto\mathcal{Q}}(a(q))$, so that $\mathcal{T}$ can follow in the continuous space $\mathcal{S}$ the discrete action a. This in itself a motion-planning problem, so SMAP can draw from successful sampling-based approaches. Note that $\textsc{ExploreAction}(\mathcal{T}, \Gamma, \Gamma_q, a)$ is invoked only for a short time. In this way, if the exploration of action a does not advance the search, then SMAP will seek to explore alternative actions in future iterations (see Algo 1).

EXPLOREACTION proceeds in an iterative fashion. At each iteration, EXPLOREACTION first selects a vertex v from which to extend $\mathcal{T}$. At a second step, EXPLOREACTION generates a dynamically-feasible trajectory $\zeta : [0, T] \to \mathcal{S}$ that starts at $v.s$, i.e., $\zeta(0) = v.s$. ζ is generated by sampling a control $u \in \mathcal{U}$ uniformly at random and simulating the dynamics forward in time starting from $v.s$ until a collision occurs, a state-constraint is violated, or a maximum number of steps is exceeded [6], [7]. Intermediate collision-free states along ζ are added as new vertices to $\mathcal{T}$. If a new vertex, v_{new}, satisfies the goal specification, i.e., $\text{map}_{\mathcal{S}\mapsto\mathcal{Q}}(v_{\text{new}})$ satisfies ϕ_{goal}, then a solution trajectory is obtained by concatenating the tree trajectories from the root of $\mathcal{T}$ to v_{new}. Otherwise, the above steps are repeated several times. The remainder of the section describes in more detail the vertex-selection and trajectory-generation strategies.

1) Vertex Selection: Over the years, numerous vertex-selection strategies have been proposed in motion-planning literature that rely on distance metrics, nearest neighbors, probability distributions, and many others, as surveyed in [6], [7]. Drawing from this body of research and earlier work [21], [22], the vertex-selection strategy in this paper combines the advantages of several successful techniques.

To bias the growth of $\mathcal{T}$ toward $\text{map}^{-1}_{\mathcal{S}\mapsto\mathcal{Q}}(a(q))$, one approach, proposed in [32], that has been shown to work well in practice is to first sample a continuous state, s, such that $\text{map}_{\mathcal{S}\mapsto\mathcal{Q}}(s) = a(q)$. The vertex from which to extend $\mathcal{T}$ is then selected from the vertices in $\Gamma_q \cup \Gamma_{a(q)}$ as the vertex whose associated continuous state, $v.s$, is the closest to s according to a distance metric. The effect of this strategy is to pull $\mathcal{T}$ toward $\text{map}^{-1}_{\mathcal{S}\mapsto\mathcal{Q}}(a(q))$. Approximate nearest neighbors [33], [34] are used to speed up computation without any significant loss in accuracy.

Another objective of EXPLOREACTION is to grow $\mathcal{T}$ toward unexplored or sparsely explored areas. This avoids over-exploration and leads the search toward new directions. As proposed in [35], [30], [31], [21], [22], an effective strategy for these purposes is to select a vertex v in $\Gamma_q \cup \Gamma_{a(q)}$ with probability inversely proportional to the density around a ball centered at $v.s$. The effect of this strategy is to push $\mathcal{T}$ toward unexplored or sparsely explored areas.

A third objective of EXPLOREACTION is to further extend the search forward by increasing the depth of $\mathcal{T}$, similar to depth-first search in a discrete setting. The vertex from which to extend $\mathcal{T}$ is then selected uniformly at random from the last vertices (around 20 is shown to work well in practice) added to $\mathcal{T}$ as a result of previous invocations of $\textsc{ExploreAction}(\mathcal{T}, \Gamma, \Gamma_q, a)$.

The overall vertex-selection strategy then simply selects at each iteration one of the above strategies uniformly at random. The effect is a vertex-selection strategy that pulls $\mathcal{T}$ toward $\text{map}^{-1}_{\mathcal{S}\mapsto\mathcal{Q}}(a(q))$, while avoiding over-exploration, finding new directions, increasing sampling in sparsely explored areas, and expanding the search depth.

2) Trajectory Generation: The trajectory-generation strategy discussed in this section is designed to work well in practice for a wide class of systems and actions. It is possible, however, to further improve these strategies by taking advantage of the problem specification. In particular, one can design specific trajectory-generation strategies for each action a. As an example, if the action is "GraspObject," then the trajectory-generation strategy can be designed to produce open-and-closing motions of the end-effector tool. In this way, action-specific strategies can be used to further improve the overall effectiveness of $\textsc{ExploreAction}(\mathcal{T}, \Gamma, \Gamma_q, a)$.

V. EXPERIMENTS AND RESULTS

The proposed method is tested on a variation of the 9-puzzle game. An illustration is provided in Fig. 1. The robot,

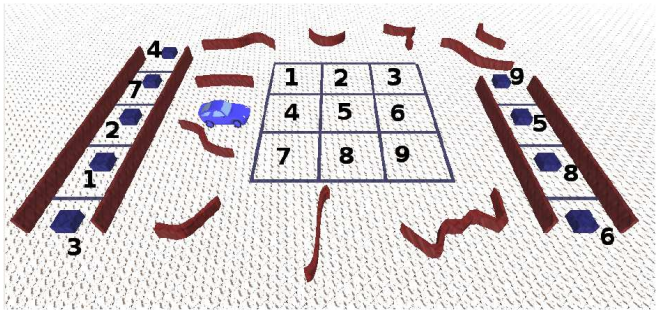

Fig. 1. The car (with second-order dynamics) needs to pick up the objects (shown in blue, labeled $1, \ldots 9$) one at a time and place them in the puzzle cells corresponding to their labels, i.e., object i should be placed in area i. Obstacles are shown in red. Figure better viewed in color.

which is a car, needs to pick up the cube objects (labeled

$1, \ldots, 9$) and place each object at the appropriate puzzle cell, so that the label of the object matches with the label of the cell. The car picks up an object by touching it, but it cannot pick up more than one object at a time. The object that is picked up can be released at any empty location (indicated by the blue lines), including empty grid cells. After releasing an object the car can pickup the same object or a different objects. The car should avoid collisions with the objects at all times (except the object that it picks up, which is allowed to be in contact with the car). No collisions should occur between two objects or with an obstacle.

The car is modeled as a second-order dynamical system. Details can be found in [7, pp. 744]. The state $s = (x, y, \theta, v, \psi)$ consists of the position $(x, y) \in \mathbb{R}^2$ ($|x|, |y| \leq 3.75m$), orientation $\theta \in [-\pi, \pi)$, velocity v ($|v| \leq 3m/s$), and steering-wheel angle ψ ($|\psi| \leq 50^\circ$). The car is controlled by setting the acceleration u_0 ($|u_0| \leq 1m/s^2$) and the rotational velocity of the steering-wheel angle u_1 ($|u_1| \leq 100^\circ/s$). The equations of motions are $\dot{x} = v\cos(\theta)$; $\dot{y} = v\sin(\theta)$; $\dot{\theta} = v\tan(\psi)/L$; $\dot{v} = u_0$; $\dot{\psi} = u_1$, where $L = 0.5m$ is the distance between the front and rear axles. The body length and width are set to L and $0.5L$, respectively. The scaling factor is $1m = 0.14$ workspace units.

The high-level discrete actions consists of picking up an object and moving them from one puzzle cell to the other, i.e., "PickUp", "Release", "PuzzleMoveLeft", "PuzzleMoveRight", "PuzzleMoveUp", "PuzzleMoveDown". The problem specification is encoded in STRIPS. An illustration of expressing one such action in STRIPS is provided below:

```
(:action PuzzleMoveUp
   :parameters (?t ?x ?py ?ny)
   :precondition (and (tile ?t) (position ?x)
         (position ?py) (position ?ny)
         (dec ?ny ?py) (at blank ?x ?ny)
         (at ?t ?x ?py))
   :effect (and (not (at blank ?x ?ny))
         (not (at ?t ?x ?py)
         (at blank ?x ?py) (at ?t ?x ?ny)))
```

This task provides several challenges. The discrete space is complex. There is a large number of action plans, which provide solutions in the discrete setting. Due to the arrangement of objects and obstacles, the objects cannot be transferred just in any order. Moreover, the car will need to move the objects around in the puzzle, from one cell to the other, in order to achieve the desired configuration.

The experiments provide promising initial validation of SMAP. These experiments highlight the importance of the interplay between symbolic action planning and sampling-based motion planning. Without symbolic action planning, it is impossible for sampling-based motion planning just by itself to find a solution, since the solution trajectory must satisfy complex high-level specifications. We conducted numerous trials (20 per planner). In each case a stand-alone sampling-based path planners failed to find a solution. On the other hand, SMAP was able to efficiently find a solution in a matter of a few minutes (average 6 mins).

VI. Discussion

This paper proposed a multi-layered approach, SMAP, which incorporates symbolic, geometric, and differential constraints into sampling-based motion planning. Given a high-level goal specification in a planning-domain definition language, such as STRIPS, SMAP computes a collision-free and dynamically-feasible trajectory that satisfies the goal specification. A crucial component of SMAP is the interplay between symbolic action planning and sampling-based motion planning. SMAP leverages from state-of-the-art sampling-based motion planning the underlying idea of searching for a solution by selectively sampling and exploring the continuous space. To effectively incorporate collision-avoidance and differential constraints imposed by underlying dynamics, SMAP conducts a tree-based exploration of the continuous state space. Drawing from research in AI, SMAP uses symbolic action planning to identify discrete actions and regions of the continuous space that sampling-based motion planning can selectively sample and explore to significantly advance the search. An objective for future work is to further improve the interplay in SMAP between symbolic action planning and sampling-based motion planning. We are also working on adapting the proposed approach to real robotic platforms. This will allow us to tackle increasingly complex problems arising in robot manipulation and automation.

Acknowledgment

This work is supported by NSF IIS-0748338.

References

[1] R. Fikes and N. Nilsson, "STRIPS: a new approach to the application of theorem proving to problem solving," *Artificial Intelligence*, vol. 2, no. 3-4, pp. 189–208, 1971.

[2] E. P. D. Pednault, "ADL and the state-transition model of action," *J Logic Computation*, vol. 4, no. 5, pp. 467–512, 1994.

[3] M. Ghallab, A. Howe, C. Knoblock, C. McDermott, A. Ram, D. Veloso, M. Weld, and D. Wilkins, "PDDL–the planning domain definition language," 1998.

[4] B. Marthi, S. Russell, and J. Wolfe, "Angelic semantics for high-level actions," in *Intl Conf on Automated Planning and Scheduling*, Providence, RI, 2007.

[5] M. Ghallab, D. S. Nau, and P. Traverso, *Automated Planning: theory and practice*. Morgan Kaufmann, 2004.

[6] H. Choset, K. M. Lynch, S. Hutchinson, G. Kantor, W. Burgard, L. E. Kavraki, and S. Thrun, *Principles of Robot Motion: Theory, Algorithms, and Implementations*. MIT Press, 2005.

[7] S. M. LaValle, *Planning Algorithms*. Cambridge, MA: Cambridge University Press, 2006.

[8] R. C. Arkin, "Integrating behavioral, perceptual and world knowledge in reactive navigation," *Robotics and Autonomous Systems*, vol. 6, pp. 105–122, 1990.

[9] D. W. Payton, J. K. Rosenblatt, and D. M. Keirsey, "Plan guided reaction," *IEEE Trans. on Systems, Man, and Cybernetics*, vol. 20, no. 6, pp. 1370–1372, 1990.

[10] A. Saffiotti, K. Konolige, and E. H. Ruspini, "A multivalued logic approach to integrating planning and control," *Artificial Intelligence*, vol. 76, no. 1-2, pp. 481–526, 1995.

[11] C. Belta, A. Bicchi, M. Egerstedt, E. Frazzoli, E. Klavins, and G. J. Pappas, "Symbolic planning and control of robot motion," *IEEE Robotics and Automation Magazine*, vol. 14, no. 1, pp. 61–71, 2007.

[12] G. E. Fainekos, A. Girard, H. Kress-Gazit, and G. J. Pappas, "Temporal logic motion planning for dynamic mobile robots," *Automatica*, vol. 45, no. 2, pp. 343–352, 2007.

[13] R. Alami, J. Laumond, and T. Siméon, "Two manipulation planning algorithms," in *Int. Work. Algo. Found. Robot.*, Stanford, CA, 1995, pp. 109–125.

[14] C. Nielsen and L. E. Kavraki, "A two level fuzzy PRM for manipulation planning," in *IEEE/RSJ Int. Conf. Intell. Robot. Syst.*, Takamatsu, Japan, 2000, pp. 1716–1722.

[15] F. Gravot, S. Cambon, and R. Alami, "aSyMov: a planner that deals with intricate symbolic and geometric problems," in *Int. Symp. Robotics Research*, ser. Springer Tracts in Advanced Robotics, Siena, Italy, 2003, vol. 15, pp. 100–110.

[16] S. Cambon, R. Alami, and F. Gravot, "A hybrid approach to intricate motion, manipulation and task planning," *Int. J. Robot. Res.*, no. 1, pp. 104–126, 2009.

[17] E. Plaku, L. E. Kavraki, and M. Y. Vardi, "Hybrid systems: From verification to falsification by combining motion planning and discrete search," *Formal Methods in System Design*, vol. 34, no. 2, pp. 157–182, 2008.

[18] ——, "Falsification of LTL safety properties in hybrid systems," in *Intl. Conf. on Tools and Algorithms for the Construction and Analysis of Systems*, ser. Lecture Notes in Comp. Sci., vol. 5505, York, UK, 2009, pp. 368–382.

[19] L. E. Kavraki, P. Švestka, J.-C. Latombe, and M. H. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *IEEE Trans. Robot. Autom.*, vol. 12, no. 4, pp. 566–580, 1996.

[20] H. Keller, *Numerical Methods for Two-Point Boundary-Value Problems*. New York, NY: Dover, 1992.

[21] E. Plaku, L. E. Kavraki, and M. Y. Vardi, "Discrete search leading continuous exploration for kinodynamic motion planning," in *Robotics: Sci. and Systems*, Atlanta, GA, 2007, pp. 326–333.

[22] ——, "Impact of workspace decompositions on discrete search leading continuous exploration (DSLX) motion planning," in *IEEE Int. Conf. Robot. Autom.*, Pasadena, CA, 2008, pp. 3751–3756.

[23] S. Russell and P. Norvig, *Artifical Intelligence: A Modern Approach*, 2nd ed. Prentice Hall, 2002.

[24] A. L. Blum and M. L. Furst, "Fast planning through planning graph analysis," *Artificial Intelligence*, vol. 90, no. 1-2, pp. 281–300, 1997.

[25] B. Bonet and H. Geffner, "Planning as heuristic search," *Artificial Intelligence*, vol. 129, no. 1-2, pp. 5–33, 2001.

[26] E. Keyder and H. Geffner, "Heuristics for planning with action costs revisited," in *European Conf. on Artificial Intelligence*, Patras, Greece, 2008, pp. 588–592.

[27] J. Hoffmann and B. Nebel, "The FF planning system: Fast plan generation through heuristic search," *J. of Artificial Intelligence Research*, vol. 14, pp. 253–302, 2001.

[28] J. Hoffmann, "The Metric-FF planning system: Translating "ignoring delete lists" to numeric state variables," *J. of Artificial Intelligence Research*, vol. 20, no. 20, pp. 291–341, 2003.

[29] B. Burns and O. Brock, "Single-query motion planning with utility-guided random trees," in *IEEE Int. Conf. Robot. Autom.*, Rome, Italy, 2007, pp. 3307–3312.

[30] G. Sánchez and J.-C. Latombe, "On delaying collision checking in PRM planning: Application to multi-robot coordination," *Int. J. Robot. Res.*, vol. 21, no. 1, pp. 5–26, 2002.

[31] A. M. Ladd and L. E. Kavraki, "Motion planning in the presence of drift, underactuation and discrete system changes," in *Robotics: Sci. and Systems*, Boston, MA, 2005, pp. 233–241.

[32] S. M. LaValle and J. J. Kuffner, "Randomized kinodynamic planning," *Int. J. Robot. Res.*, vol. 20, no. 5, pp. 378–400, 2001.

[33] E. Plaku and L. E. Kavraki, "Quantitative analysis of nearest-neighbors search in high-dimensional sampling-based motion planning," in *Int. Work. Algo. Found. Robot.*, ser. Springer Tracts in Advanced Robotics, New York, NY, 2006.

[34] ——, "Nonlinear dimensionality reduction using approximate nearest neighbors," in *SIAM Int.Conf. on Data Mining*, Minneapolis, Minnesota, 2007, pp. 3711–3716.

[35] D. Hsu, R. Kindel, J.-C. Latombe, and S. Rock, "Randomized kinodynamic motion planning with moving obstacles," *Int. J. Robot. Res.*, vol. 21, no. 3, pp. 233–255, 2002.