Learning Feasibility for Task and Motion Planning in Tabletop Environments

Andrew M. Wells[1], Neil T. Dantam[2], Anshumali Shrivastava[1] and Lydia E. Kavraki[1]

***Abstract*—Task and motion planning (TMP) combines discrete search and continuous motion planning. Earlier work has shown that to efficiently find a task-motion plan, the discrete search can leverage information about the continuous geometry. However, incorporating continuous elements into discrete planners presents challenges. We improve the scalability of TMP algorithms in tabletop scenarios with a fixed robot by introducing geometric knowledge into a constraint-based task planner in a robust way. The key idea is to learn a classifier for feasible motions and to use this classifier as a heuristic to order the search for a task-motion plan. The learned heuristic guides the search towards feasible motions and thus reduces the total number of motion planning attempts. A critical property of our approach is allowing robust planning in diverse scenes. We train the classifier on minimal exemplar scenes and then use principled approximations to apply the classifier to complex scenarios in a way that minimizes the effect of errors. By combining learning with planning, our heuristic yields order-of-magnitude run time improvements in diverse tabletop scenarios. Even when classification errors are present, properly biasing our heuristic ensures we will have little computational penalty.**

***Index Terms*—Motion and Path Planning; Task Planning**

I. Introduction

INTEGRATED task and motion planning (TMP) requires planning over a task-motion space that combines discrete actions for manipulating objects with continuous, collision-free motions that achieve these actions in a high degree-of-freedom (DOF) composite space representing the state of the robot and the objects. A strictly-hierarchical approach, where a task planner first chooses a sequence of actions and then a motion planner finds valid trajectories for each action, is incomplete because there is no guarantee that actions selected by the task planner will be geometrically feasible. Previous works [1], [2], [3] break the strict hierarchy between task planning and motion planning by communicating motion feasibility information from the motion planner to the task planner.

Obtaining completeness in TMP presents challenges due to the computational difficulty of high-dimensional motion planning. Most practical motion-planning algorithms for high DOF systems such as manipulators are not complete but at best probabilistically complete. A probabilistically complete motion planner cannot prove that a motion is infeasible; the planner can only run until a timeout. A TMP framework for a high DOF system must call a probabilistically complete motion planner using a timeout; otherwise, the motion planner could run forever. On infeasible actions, the planner is guaranteed to timeout, but there is also a possibility that the planner will time out on a feasible action. So when the planner times out, a probabilistically complete TMP framework will typically defer the action, instead attempting another action. It will then later revisit the action to reattempt motion planning with a longer timeout. The need to expend computation on many possibly infeasible motions imposes a high computational cost.

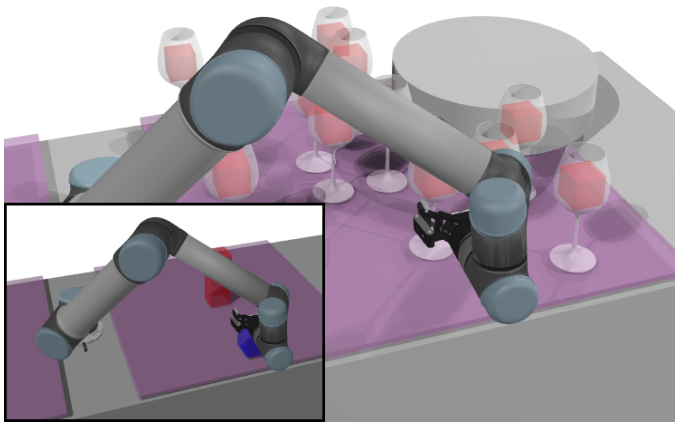


Fig. 1: Our learned heuristic expands from training on two prisms (front left) to runtime scenes with arbitrary numbers and types of objects. We approximate arbitrary meshes with inscribed prisms (the red blocks in the wine glasses). Errors induced by the approximation are corrected during planning, producing a robust system.

We show empirically that the run time of TMP for tabletop manipulation domains is dominated by infeasible motion planning attempts (see section VI). As described previously, reducing the timeout of the motion planner is not a viable approach, because then the motion planner will frequently fail to solve feasible problems where a plan exists. Typical off-the-shelf task planners used in TMP frameworks [1], [3] focus on efficient planning in discrete spaces and admit only limited geometric information. For example, in a scenario where the task planner is searching for a discrete plan containing six steps, it will enumerate satisfying plans in an arbitrary order based on the solver's heuristics and then call a motion planner for each discrete step until all steps in a plan succeed.

Because calling motion planning on infeasible actions increases overall run time, we propose a principled method

Manuscript received: August, 18, 2018; Revised November, 19, 2018; Accepted December, 15, 2018.

This paper was recommended for publication by Editor Nancy Amato upon evaluation of the Associate Editor and Reviewers' comments. This work was supported by a NASA Space Technology Research Fellowship.)

[1]Andrew Wells, Anshumali Shrivastava and Lydia Kavraki are with the School of Engineering, Department of Computer Science, Rice University, Houston TX andrew DOT wells AT rice DOT edu, lastname AT rice DOT edu

[2]Neil Dantam is with the School of Engineering, Department of Computer Science, Colorado School of Mines, Golden CO dantam AT mines DOT edu

Digital Object Identifier (DOI): see top of this page.

of incorporating a motion feasibility heuristic into a TMP framework. TMP frameworks break the overall plan into smaller steps, which in turn are repetitive, suggesting this problem may be amenable to machine learning algorithms, in our case an SVM using geometric features of the objects being manipulated Figure 2. This approach balances the difficulty of proving motion infeasibility with the impracticality of learning a perfect classifier. Rather than attempting to generalize the classifier, we expand the domain by using the classifier on approximations that are intentionally biased to reduce the cost of errors on overall run time.

This paper improves the scalability of task and motion planning by demonstrating a new, principled method to incorporate a heuristic for motion feasibility and to expand the domain of the heuristic to new inputs. We believe that the underlying method can be applied to many different TMP approaches because expending computation to search for infeasible motion plans is a fundamental challenge in probabilistically complete TMP. We demonstrate that a classifier for feasibility can be trained, that it improves scalability, and that we can incorporate it in a way that is robust to classification error.

II. Related Work

Task Planning: Research in task planning for robotics traces back to early efforts in STRIPS [4]. Task planning approaches typically focus on efficiently searching the state space, either via heuristic search [5], [6], [7] or constraint-based methods [8], [9]. Heuristic search methods, such as FF [6], use general-purpose heuristics to reduce the number of states that are expanded. Constraint-based methods, such as Madagascar [9], propagate constraints to avoid searching the entire state space.

Motion Planning: For high DOF manipulators, the major motion planning approaches are optimization-based and sampling-based planners. Optimization-based planners [10], [11] can quickly solve some problems, but other problems—especially cases involving small obstacles or narrow passages—pose fundamental challenges for these planners. While optimization-based motion planners are used in some TMP frameworks, they are incomplete. Thus, TMP algorithms using optimization-based motion planning will be incomplete. In contrast to optimization-based planners, typical sampling-based planners are probabilistically complete, i.e. when a solution exists, the sampling-based planner will eventually find it [12], [13]. However, probabilistic completeness does not guarantee that a solution will be found within a given time limit, so decision problem versions of these algorithms are semi-deciding [14]. In our work, we use RRT-Connect [13] to ensure probabilistic completeness, but other standard sampling-based planners could be used instead with only minor modifications [15]. Heuristic search can also be used for high DOF motion planning [16].

Task and Motion Planning: Due to the inherent difficulty of the TMP sub-problems—task planning is NP-hard and motion planning is PSPACE hard—many TMP approaches consider restricted domains or set aside completeness to improve efficiency for specific problem classes [14]. [2] can guarantee completeness under some assumptions, such as reversible actions, but performs greedy execution of plans, which consumes time and energy if actions must ultimately be reversed. [17] uses geometric constraints to reason about motion planning in some scenarios, and [18] uses geometric information to guide the task-level search. [3] provides a framework to interface between task planners and motion planners, but does not achieve probabilistic completeness until the later work mentioned below. [19] focuses on precomputing motion roadmaps to guide the search.

Approaches that are probabilistically complete include [20], which achieves asymptotic optimality in planar manipulation domains. The aSyMov planner [21], [22] uses an FF based task planner with lazily-expanded roadmaps; roadmaps allow plan re-use but present difficulty when configuration spaces change as objects interact—e.g., when stacking and pushing. The Synergistic Framework and related methods [23], [24], [25] use a discrete search weighted by attempted motion planning to provide feedback between the discrete and continuous layers. In contrast, our work leverages both a constraint solver and learned heuristic to more efficiently search the task space.

Other methods employ discrete abstractions of geometric domains to synthesize policies [26] or handle dynamics [25], but these methods do not focus on the high-DOF and changing configuration spaces that occur during manipulation.

Our approach extends the constraint-based TMP implementation of [1], [27] by incorporating a learned feasibility heuristic to guide search even in novel scenes.

Heuristics for Task and Motion Planning: In an attempt to learn a general heuristic for task planning, Yoon and Xu [28], [29] focus on learning a correcting factor to the relaxed-deletion based heuristic of FF-search [6].

Learned heuristics have shown promise in the realm of motion planning. For instance, sampling-based planners can learn to bias the sampling distribution to achieve better planning times [30]. Dynamic Movement Primitives attempt to learn approaches to manipulation to allow cooperation between multiple robots [31]. Our approach differs by using learning to prune the TMP search but not to guide individual instances of motion planning.

Garrett et al. [32] introduce a heuristic that incorporates geometric information into FF search. The heuristic is not based on learning and does not offer our guarantees on domain expansion. Additionally it is nontrivial to incorporate into frameworks that use a constraint-based task planner.

Several recent works apply learning to TMP. Chitnis et al. [33] give a probabilistically complete method which uses reinforcement learning to guide both the task level search for a plan as well as the "refinement" from a symbolic description into a concrete plan. In [34] the notion of "score-space" is introduced as a metric to measure similarity between problem instances to improve motion planning time. In both approaches, learning is used for a different purpose than we propose here and a principled approach to handling classification error is not given.

Experience Based Planning: Though we do not use experience based motion planning, it is an important point of discussion for our work since we too use pre-computation to improve runtime performance. Deep learning approaches such as [35] are promising but require large amounts of training data to learn a sampling distribution. We focus on an easier problem of predicting motion feasibility and require less training data.

[36] focuses on reusing planning for common areas of motion, but it is not clear how it would be adapted to deal with unseen objects. The Thunder framework [37] runs planning from scratch in parallel with a retrieve and repair planner to maintain completeness. In general, we view such methods as complementary to our own as they solve a different, but related problem. It should be noted, however, that building any form of experience graph becomes significantly more complicated as the number—and combinations—of manipulable objects increases. The free configuration space depends on the objects in question, and thus can grow with the number of combinations of objects (e.g., stacking or nested objects). The aSyMov planner uses such a method, but the authors note that scalability quickly becomes an issue [22]. One option is to prune the graph by keeping the "important" nodes and discarding the rest, but doing this for experience graphs on a variety of objects is an open problem. Our approach utilizes an SVM which takes milliseconds per call and has a small memory footprint.The novel contribution of our work is a principled method of incorporating feasiblity heuristics into a TMP framework to allow domain expansion.

III. Problem Description

We focus on Task and Motion Planning involving manipulation in deterministic, fully observable cases. Additionally, we assume the objects in our scene are rigid and that our C-space is connected. Informally, we consider a class of problems where a discrete task planning problem is specified (in e.g., PDDL) and some subset of the task actions (operators) have a continuous motion component. To specify this component, we include definitions of *domain semantics* that map the discrete operators to continuous motions. For example, the operator `PICK-UP ?obj` has discrete preconditions corresponding to the discrete facts that the gripper is empty and the object has nothing resting upon it (assuming we disallow moving stacks of objects) as well as appropriate postconditions. Additionally, the domain semantics map this discrete action to a continuous motion planning problem of finding a collision free trajectory from the current state to a state where the gripper is positioned appropriately for grasping.Within this domain, we must find a *task-motion plan*, a collision-free continuous trajectory corresponding to a discrete series of task actions that satisfy the task specification.

We also include as input to our algorithm a classifier that can be queried for feasibility information on a motion planning problem instance (C-space and start, goal poses).For a complete, formal problem definition of standard TMP we point the reader to [1]. Our problem definition differs by including a classifier that maps a C-space and set of motion planning poses to true or false. $\mathcal{C} : \Xi \times 2^{\mathcal{Q}} \mapsto \{0, 1\}$ in the language of [27], where Ξ is a set of C-spaces and $\mathcal{Q}$ is a sequence of motion planning poses.

Definition III.1. *Task-Motion Plan*
A task and motion plan is a sequence of pairs of task operators and motion plans, $\mathcal{TMP} = \left(\left\langle a^{[0]}, Q^{[0]}\right\rangle, \ldots, \left\langle a^{[n]}, Q^{[n]}\right\rangle\right)$ *such that* $\left(a^{[0]}, \ldots, a^{[n]}\right)$ *is in the task language* $\mathcal{L}$ *(i.e., is a valid task plan) and for each pair* $\left\langle a^{[i]}, Q^{[i]}\right\rangle$ $Q^{[i]}$ *is a motion corresponding to task operator* $a^{[i]}$ *in the appropriate configuration:* $\text{first}(Q^{[0]}) = q^{[0]}$ *and for all subsequent* i*,* $\text{last}(Q^{[i]}) = \text{first}(Q^{[i+1]})$.

IV. Algorithm

In a typical TMP system, the task planner and motion planner iterate between finding high-level plans and refining the high-level plan into corresponding motion plans. Empirically, we have found that most of the run time for common manipulation problems is spent exploring motions that are ultimately infeasible (see Figure 4). To address this bottleneck, we *learn a classifier for motion feasibility that expands to new domains*. We use the classifier to quickly estimate feasibility of high-level actions, only attempting motion planning when the entire high-level plan is classified as feasible. By avoiding motion planning attempts for infeasible actions, we address the major bottleneck in TMP and improve scalability.

Crucially, we integrate our learned heuristic within a TMP framework [27] to produce a planning method that tolerates classification error. False positives (infeasible actions classified as feasible) are passed on to the motion planner, which will determine infeasibility based on its timeout; thus a classifier that always returns $feasible$ will yield the original algorithm without our learned heuristic having any effect. False negatives (feasible motions classified as infeasible) will cause us to defer motion planning on a feasible instance and may reduce performance; however, our overall framework will later retry the action, thus maintaining probabilistic completeness, assuming a probabilistically complete motion planner is used.

Algorithm 1 shows our overall algorithm. The algorithm has three broad phases: a task planning phase (lines 4-10), a feasibility heuristic phase (lines 11-18), and a motion planning phase (lines 20-29). The three phases repeat until finding a valid task-motion. Compared to prior work [27], we introduce a new feasibility heuristic phase that applies our learned classifiers for motion feasibility.

The task planning phase searches for a new, candidate high-level plan of up to *horizon* steps using the constraint-based planner described in [1]. If no new plan below length `horizon` exists (line 6), we extend `horizon` and the motion planning timeout, reset constraints ϕ, and search again for the high-level plan—potentially revisiting previous task plans with the increased motion planning timeout, which is necessary for probabilistic completeness (see Theorem 1). Once the high-level plan A is found, we pass it to the feasibility heuristic phase.

The feasibility heuristic phase uses learned classifiers to estimate the feasibility of each action in the high-level plan. For each action in the high-level plan A, we map the action to a motion planning problem—specifically, the free configuration space represented using a scene graph and goal pose—via the domain semantics function λ_ρ (line 14). Then, we call our classifier to estimate when the motion is feasible (line 15). If a motion is classified as infeasible, we update the constraint equations to block the infeasible motion (line 17) and return to the task planner. Otherwise, if all motions are feasible, we proceed to motion planning.

Finally, the motion planning phase attempts to refine each high-level action into a corresponding motion plan (line 22). If

motion planning succeeds for all high-level actions, we have successfully computed the task and motion plan. Otherwise, if motion planning for any action fails (exceeds the timeout), we update the constraint equations to find an alternate plan (line 25) and return to the task planner. Note that we reconsider this plan later to preserve completeness (see Theorem 1).

Algorithm 1: TMP with Learned Heuristic

```
   Input: 𝒟 = (ℒ, λ_α, λ_ρ, 𝒞, σ^[0], 𝒢) // TM Domain
   Input: motionTO, cutoffTime, horizon
   Output: T // Task-Motion Plan
1  (s^[0], γ^[0], q^[0]) ← σ^[0]; // state,scene,config
2  φ ← initialConstraints(s^[0], λ_α(γ^[0]), ℒ);
3  repeat
4      repeat /* Task Planning Phase      */
5          A ← nextTaskPlan(φ);
6          if ∅ = A then // UNSAT
7              horizon ← horizon + 1;
8              motionTO ← motionTO + 1;
9              φ ← reset(φ, λ_α(𝒢), horizon);
10     until A;
       /* Feasibility Heuristic Phase    */
11     f ← true;
12     if motionTO ≤ cutoffTime then
13         foreach a^[i] ∈ A do
14             m^[i] ← λ_ρ(a^[i]); // Domain
                Semantics
15             if ¬feasible(C, m) then
16                 f ← false;
17                 φ ← addConstraints(a^[i]);
18                 break;
19     if f then /* Motion Planning Phase  */
20         T ← ∅;
21         foreach a^[i] ∈ A do
22             Q^[i] ←
                motionPlan(m^[i], q^[i], motionTO);
23             if ∅ = Q^[i] then // timeout
24                 T ← ∅;
25                 φ ← addConstraints(a^[i]);
26                 break;
27             else
28                 q^[i+1] ← last(Q^[i]);
29                 T^[i] ← ⟨a^[i], Q^[i]⟩;
30 until T;
```

A. Training Motion Feasibility Classifiers

We train the motion feasibility classifier based on a given set of minimal, exemplar scenes. Each classifier is specific to a robot, a task action, and the minimal set of fixed obstacles in the scene (in our case, a table). The feature vector for the classifier encodes the geometry and positions of movable objects in the exemplar scene and the robot's motion goal. We obtain the ground truth, i.e., feasible or infeasible, for each feature vector by running a sampling-based motion planner with a long enough timeout to ensure that the probability of not finding a feasible motion plan is negligible. We label the data based on the result of this motion planning, and use the labeled data to train a supervised learner; specifically, we train a support vector machine, though other supervised learning techniques could be applied.

B. Applying To Complex Scenes

To reduce training time, we limit training to scenarios with two rectangular prisms manipulated by a UR5 mounted to a table. At runtime, however, tabletop scenarios can include arbitrary numbers of objects with arbitrary meshes. Using a finite feature vector (see Figure 2), we cannot perfectly represent scenes with an arbitrary mesh or number of objects. Instead, we expand to these domains by approximating them and/or decomposing them so they can be input to our classifier. We use the following approximations to handle these more complex scenes. Motivating insights and a proof of correctness are discussed in subsection V-C.

Compared to our training scenes, a runtime scene may have additional fixed or stationary objects. Each of these objects can either be of the same type as those used in training (rectangular prisms) or new objects types, including arbitrary meshes. We generalize in these cases as follows:

1) Additional Fixed Obstacles: The classifier can ignore additional fixed obstacles as justified below (see subsection V-C). Note that motion planning does not ignore these obstacles.

2) Multiple Objects: When working with more than two objects, we decompose the problem into estimating feasibility for each pair of objects. We consider the problem to be infeasible if any pair classifies as infeasible and feasible otherwise.

3) New Movable Objects: We approximate non-rectangular objects with a rectangular prism inscribed in the object.

We formally characterize these three domain expansions (subsection V-C) and empirically validate the approach (section VI) for a tabletop manipulation domain.

V. Analysis of the Algorithm

Now, we analyze the properties of our approach. First, we show that our use of a learned heuristic maintains probabilistic completeness, even when the classifier is wrong. Then, we discuss the impact of classification error. Finally, we characterize how the classifiers expand to more complex domains.

A. Probabilistic Completeness

Definition V.1. *Probabilistic Completeness*
An algorithm is probabilistically complete if, as the run time approaches infinity, the probability of the algorithm finding a solution, should one exist, approaches one.

Probabilistic completeness requires certain assumptions. For sampling-based motion planners, a path is assumed to have some clearance. Additionally, we assume for TMP that we can enumerate a sufficient set of placement locations and actions to solve the problem. Under these assumptions, the key to maintaining probabilistic completeness in our approach

is handling cases when the classifier is wrong, which we do by eventually ignoring the classifier. It should be noted that because we train the classifier using RRT-connect with a long timeout, even if a query is exactly the same as a training example the classifier could still be wrong because RRT-connect may have timed out on a feasible instance. Thus, ignoring the classifier is unavoidable. Then, the proof of probabilistic completeness follows [1].

Theorem 1. *Algorithm 1 is probabilistically complete.*

Proof. As the run time of the algorithm increases, the motion planning timeout increases without bound (line 8), while the cutoff time for ignoring the classifier remains constant. Eventually, the motion planning timeout is increased beyond the cutoff time. From this point on, the classifier is not used (line 15). Then `attemptMotion` can only be set to `true` and probabilistic completeness follows the proof of [1]: The task planner is complete and enumerates all valid task plans (line 4). The motion planner attempts to refine each candidate task plan (line 22). If a probabilistically complete motion planner is used, the probability of success approaches one as `motionTimeout` increases. We progressively increase `motionTimeout` with each increase of the task planning `horizon`, revisiting all prior candidate plans with the greater timeout. Thus over time, the probability of successfully refining a feasible candidate plan approaches one. □

B. Effect of Classification Error

Because we eventually ignore the classifier, classification errors do not impact completeness, but they do reduce the performance of our algorithm. The classifier may produce false positives—infeasible actions labeled as feasible—or false negatives—feasible actions labeled as infeasible. Planning is faster when the classifier produces a false positive than a false negative. A false positive will be passed on to the motion planner, which must then explore the infeasible action until timeout. Thus, a classifier that always labels every action as feasible will be identical to the algorithm in [1]. A false negative, however, may be significantly more expensive. The false negative action will not be passed to the motion planner until after the motion planning timeout has exceeded the cutoff timeout (line 15), potentially spending significant time to explore additional plans before the cutoff. Thus, we bias the classifier to prefer false positives to false negatives. Classification errors have two causes: errors from the classifier proper and errors from domain expansion (see subsection IV-B). We first discuss expansion errors and then later discuss errors from the classifier proper.

C. Analysis of Domain Expansion

We analyze the classification errors introduced by our domain expansions and their effect in the overall planning framework. To simplify this discussion, we first assume the classifier is an oracle for motion feasibility queries on scenes with two prisms; in subsection VI-A we will discuss error with the classifier proper. We prove that applying the oracle to complex objects and environments via this domain expansion cannot produce additional false negatives, which would adversely effect computation time. These theoretical results have important implications for the applicability of our work and are confirmed by our experiments.

Definition V.2 (Oracle configuration space)**.** *Feature vector* $\mathbf{f}$ *and classifier* c *represent a query in free C-space,* ξ_{oracle}*, where:*

- *Movable objects have geometry and positions from* f
- *Fixed obstacles and robot are internal to* c*, i.e., based on the training data*

To apply the oracle to objects beyond pairs of rectangular prisms, we must approximate the true runtime C-space, ξ_{run}, with the feature vector. The relationship between ξ_{run} and ξ_{oracle} affects classification error. Any difference between ξ_{oracle} and ξ_{run} may induce classification errors. A plan that is feasible in ξ_{run} but not in ξ_{oracle} induces a false negative, while a plan that is not feasible in ξ_{run} but is feasible in ξ_{oracle} induces a false positive.

Because we prefer false positives to false negatives (see subsection V-B), we want an oracle free C-space ξ_{oracle} that over-approximates the true runtime free C-space ξ_{run}. That is, every valid plan through ξ_{run} must also be valid in ξ_{oracle} (no induced false negatives), but a valid plan in ξ_{oracle} may be invalid in ξ_{run} (induced false positives). This positive-biased approximation holds when ξ_{run} is a subset of ξ_{oracle}.

Theorem 2 (C-space approximation)**.** *If* $\xi_{\text{run}} \subseteq \xi_{\text{oracle}}$*, then the C-space approximation will not increase the false negative ratio of a given classifier.*

Proof. Assume to the contrary that the approximation may induce a false negative and thus defer a feasible action. Then, there exists some motion plan Q that is feasible in ξ_{run} but not feasible in ξ_{oracle}. But this contradicts $\xi_{\text{run}} \subseteq \xi_{\text{oracle}}$. □

Thus, when $\xi_{\text{run}} \subseteq \xi_{\text{oracle}}$, our approximation of the true free C-space induces no false negatives.

Motivated by this observation, we expand our method to new domains in the following ways:

1) Additional Fixed Obstacles: Adding additional fixed obstacles at runtime (e.g., a shelf too large for the robot to manipulate) causes the true free C-space to under approximate the oracle free C-space. Thus we can use the oracle in these scenarios without introducing new false negatives (Figure 5d).

2) Multiple Objects: When working with more than two objects, we can decompose the problem into estimating feasibility for pairs of objects. Each pair involves the moving object and one of the other objects. Thus there will be $n-1$ pairs in a scene with n modeled objects. The problem is considered infeasible if any call fails and feasible otherwise (Figure 5c).

3) New Movable Objects: Adding new movable objects, e.g., glasses or teapots, that are under-approximated by inscribed rectangular prisms, causes the true free C-space to under approximate the oracle free C-space. Thus we can use the classifier based on these inscribed prisms without introducing new false negatives (Figure 5e).

Finally, we note that any subset of these domain expansions can be safely combined without introducing new false negatives (Figure 5f).

In the preceding, we assumed that we had an oracle for feasibility. Based on this analysis of an oracle, it follows that *a classifier without false negatives on the exemplar scenes expands well, imposing at most the overhead of calling the classifier*. While assuming an oracle may be reasonable in simple domains, a more complex domain will make this unlikely. In practice we try to use a classifier with few or no false negatives. The classifier can be statistically biased to avoid false negatives, but in our experiments, we found this unnecessary. It should be noted that our domain expansions will tend to reduce the number of false negatives in the framework, but our proof only guarantees that it will not introduce new ones. To improve overall scalability, the classifier must generate true negatives to guide the search. We demonstrate these improvements empirically.

VI. Experiments

We use experiments to provide training data to our SVM and to test the overall method. Training experiments involve two rectangular prisms and the fixed table. Ideally an SVM would be trained for each new placement of fixed obstacles, but this is not mandatory (see Theorem 2). Each training point for the SVM is generated by running RRT-Connect three times with a 30 second timeout and checking if any run solves the instance. Each test scenario for the overall method is run in 10 independent trials.

A. Training the SVM

While manipulating one object in the presence of a stationary object, we are learning to predict motion feasibility given the locations and the sizes of the objects relative to the robot. We use the method of support vector machines (SVM) for our prediction, due to the low memory footprint and speed (it may be called thousands of times for a single problem and only takes a few milliseconds per call). Other classification models could be used, especially in more complex environments, but we found the increased accuracy from using a neural network did not justify the additional costs to train, store and evaluate the network.. Figure 2 illustrates the feature vector for an example training scene. We use polar coordinates d and θ describing the locations of the objects relative to the robot and to each other, and we use the heights h, widths w, and lengths ℓ of both the objects under consideration.

The data set consists of roughly 10,000 runs of pairs of rectangular prisms placed on a grid, which took about two days to generate on a desktop machine. Note that this time is spent running motion planning queries to produce a labeled data set. This time could be reduced, but we chose to attempt each query multiple times with long timeouts to ensure a reliable set of training data. The prisms vary in size, but are always axis-aligned to increase the consistency and minimize the size of the feature vector. The prisms are placed on a grid with cells 10cm x 10cm throughout reachable portions of the table. The objective is to move one of the prisms to a specified open space. We do this because we are interested in calculating whether the specified object can be usefully manipulated. The capability to pick up the object is not sufficient, since picking up an object changes the free C-space which can restrict the ability to manipulate. Because we are not modeling dynamics, our trajectories are reversible. Thus, we can use the same data set to learn about picking up and putting down an object. This reuse reduces the training time. With this exception, we train a separate classifier for each action, in our case for each grasp face. Because the runs rely on a sampling-based planner, we attempt the same problem instance up to three times to achieve an accurate ground-truth for the classifier.

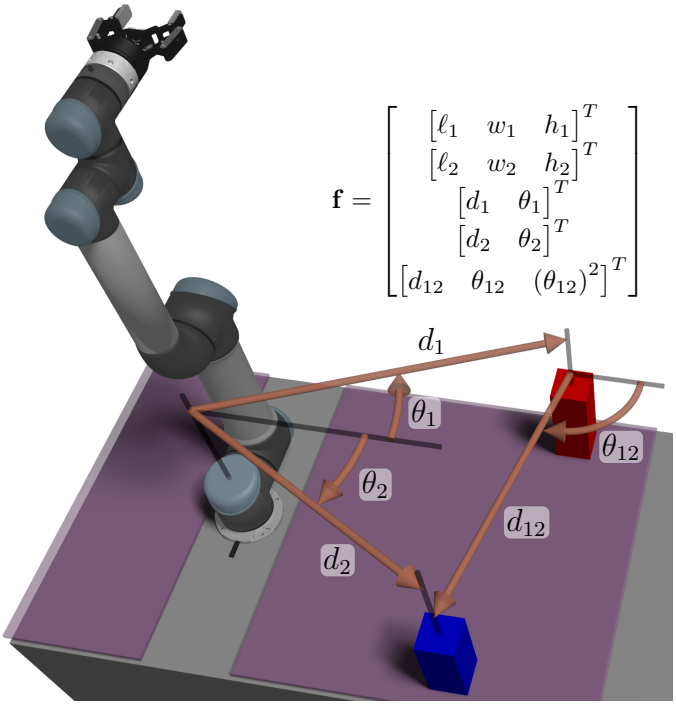


Fig. 2: Representation of polar coordinates used in the feature vector.

We use the open-source library LIBSVM's implementation of a C-SVC with a radial basis function for the kernel and the parameters of the C-SVC set to $cost = 5000$ and $gamma = 0.95$ [38]. With these settings, LIBSVM takes less than 10 seconds to fit the data on a desktop machine. On a hold out sample of scenes similar to training, that is, scenes consisting of two prisms, the classifier is correct approximately 96% of the time. For a more complicated scenario, such as a mobile robot, the classification rate may be lower. In such a case, a classifier can be biased to prefer false positives over false negatives to maintain the desirable properties discussed in section V. We show a confusion matrix in Figure 3.

Classifier Feasibility	True Feasibility: Positive	True Feasibility: Negative
Positive	501 ± 8.86	10.9 ± 4.67
Negative	12.7 ± 3.71	76 ± 8.72

Fig. 3: SVM's confusion matrix from 10-fold cross-validation

B. Runtime Experiments

All experiments are run on a desktop machine and the run times are averaged across 10 independent runs with error bars

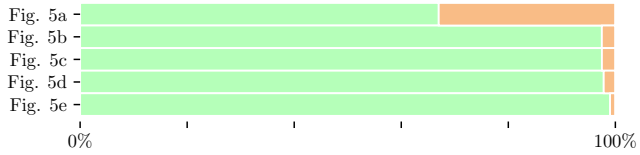


Fig. 4: Time spent on task planning (orange) and motion planning (green) for the given scenes.

showing one standard deviation. We use default settings for TMKit with the exception that the motion planning timeout is set to 30 seconds as this was found necessary for the scene in Figure 5d Below we include figures of scenes used to test our new approach along with run time results and an explanation of why these scenes were chosen.

The scene in Figure 5a is chosen to estimate the drop in the performance due to feasibility heuristic in scenarios when it is not needed, i.e., where the first task plan chosen by TMKit contains only feasible motions. There are two moveable blocks (blue and red) that can be placed on the purple regions of the table. The initial scene is given on the left and the final on the right. The blue block is moved to a location that is not obstructed by the red block, so the motion planner succeeds on the first task plan it attempts to refine. As expected, because it has the additional overhead of calling the SVM, the new approach is slower. However, the drop in the performance is only marginal.

The scene in Figure 5b is chosen to test the performance increase in the simplest possible example where the initial task plan chosen by TMKit will encounter infeasible motion. In this setting, the blue block is obstructed by the red block, which must be replaced to the original position for a feasible plan. The motion planner needs to evaluate several task plans before it succeeds in moving the blue block to its goal location. As expected the new approach is significantly faster than the old because it avoids expensive planning while attempting infeasible motions.

Next we look at scenes which differ significantly from the scenes used to train the SVM classifier. Our goal is to understand how well our approach expands to new domains.

The scene in Figure 5c is chosen to test our method of dealing with more than two objects. We can also contrast it with other works [34] [33] that are tested in similar environments. Note however, that our robot is stationary while the existing works use a mobile PR2.

The scene in Figure 5d is chosen to demonstrate the scaling of our approach to environments with additional obstacles. The shelf is fixed but causes many motions that were feasible to become infeasible (false positives).

The scene in Figure 5e is chosen to demonstrate scaling of our approach to more complex objects. The motions of the teapot are classified as if they were motions of a rectangular prism fitting inside the teapot handle. Note that we must use the handle rather than the body of the teapot to preserve the correct offset from the gripper.

The scene in Figure 5f tests all of the domain expansions together. The cylinder on the table is a fixed obstacle. The wineglasses are approximated by rectangular prisms (here we use a tighter approximation and treat the cup as if it were not hollow). Our experiments show that feasibility prediction can significantly improve the overall run time of the TMP framework in tabletop environments.

VII. Conclusion

We have demonstrated an approach that utilizes approximations and decompositions to robustly combine learning and planning in TMP frameworks. We prove that our learned heuristic expands from simple to complex scenes and that our framework maintains probabilistic completeness. We demonstrate results on a variety of tabletop manipulation scenarios, showing order-of-magnitude performance improvements.

Limitations of our approach include the need for substantial training data, the assumptions of rigid objects and connected C-space, the domain expansions loss of detail and the need to retrain for different robots.

There are several possibilities to improve our method. Ideal performance requires that objects can be well approximated by rectangular prisms. While we offer a domain expansion that uses prisms to under approximate arbitrary meshes, the classifier may fail to prune many motions that are infeasible. A richer feature vector and alternative learning techniques could give further speed improvements.

The training time is substantial, though the cost is amortized over all planning runs. It is possible that beneficial results could still be achieved by reducing the training time and statistically biasing the SVM to prefer false positives.

While we have only tested in a pick-place scenario (with stacking), the framework we have tested supports various actions. We would like to expand our heuristic to more general scenarios, such as a mobile robot that must move around the room and choose a location from which to attempt manipulation. Further work can incorporate sensing, uncertainty, and probabilities that motion planning instances are infeasible rather than a binary output.

References

[1] N. T. Dantam, Z. Kingston, S. Chaudhuri, and L. E. Kavraki, "An incremental constraint-based framework for task and motion planning," *International Journal of Robotics Research*, 2018.

[2] L. P. Kaelbling and T. Lozano-Pérez, "Hierarchical task and motion planning in the now," in *Int. Conf. on Robotics and Automation (ICRA)*. IEEE, 2011, pp. 1470–1477.

[3] S. Srivastava *et al.*, "Combined task and motion planning through an extensible planner-independent interface layer," in *Int. Conf. on Robotics and Automation (ICRA)*. IEEE, 2014, pp. 639–646.

[4] R. E. Fikes and N. J. Nilsson, "STRIPS: A new approach to the application of theorem proving to problem solving," *Artificial Intelligence*, vol. 2, no. 3, pp. 189–208, 1972.

[5] M. Helmert, "The fast downward planning system." *Journal Artificial Intelligence Resesearch*, vol. 26, pp. 191–246, 2006.

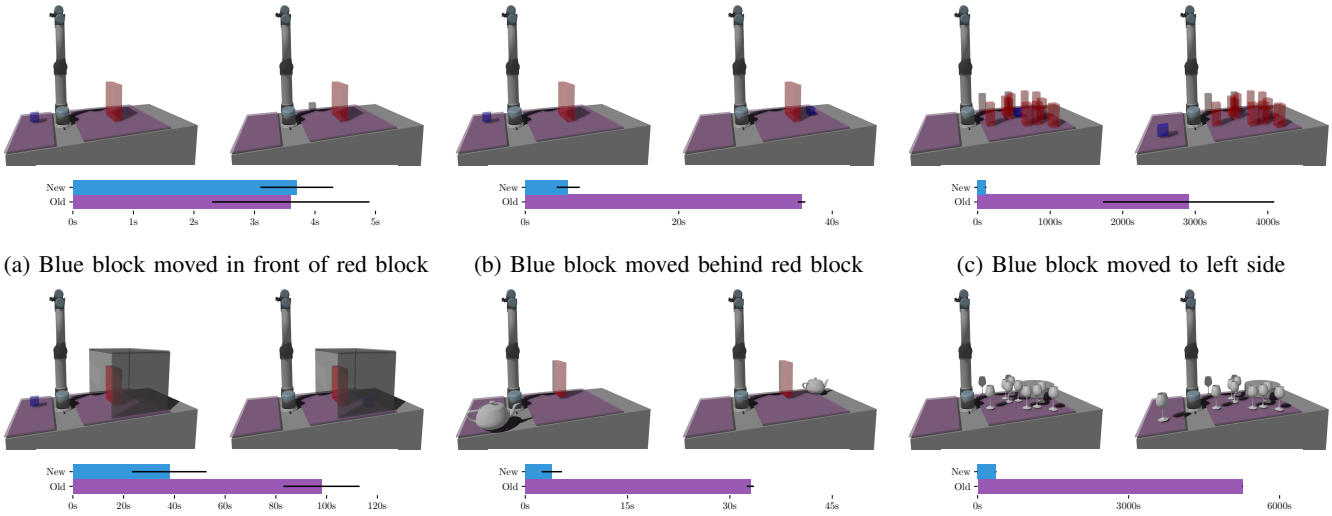
[6] J. Hoffmann and B. Nebel, "The FF planning system: Fast plan generation through heuristic search," *Journal of Artificial Intelligence Research*, pp. 253–302, 2001.

[7] V. Vidal, "YAHSP3 and YAHSP3-MT," in *8th Int. Planning Competition (IPC-2014)*, 2014, pp. 64–65.

[8] H. Kautz and B. Selman, "Unifying SAT-based and graph-based planning," in *Int. Joint. Conf. on Artifial Intelligence (IJCAI)*, vol. 99, 1999, pp. 318–325.

[9] J. Rintanen, "Madagascar: Scalable planning with sat," in *8th Int. Planning Competition (IPC-2014)*, 2014, pp. 66–70.

[10] J. Schulman *et al.*, "Motion planning with sequential convex optimization and convex collision checking," *Int. Journal of Robotics Research (IJRR)*, vol. 33, no. 9, pp. 1251–1270, 2014.



(a) Blue block moved in front of red block (b) Blue block moved behind red block (c) Blue block moved to left side

(d) Blue moved behind red block (e) Teapot moved behind red block (f) Empty wineglass moved to left side

Fig. 5: Evaluation on tabletop scenes. Times are averaged over ten runs and plotted with an error bar showing one standard deviation.

[11] M. Zucker *et al.*, "Chomp: Covariant hamiltonian optimization for motion planning," *The Int. Journal of Robotics Research*, vol. 32, no. 9-10, pp. 1164–1193, 2013.

[12] L. E. Kavraki, P. Švestka, J.-C. Latombe, and M. H. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *Trans. on. Robotics and Automation (T-RO)*, vol. 12, no. 4, pp. 566–580, 1996.

[13] S. M. LaValle and J. J. Kuffner, "Rapidly-exploring random trees: Progress and prospects," in *Algorithmic and Computational Robotics: New Directions*, B. Donald, K. Lynch, and D. Rus, Eds. A. K. Peters/CRC Press, 2001, pp. 293–308.

[14] S. M. LaValle, *Planning algorithms*. Cambridge university press, 2006.

[15] I. A. Şucan, M. Moll, and L. E. Kavraki, "The open motion planning library," *Robotics & Automation Magazine (RAM)*, vol. 19, no. 4, pp. 72–82, 2012.

[16] B. Cohen, S. Chitta, and M. Likhachev, "Single and dual-arm motion planning with heuristic search," *Int. Journal of Robotics Research (IJRR)*, vol. 3, no. 2, pp. 305–320, February 2014.

[17] F. Lagriffoul, D. Dimitrov, A. Saffiotti, and L. Karlsson, "Constraint propagation on interval bounds for dealing with geometric backtracking," in *Int. Conf. on Intelligent Robots and Systems (IROS)*. IEEE, 2012, pp. 957–964.

[18] F. Lagriffoul and B. Andres, "Combining task and motion planning: A culprit detection problem," *The Int. Journal of Robotics Research*, vol. 35, no. 8, pp. 890–927, 2016.

[19] J. Ferrer-Mestres, G. Francès, and H. Geffner, "Combined task and motion planning as classical AI planning," *CoRR*, vol. abs/1706.06927, 2017.

[20] W. Vega-Brown and N. Roy, "Asymptotically optimal planning under piecewise-analytic constraints," in *Workshop on the Algorithmic Foundations of Robotics*, 2016.

[21] S. Cambon, R. Alami, and F. Gravot, "A hybrid approach to intricate motion, manipulation and task planning," *Int. Journal of Robotics Research*, vol. 28, no. 1, pp. 104–126, 2009.

[22] F. Gravot, S. Cambon, and R. Alami, "aSyMov: a planner that deals with intricate symbolic and geometric problems," in *Int. Symp. on Robotics Research (ISRR)*. Springer, 2005, pp. 100–110.

[23] A. Bhatia, M. R. Maly, L. E. Kavraki, and M. Y. Vardi, "Motion planning with complex goals," *Robotics & Automation Magazine (RAM)*, vol. 18, no. 3, pp. 55–64, Sept. 2011.

[24] K. He, M. Lahijanian, L. E. Kavraki, and M. Y. Vardi, "Towards manipulation planning with temporal logic specifications," in *Int. Conf. on Robotics and Automation (ICRA)*. Seattle, WA: IEEE, 2015, pp. 346–352.

[25] E. Plaku, L. E. Kavraki, and M. Y. Vardi, "Motion planning with dynamics by a synergistic combination of layers of planning," *IEEE Trans. on Robotics*, vol. 26, no. 3, pp. 469–482, 2010.

[26] G. E. Fainekos, H. Kress-Gazit, and G. J. Pappas, "Temporal logic motion planning for mobile robots," in *Proceedings of the 2005 IEEE Int. Conf. on Robotics and Automation*, 2005, pp. 2020–2025.

[27] N. T. Dantam, S. Chaudhuri, and L. E. Kavraki, "The task motion kit," *Robotics and Automation Magazine*, 2018.

[28] S. Yoon, A. Fern, and R. Givan, "Learning heuristic functions from relaxed plans," in *Proceedings of Interenational Conf. on Automated Planning and Scheduling*, ser. ICAPS'06, 2006.

[29] Y. Xu, A. Fern, and S. Yoon, "Discriminative learning of beam-search heuristics for planning," in *Intl. Joint Conf. on Artifical Intelligence*, 2007, pp. 2041–2046.

[30] O. Arslan and P. Tsiotras, "Machine learning guided exploration for sampling-based motion planning algorithms," in *2015 IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, Sept 2015, pp. 2646–2652.

[31] S. Schaal, *Dynamic Movement Primitives -A Framework for Motor Control in Humans and Humanoid Robotics*. Tokyo: Springer Tokyo, 2006, pp. 261–280.

[32] C. R. Garrett, T. Lozano-Pérez, and L. P. Kaelbling, "Ffrob: Leveraging symbolic planning for efficient task and motion planning," *CoRR*, vol. abs/1608.01335, 2016.

[33] R. Chitnis *et al.*, "Guided search for task and motion plans using learned heuristics," in *Int. Conf. on Robotics and Automation (ICRA)*. IEEE, 2016.

[34] B. Kim, L. Pack Kaelbling, and T. Lozano-Perez, "Learning to guide task and motion planning using score-space representation," in *Int. Conf. on Robotics and Automation (ICRA)*. IEEE, 2017.

[35] B. Ichter, J. Harrison, and M. Pavone, "Learning sampling distributions for robot motion planning," *CoRR*, vol. abs/1709.05448, 2017.

[36] M. Phillips, B. Cohen, S. Chitta, and M. Likhachev, "E-graphs: Bootstrapping planning with experience graphs," in *Robotics Science and Systems (RSS)*, Sydney, Australia, 07/2012 2012.

[37] D. Coleman *et al.*, "Experience-based planning with sparse roadmap spanners," *CoRR*, vol. abs/1410.1950, 2014.

[38] C.-C. Chang and C.-J. Lin, "LIBSVM: A library for support vector machines," *ACM Transactions on Intelligent Systems and Technology*, vol. 2, pp. 27:1–27:27, 2011.