Simultaneous Action and Grasp Feasibility Prediction for Task and Motion Planning through Multi-Task Learning

Smail Ait Bouhsain, Rachid Alami, Thierry Simeon

Simultaneous Action and Grasp Feasibility Prediction for Task and Motion Planning through Multi-Task Learning

Smail Ait Bouhsain[1], Rachid Alami[1] and Thierry Siméon[1]

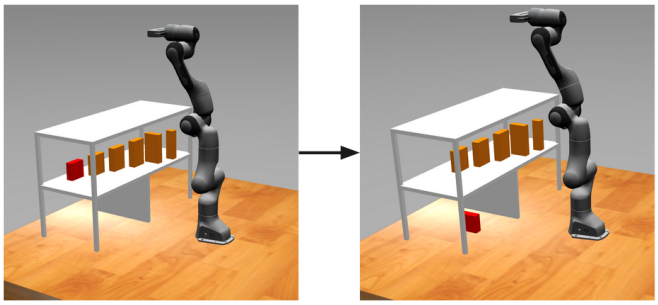
***Abstract*— In this paper, we address task and motion planning (TAMP) which is an important yet challenging robotics problem. It is known to suffer from the high combinatorial complexity of discrete search, often requiring a large number of geometric planning calls. We build upon recent works in TAMP by taking advantage of learning methods to provide action feasibility information as a heuristic to the symbolic planner, thus guiding it to a geometrically feasible solution and reducing geometric planning time. We propose AGFP-Net, a multi-task neural network predicting not only action feasibility, but also the feasibility of a set of grasp types. We also propose an improved feasibility-informed TAMP algorithm capable of solving more complex problems, and handling goals which are not fully specified. Comparative results obtained on different problems of varying complexity show that our method is able to greatly reduce task and motion planning time.**

I. Introduction

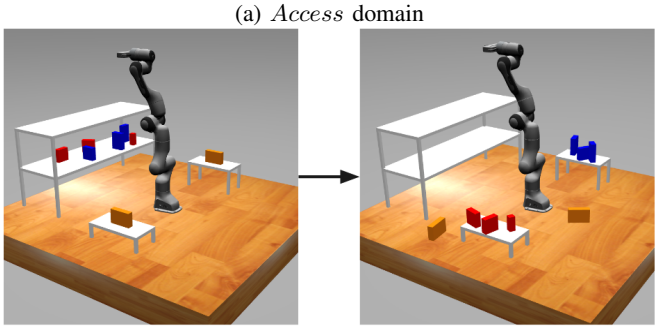
Task and motion planning (TAMP) [1]–[30] is a robotics problem that involves determining a sequence of actions that a robot must take to achieve a desired goal, along with the corresponding motions of the robot. It combines discrete symbolic planning with continuous geometric planning. TAMP problems are challenging due to the high dimensionality of the state and action spaces, but also to the combinatorial complexity of discrete search.

Geometric planning is a major bottleneck to TAMP. Since symbolic planners do not have any geometric reasoning capabilities, they generate task plans without a guarantee of feasibility. A geometric planner has to be used in order to verify the feasibility of each generated task plan and construct the corresponding motions. However, in complex problems, symbolic planners generate a high number of geometrically infeasible solutions before finding a feasible one, each plan requiring a call to the geometric planner which results in long planning time. Moreover, geometric planning might be time consuming even in the case of feasible tasks. Since not all grasps are feasible or lead to a feasible motion, finding the right one might require a lot of time.

This work aims at providing the symbolic planner with a feasibility prediction capability, which increases the probability that the generated task plans are feasible, and thus reduces the number of calls to the geometric planner. First, we improve upon our previous approach [30] in order to enrich the information provided to the task planner, and reduce the dimensionality of the action space. Indeed, instead of predicting the feasibility of a *Pick* or *Place* action with a single specific grasp type, we propose a multi-task learning neural network capable of predicting the feasibility of an action as well as the feasibility of each grasp type. We also propose a new training strategy allowing a better generalization to new environments. In addition, we propose a more powerful TAMP algorithm capable of solving a wider range of problems with higher complexity, in particular ones with semi-specified goals. We develop a method for combining probabilities of feasibility allowing the task planner to make better use of the predictions obtained by the neural network, and reduce the branching factor of the search. We also introduce a new cost function that incorporates both the action feasibility as well as the grasp feasibility predictions. Furthermore, we leverage the latter to accelerate geometric planning, thus reducing the time spent on finding a grasp that leads to a feasible motion.



(a) *Access* domain



(b) *Sort* domain

Fig. 1: A visualization of the initial and goal states of two complex TAMP problems solved by the proposed planner, and on which Bouhsain et al. 2023 [30] fails.

II. Related Work

Task and motion planning combines discrete symbolic planning and continuous geometric planning. Early works [1]–[11] view the problem from the geometrical perspective, by formalizing it as a multi-modal motion planning problem.

[1]LAAS-CNRS, Toulouse, France, {saitbouhsa, alami, simeon}@laas.fr

Most existing TAMP methods [12]–[20] combine a task planner with a geometric planner, while using geometric backtracking as an interface between the two. These methods are known to suffer from the high combinatorial complexity of discrete search and the slow geometric planning time, since a call to the geometric planner is still needed to verify the feasibility of symbolic plans.

Recent works in TAMP [21]–[30] propose to accelerate the planning process by leveraging learning methods. They either aim at guiding the planning process towards a solution, or reducing the number of calls to the geometric planner by providing a learned heuristic to the task planner. Learning approaches have also been proposed for grasp planning [31]–[34], these works aim at finding possible grasps of complex objects, which is a step of TAMP often overlooked. Works such as [22] [24] [25] propose to train a learning model to predict the geometrical feasibility of $Pick$ and $Place$ actions in tabletop environments. Bouhsain et al. [30] extend this idea to 3D environments and proposes a feasibility-informed task and motion planner which takes advantage of feasibility prediction. This work improves upon [30] by proposing a multi-task learning neural network providing richer geometric feedback to the task and motion planner. We also develop a TAMP algorithm which uses a new method for handling feasibility predictions, and is capable of solving more complex problems, for instance ones with semi-specified goals.

III. Problem Description

We focus on manipulation problems involving the rearrangement of a set of objects in a three-dimensional manipulation environment E. The latter is composed of a single robot arm with a parallel gripper, n_O box-shaped movable objects, n_{ss} stable support surfaces and n_{obs} fixed obstacles. A state s of the planning scene is defined by the configuration of the robot and of all movable objects, i.e the support surface and the pose of each object. We denote the configuration of a movable object O in the state s as $s(O)$. An action a is defined as the transition between two distinct states s and s'. It involves picking a single movable object O from its configuration in s, and placing it at a new configuration $\mathbf{q}$. The action a can then be explicitly formulated as:

$$a = Move_O(s(O) \rightarrow \mathbf{q}) \tag{1}$$

Finding a solution to a TAMP problem consists in finding a sequence of actions $\tau = \{a_0, a_1, ..., a_K\}$, as well as the corresponding sequence of motions $\Pi = \{\pi_0, \pi_1, ..., \pi_K\}$, bringing the environment from an initial state s_0 to a goal state s_{goal}. Each movable object might have either a fully-specified goal placement (i.e a support surface and a pose), or a semi-specified goal placement (i.e a support surface only), or no specified goal at all.

We also define G as the set of 6 grasp types described in [30]. They correspond to the side from which the object is grasped in the perspective of the robot. Hence, each grasp type is a subspace of grasps from the corresponding side of the object.

IV. Action and Grasp Feasibility Prediction

As in our previous work [30], we aim at reducing the number of calls to the geometric planner by providing a feasibility prediction capability to the symbolic planner, allowing it to generate task plans with a high probability of feasibility first. In this paper, rather than predicting the feasibility of an action with a specific grasp type, we train a neural network to simultaneously predict the general feasibility of a $Pick$ or $Place$ action without considering the grasp, as well as the feasibility of each grasp type. This means that, contrarily to the previous method, the grasp is no longer an input to the neural network but rather an output.

A. Neural Network

The proposed neural network takes as input the 3D scene and the action to be tested. We use the same representation of the 3D environment as in [30], using 5 depth images which correspond to different views of the scene (i.e top, front, rear, right, left). Each view corresponds to a specific plane in the 3D space. The top view shows a projection of the environment on the (x, y) plane, the front and rear views correspond to the (y, z) plane, and the right and left views correspond to the (x, z) plane.

Regarding action representation, we consider $Pick$ and $Place$ tasks as equivalent. Indeed, if we construct these actions as starting and ending at the same home configuration of the robot, the motion for placing an object at a configuration is the same as the reversed motion of picking the same object from the same configuration. Using this assumption[1], we always consider the action to be tested as a $Pick$ task, which reduces the number of inputs to the neural network, and allows to speedup data generation and annotation (Section IV-B).

As in [30], the object of interest is represented using 5 masks over the scene views, showing only the object at the configuration it should be picked or placed at. Instead of combining the scene views and the object masks into a single 10-channel image, we combine them into 3 separate images depending on the plane the views correspond to. The first is a 2-channel image I_{xy} consisting of the top scene view and mask. The second and third are two 4-channel images I_{xz} and I_{yz}, consisting of the left-right and the front-rear scene views and masks respectively[2]. These images are encoded using three ResNet-based convolutional neural networks [35]. The resulting embeddings are concatenated, then fed to a fully connected layer to obtain an encoding of the 3D scene and the action. A multi-layer perceptron followed by a Sigmoid activation function are then used to obtain 2 predictions. The first one is the probability of feasibility of the action p_F. The second is a vector $\mathbf{p}_G$ composed of 6 values, each one being the probability of

[1]Note that this assumption is used only at the feasibility prediction level. The geometric planner used in our TAMP approach considers $Pick$ and $Place$ as different tasks that might start and end at different configurations of the robot.

[2]This separation of the views allows a more sane scene representation and prevents the neural network from confusing the views

feasibility of one of the grasp types in G. Figure 2 shows the complete architecture of our proposed neural network.

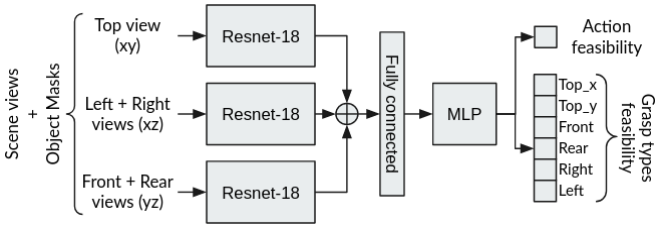


Fig. 2: The architecture of the proposed neural network **AGFPNet**.

B. Data Generation and Annotation

In order to train the proposed neural network, we develop an automatic data generation method which can be divided into three steps. The first is scene generation, which follows the same procedure detailed in [30]. It generates scenes containing a large table, up to 4 elevated support surfaces, and 2 objects. The second is the construction of the dataset. For each object O in a generated scene, we generate one $Pick$ action from its configuration $\mathbf{q}$. Using the assumption that $Pick$ and $Place$ actions are equivalent, these generated $Pick$ actions also count as $Place$ actions at the same configuration $\mathbf{q}$.

The third step of data generation is annotation. In this step, we need to label each generated action but also each grasp type as feasible or infeasible. We consider a grasp type as feasible if at least one grasp corresponding to this type is feasible. All the generated actions causing a collision are guaranteed to be infeasible, and are thus automatically labeled as such. We annotate the rest of the actions using a geometric planner composed of a grasp sampler, an inverse kinematics solver and a sampling based motion planner.

V. TAMP ALGORITHM

Compared to our prior work [30], we propose an improved TAMP algorithm, capable of solving a wider range of more complex problems. In addition to leveraging feasibility predictions to accelerate the planning process, the branching factor of our task planner is reduced by using a different definition of actions at the symbolic level. While in our previous algorithm, an action was defined as either a $Pick$ or a $Place$ task with a specific grasp type, in this work, we define actions as combined $Pick-Place$ tasks without any grasp type specified.

The main algorithm remains similar to our previous planner. It combines a symbolic task planner which finds a sequence of actions bringing the environment to a goal state, and a geometric planner which checks the feasibility and finds the motions corresponding to each action. The task planner is a best-first tree search in which nodes are states of the environment and edges are actions, as defined in Section III. At each iteration, the node s with the lowest cost is retrieved from the set of open nodes. If s is a goal state, we reconstruct the complete action sequence τ leading to s. The geometric planner is then called to find the corresponding sequence of motions Π. If geometric planning succeeds, then a geometrically feasible solution was found and the problem is solved. On the other hand, if one of the actions is infeasible, we prune all children of the first infeasible node in the task plan from the tree, and continue the search.

Algorithm 1 findChildren

Input: s, E, s_{goal}
1: $children \leftarrow \emptyset$
2: $A \leftarrow findPossibleActions(s, E, s_{goal})$
3: **for** each a in A **do**
4: $\quad [p_F(a), \mathbf{p}_G(a)] \leftarrow predictFeasibility(s, E, a)$
5: $\quad child \leftarrow nextState(s, a)$
6: $\quad child.cost \leftarrow computeCost(child, s_{goal}, p_F(a))$
7: $\quad children \leftarrow children \cup child$
8: **end for**
9: **return** $children$

Algorithm 2 findPossibleActions

Input: s, E, s_{goal}
1: $A \leftarrow \emptyset$
2: **for** each O in movable objects **do**
3: $\quad$ **if** $s(O) \notin s_{goal}(O)$ **then**
4: $\quad\quad$ **if** $s_{goal}(O)$ is fully-specified **then**
5: $\quad\quad\quad a \leftarrow Move_O(s(O) \rightarrow s_{goal}(O))$
6: $\quad\quad\quad A \leftarrow A \cup a$
7: $\quad\quad$ **else if** $s_{goal}(O)$ is semi-specified **then**
8: $\quad\quad\quad R \leftarrow samplePlacements(E, s_{goal}(O))$
9: $\quad\quad\quad$ **for** each $\mathbf{q}$ in R **do**
10: $\quad\quad\quad\quad a \leftarrow Move_O(s(O) \rightarrow \mathbf{q})$
11: $\quad\quad\quad\quad A \leftarrow A \cup a$
12: $\quad\quad\quad$ **end for**
13: $\quad\quad$ **end if**
14: $\quad$ **end if**
15: $\quad R \leftarrow sampleRandomPlacements(E)$
16: $\quad$ **for** each $\mathbf{q}$ in R **do**
17: $\quad\quad a \leftarrow Move_O(s(O) \rightarrow \mathbf{q})$
18: $\quad\quad A \leftarrow A \cup a$
19: $\quad$ **end for**
20: **end for**
21: **return** A

In case s is not a goal node, we call the $findChildren$ function described in Algorithm 1. We first proceed to finding the actions applicable at s by calling the function $findPossibleActions$ detailed in Algorithm 2. For each movable object O in the scene, we generate two types of actions. The first action type tries to move the object O from its placement at s denoted $s(O)$, directly to its goal placement $s_{goal}(O)$. Within this type of actions, there are two different scenarios. If O has a fully-specified goal (i.e support surface and pose), then only one action is generated that moves the object to that goal (Alg. 2, lines 4-6). However, if O has a semi-specified goal (i.e it has to be placed on a specific support surface whatever the pose), then the number

of goal placements is infinite. In this case, we sample a fixed number of placements on the goal support surface. We then generate multiple actions, each one moving object O to one of the sampled placements (Alg. 2, lines 7-12).

The second action type aims at moving the object O to a random placement without considering the goal. We sample a fixed number of random placements, then generate multiple actions moving object O to each one of them (Alg. 2, lines 15-19). In both action types, the number of placements sampled is a parameter chosen by the user[3].

Algorithm 3 predictFeasibility

Input: s, E, a

1: $O \leftarrow getObjectToMove(a)$
2: $\mathbf{q}_{start} \leftarrow s(O)$
3: $\mathbf{q}_{end} \leftarrow getEndPlacement(a)$
4: $[p_F(Pick),\ \mathbf{p}_G(Pick)] \leftarrow predict(s, E, O, \mathbf{q}_{start})$
5: $[p_F(Place),\ \mathbf{p}_G(Place)] \leftarrow predict(s, E, O, \mathbf{q}_{end})$
6: $\mathbf{p}_G(a) \leftarrow \mathbf{p}_G(Pick) \otimes \mathbf{p}_G(Place)$
7: $p_F(a) = p_F(Pick) \times p_F(Place) \times max(\mathbf{p}_G(a))$
8: **return** $[p_F(a), \mathbf{p}_G(a)]$

For each possible action a found, we compute its probability of feasibility $p_F(a)$ as well as the probability of feasibility of each grasp type $\mathbf{p}_G(a)$ (Alg. 1, line 5). Since the neural network predicts the feasibility of $Pick$ and $Place$ tasks separately, we need a strategy for combining these predictions in order to obtain the probability of feasibility of the complete $Pick - Place$ action. This is done in the $predictFeasibility$ function described in Algorithm 3. First the action a is decomposed into a $Pick$ task and a $Place$ task. Then **AGFP-Net** is queried twice to obtain the probability of feasibility and the grasp types probabilities for the $Pick$ and $Place$ actions separately. For a single grasp type g, we define the probability $p_g(a)$ that the complete action a is feasible using g as follows:

$$p_g(a) = p_g(Pick) \times p_g(Place) \tag{2}$$

with $p_g(Pick)$ and $p_g(Place)$ being the probability that the $Pick$ and the $Place$ tasks, respectively, are feasible using g. Using this definition, we can obtain the complete vector of grasp types probabilities $\mathbf{p}_G(a)$:

$$\mathbf{p}_G(a) = \mathbf{p}_G(Pick) \otimes \mathbf{p}_G(Place) \tag{3}$$

where $\otimes$ represents the element-wise product.

Regarding the probability of feasibility of the action $p_F(a)$, multiplying the probabilities of the $Pick$ and $Place$ tasks is not sufficient. The reason is that even if both these tasks are feasible, the grasps leading to a feasible $Pick$ might not be the same as the ones leading to a feasible $Place$. Indeed, the complete action a is feasible iff. both the $Pick$ and the $Place$ tasks are feasible, and have at least one feasible grasp type in common. Therefore, we define $p_F(a)$ as:

$$p_F(a) = p_F(Pick) \times p_F(Place) \times max(\mathbf{p}_G(a)) \tag{4}$$

where $p_F(Pick)$ and $p_F(Place)$ are the probabilities of feasibility of the $Pick$ and the $Place$ tasks respectively, and $max(\mathbf{p}_G(a))$ is the highest probability in the vector $\mathbf{p}_G(a)$.

After the feasibility prediction step, we construct a new child as the result of applying action a to state s (Alg. 1, line 5). We then compute its cost which is defined as:

$$C_{Total} = C_{SoFar} + C_{ToGoal} + C_{Feasibility} \tag{5}$$

where C_{SoFar} is the number of actions in the branch leading to the child node, C_{ToGoal} is the number of objects that are not at their goal placement, and $C_{Feasibility}$ is a feasibility cost calculated using the expression:

$$C_{Feasibility} = \frac{1}{p_F(a)} - 1 \in]0, \infty[\tag{6}$$

This definition allows to incorporate the feasibility predictions in the state costs. Thus, the higher the probability of feasibility of an action leading to a state is, the lower its cost will be. After all the children are generated, they are added to the set of open nodes which is sorted by increasing cost.

One important novelty of this work is the use of grasp feasibility predictions to accelerate geometric planning. When the geometric planner is called to find the motion of an action a, we use the previously computed grasp type probabilities $\mathbf{p}_G(a)$ in order to prioritize feasible grasp types during grasp sampling. Thanks to this heuristic, the geometric planning time on feasible actions can be reduced.

VI. Experiments

In order to test our method, we first train and test the proposed neural network on a generated dataset. Then, we define a set of TAMP problems to evaluate the performance of our algorithm.

A. Neural Network Training and Testing

Using the data generation method described in IV-B, we construct a training dataset based on $20'000$ generated scenes. We also generate a validation set and a testing set, based on $5'000$ generated scenes each, in order to evaluate the performance of the trained neural network. For annotation, we use an adapted version of Moveit! Task Constructor [36] with a BiTRRT motion planner [37]. The model is implemented on Pytorch [38] and trained using a weighted binary cross entropy loss to account for the imbalance between feasible and infeasible datapoints. We use the ADAM optimizer with a learning rate of 0.0001 and a batch size of 128. To prevent overfitting, we use dropout with a probability of 0.1, and weight decay of 0.0001.

Moreover, in order to increase the size of our training set, we use online data augmentation. Since the robot can turn $360°$ around its base, rotating the scene around the z axis does not affect the feasibility of the action or the grasp types. This allows to obtain new labeled datapoints from already

[3]In case none of the sampled placements lead to a geometrically feasible solution, a node can be expanded more than once in order to resample more placements.

annotated ones. During training, scenes are either kept as they are, or rotated by a multiple of $\frac{\pi}{2}$ around the z axis. This data augmentation method improves training by allowing the neural network to learn more scenarios, and ensuring that the predictions are consistent between equivalent scenes. It also allows for a better generalization to new environments. The neural network is trained for 200 epochs, which takes approximately 24 hours.

B. Test Problems for the TAMP Algorithm

In order to compare the performance of our proposed approach with our prior work, we reuse the 5-object versions of the three TAMP problems described in [30]. The first is the $Reorder$ domain in which the goal is to move a set of movable objects from a shelf to another while changing the order. The second is the $Unpack$ problem, in which a set of objects on a tray have to be organized on a shelf. The third is the $Swap$ domain in which the goal is to swap the placements of different objects. In these problems, the challenge comes either from grasp choice, the proximity between objects, or the occupancy of goal placements.

We also define two more complex TAMP problems that demonstrate the additional capabilities of our new method. The $Access$ domain, shown in Figure 1a, involves moving a single object (red) to its goal placement while ensuring that the other movable objects are back at their initial placement. In this problem, the challenge comes from the fact that each object blocks access to the object next to it. The robot has to move all blocking objects one by one to a temporary placement, move the red object to its goal placement and finally put back the previously moved objects at their initial placements in the same order.

The $Sort$ problem, shown in Figure 1b aims at demonstrating the ability of our algorithm to handle semi-specified and unspecified goals. The goal is to move a set of objects from a shelf and to one of two small tables depending on their color. Here, the challenge comes from placement sampling, meaning that the planner must find the set of placements allowing all objects of the same color to be on a narrow support surface. The initial proximity of objects is also a challenge, since the objects have to be moved in a specific order. Moreover, we make the problem more complex by adding blocking objects (orange) with unspecified goals on the tables.

The $Access$ and $Sort$ problems also aim at testing the generalizability of **AGFP-Net** to more realistic scenes, with a higher number of objects and vertical obstacles.

VII. Results

We first analyze the performance of **AGFP-Net** on the generated test data. We then evaluate the performance of the proposed TAMP algorithm on the test TAMP problems.

A. Neural Network Performance

Table I shows the performance of the proposed neural network. Comparing the results obtained by **AGFP-Net** to the one yielded by [30] shows an overall improvement in performance with an F1-Score of 92.4%. Also, the yielded true positive rate (TPR) of 93.7% and true negative rate (TNR) of 97% show that **AGFP-Net** is as good at predicting feasibility as it is at predicting infeasibility. Regarding grasp feasibility prediction, the results show that the model is able to accurately predict the feasibility of each grasp type with an F1-Score of 86.6%, a TPR of 93.3% and a TNR of 97.1%.

TABLE I: Comparison of the performance of **AGFP-Net** and our prior model **AFP-Net**.

Model	Prediction Task	F1-Score	TPR	TNR
AFP-Net [30]	Action Feasibility	90%	91.5%	84.3%
AGFP-Net	Action Feasibility	92.4%	93.7%	97%
	Grasp Feasibility	86.6%	93.3%	98.1%

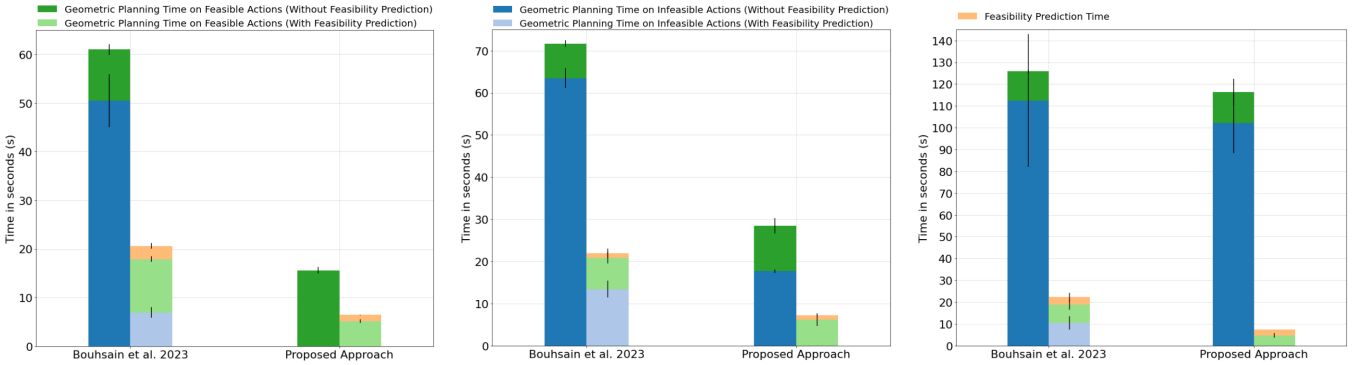
B. Performance of the feasibility-informed TAMP algorithm

We ran our algorithm on all five TAMP problems defined in Section VI-B with 10 runs each. In order to measure the performance gain of action and grasp feasibility prediction, we do the same with a non-informed version of our TAMP algorithm. This is done by disabling feasibility prediction and setting all probabilities of feasibility to 1.

TABLE II: Planning performances with and without using a feasibility heuristic, averaged over 10 runs.

Domain	Method	Heuristic	Infeasible Task Plans	Total Planning Time
Reorder	[30]	None	5.0	61.0
		AFP-Net	0.4	20.5
	Proposed	None	**0.0**	15.6
		AGFP-Net	**0.0**	**6.4**
Unpack	[30]	None	11.8	71.7
		AFP-Net	3.3	21.9
	Proposed	None	3.0	28.5
		AGFP-Net	**0.1**	**7.6**
Swap	[30]	None	30.6	125.8
		AFP-Net	3.9	22.2
	Proposed	None	17.9	117.2
		AGFP-Net	**0.2**	**7.4**
Sort	[30]	Not handled		
	Proposed	None	108.3	599.1
		AGFP-Net	**1.1**	**50.0**
Access	[30]	Planning Failure		
	Proposed	None	339.3	1500.1
		AGFP-Net	**1.9**	**95.2**

We first compare our approach to the one proposed in [30]. Figure 3 and Table II show a comparison of the two methods on the $Reorder$, $Unpack$ and $Swap$ problems. Results obtained without feasibility prediction show that the new TAMP algorithm generates less geometrically infeasible task plans, which translates into less geometric planning time spent on infeasible actions. This is due to the fact that grasp types are no longer considered by the task planner, but rather by the geometric planner. As a result, in problems where the grasp choice is important such as the $Reorder$ and $Unpack$ domains, the task planner does not generate task plans that are infeasible due to the choice of grasp. In the $Swap$ problem, however, this gain in performance is not



(a) *Reorder* domain (b) *Unpack* domain (c) *Swap* domain

Fig. 3: Planning time comparison between the proposed approach and Bouhsain et al. 2023 [30] averaged over 10 runs. For each method, we report the mean and standard deviation of the planning time with and without feasibility prediction, using **AFP-Net** for [30] and **AGFP-Net** for the proposed approach.

as pronounced since in this domain, infeasibility is mostly due to collisions.

On the other hand, comparing the performances yielded using the feasibility prediction heuristic shows that thanks to the improved accuracy of **AGFP-Net** (producing less misclassifications), and the new probability combination strategy, the proposed TAMP algorithm is able to reduce the number of infeasible task plans generated to 0 in most runs, even in problems where [30] has some failures such as the *Unpack* and the *Swap* domains. Figures 3b and 3c show that geometric planning time on infeasible actions is eliminated using our feasibility-informed planner.

Moreover, the results demonstrate the improvement in planning time yielded using grasp feasibility prediction. Indeed, comparing geometric planning time on feasible actions with and without **AGFP-Net** shows that using the grasp feasibility heuristic allows the geometric planner to find grasps leading to a feasible motion faster. This decrease in geometric planning time not only compensates for the time spent on feasibility prediction, but also reduces the total planning time as shown particularly in Figure 3a.

On the *Access* problem, the method proposed in [30] fails to find a geometrically feasible solution in a reasonable amount of time, whether feasibility prediction is used or not. This is due to the high combinatorial complexity of the problem. Thanks to the reduced branching factor, our algorithm is able to solve this problem in approximately 25 minutes when no feasibility heuristic is used. However, Table II shows that the number of infeasible task plans generated before finding a feasible solution is high with an average of 339.3. Using **AGFP-Net**, this number is reduced to an average of 1.9, yielding a 93% reduction in total planning time. For the *Sort* domain, the number of infeasible task plans generated is 108, which is reduced to 1 using **AGFP-Net**. Hence, the total planning time is 12 times faster.

Furthermore, these results show that, although **AGFP-Net** has been trained on scenes containing two objects only and no vertical obstacles, it is able to generalize to scenes with a higher number of objects and a limited number of obstacles while keeping feasibility predictions accurate.

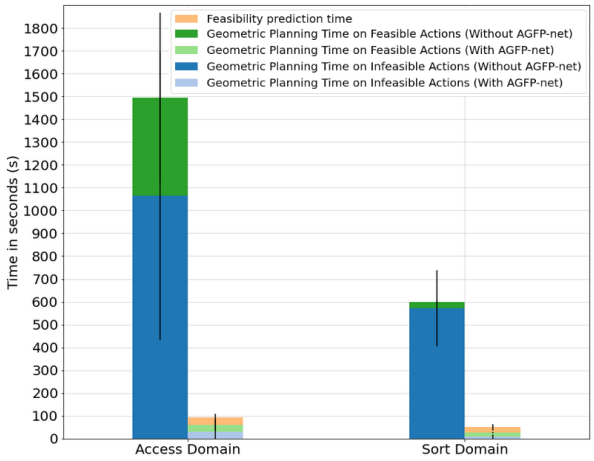


Fig. 4: Detailed planning time of the proposed approach on the *Sort* and *Access* problems with/without **AGFP-Net**.

VIII. Conclusion

In this work, we propose a feasibility-informed task and motion planner which improves total planning times on three fronts. First, the branching factor of the task planner is reduced allowing better handling of the combinatorial complexity of the search. Second, we take advantage of feasibility predictions to generate geometrically feasible task plans. Finally, we leverage grasp feasibility predictions to accelerate geometric planning time on feasible actions. We also demonstrate the performance gain yielded by our method on 5 different TAMP problems.

As future work, we consider the extension of this approach to multi-robot TAMP problems such as ones involving multiple robot arms, or mobile manipulators. Also, It is also important to test and generalize the proposed method to real-life scenarios where the shapes of objects, support surfaces and obstacles might be more complex.

REFERENCES

[1] R. Alami, T. Simeon, and J.-P. Laumond, "A geometrical approach to planning manipulation tasks. the case of discrete placements and grasps," in *The fifth international symposium on Robotics research*. MIT Press, 1990, pp. 453–463.

[2] Y. Koga and J.-C. Latombe, "On multi-arm manipulation planning," in *Proceedings of the 1994 IEEE International Conference on Robotics and Automation*. IEEE, 1994, pp. 945–952.

[3] J. M. Ahuactzin, K. Gupta, and E. Mazer, "Manipulation planning for redundant robots: a practical approach," *The International Journal of Robotics Research*, vol. 17, no. 7, pp. 731–747, 1998.

[4] T. Siméon, J.-P. Laumond, J. Cortés, and A. Sahbani, "Manipulation planning with probabilistic roadmaps," *The International Journal of Robotics Research*, vol. 23, no. 7-8, pp. 729–746, 2004.

[5] K. Hauser and J.-C. Latombe, "Multi-modal motion planning in non-expansive spaces," *The International Journal of Robotics Research*, vol. 29, no. 7, pp. 897–915, 2010.

[6] K. Hauser, "Randomized belief-space replanning in partially-observable continuous spaces," in *Algorithmic Foundations of Robotics IX*. Springer, 2010, pp. 193–209.

[7] K. Hauser and V. Ng-Thow-Hing, "Randomized multi-modal motion planning for a humanoid robot manipulation task," *The International Journal of Robotics Research*, vol. 30, no. 6, pp. 678–698, 2011.

[8] J. Barry, K. Hsiao, L. P. Kaelbling, and T. Lozano-Pérez, "Manipulation with multiple action types," in *Experimental Robotics*. Springer, 2013, pp. 531–545.

[9] J. Barry, L. P. Kaelbling, and T. Lozano-Pérez, "A hierarchical approach to manipulation with diverse actions," in *2013 IEEE International Conference on Robotics and Automation*. IEEE, 2013, pp. 1799–1806.

[10] P. Englert, I. M. R. Fernández, R. K. Ramachandran, and G. S. Sukhatme, "Sampling-based motion planning on sequenced manifolds," *arXiv preprint arXiv:2006.02027*, 2020.

[11] F. Lamiraux and J. Mirabel, "Prehensile manipulation planning: Modeling, algorithms and implementation," *IEEE Transactions on Robotics*, vol. 38, no. 4, pp. 2370–2388, 2021.

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

[13] S. Srivastava, E. Fang, L. Riano, R. Chitnis, S. Russell, and P. Abbeel, "Combined task and motion planning through an extensible planner-independent interface layer," in *2014 IEEE international conference on robotics and automation (ICRA)*. IEEE, 2014, pp. 639–646.

[14] F. Lagriffoul, D. Dimitrov, J. Bidot, A. Saffiotti, and L. Karlsson, "Efficiently combining task and motion planning using geometric constraints," *The International Journal of Robotics Research*, vol. 33, no. 14, pp. 1726–1747, 2014.

[15] L. De Silva, M. Gharbi, A. K. Pandey, and R. Alami, "A new approach to combined symbolic-geometric backtracking in the context of human-robot interaction," in *2014 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2014, pp. 3757–3763.

[16] M. Gharbi, R. Lallement, and R. Alami, "Combining symbolic and geometric planning to synthesize human-aware plans: toward more efficient combined search," in *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2015, pp. 6360–6365.

[17] J. Bidot, L. Karlsson, F. Lagriffoul, and A. Saffiotti, "Geometric backtracking for combined task and motion planning in robotic systems," *Artificial Intelligence*, vol. 247, pp. 229–265, 2017.

[18] C. R. Garrett, T. Lozano-Perez, and L. P. Kaelbling, "Ffrob: Leveraging symbolic planning for efficient task and motion planning," *The International Journal of Robotics Research*, vol. 37, no. 1, pp. 104–136, 2018.

[19] C. R. Garrett, T. Lozano-Pérez, and L. P. Kaelbling, "Sampling-based methods for factored task and motion planning," *The International Journal of Robotics Research*, vol. 37, no. 13-14, pp. 1796–1825, 2018.

[20] C. R. Garrett, R. Chitnis, R. Holladay, B. Kim, T. Silver, L. P. Kaelbling, and T. Lozano-Pérez, "Integrated task and motion planning," *Annual review of control, robotics, and autonomous systems*, vol. 4, pp. 265–293, 2021.

[21] J. Carpentier, R. Budhiraja, and N. Mansard, "Learning feasibility constraints for multi-contact locomotion of legged robots," in *Robotics: Science and Systems*, 2017, p. 9p.

[22] A. M. Wells, N. T. Dantam, A. Shrivastava, and L. E. Kavraki, "Learning feasibility for task and motion planning in tabletop environments," *IEEE robotics and automation letters*, vol. 4, no. 2, pp. 1255–1262, 2019.

[23] B. Kim, Z. Wang, L. P. Kaelbling, and T. Lozano-Pérez, "Learning to guide task and motion planning using score-space representation," *The International Journal of Robotics Research*, vol. 38, no. 7, pp. 793–812, 2019.

[24] D. Driess, O. Oguz, J.-S. Ha, and M. Toussaint, "Deep visual heuristics: Learning feasibility of mixed-integer programs for manipulation planning," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2020, pp. 9563–9569.

[25] D. Driess, J.-S. Ha, and M. Toussaint, "Deep visual reasoning: Learning to predict action sequences for task and motion planning from an initial scene image," *arXiv preprint arXiv:2006.05398*, 2020.

[26] D. Xu, A. Mandlekar, R. Martín-Martín, Y. Zhu, S. Savarese, and L. Fei-Fei, "Deep affordance foresight: Planning through what can be done in the future," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2021, pp. 6206–6213.

[27] L. Xu, T. Ren, G. Chalvatzaki, and J. Peters, "Accelerating integrated task and motion planning with neural feasibility checking," *arXiv preprint arXiv:2203.10568*, 2022.

[28] B. Kim, L. Shimanuki, L. P. Kaelbling, and T. Lozano-Pérez, "Representation, learning, and planning algorithms for geometric task and motion planning," *The International Journal of Robotics Research*, vol. 41, no. 2, pp. 210–231, 2022.

[29] M. J. McDonald and D. Hadfield-Menell, "Guided imitation of task and motion planning," in *Conference on Robot Learning*. PMLR, 2022, pp. 630–640.

[30] S. Ait Bouhsain, R. Alami, and T. Simeon, "Learning to predict action feasibility for task and motion planning in 3d environments," in *2023 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2023.

[31] J. Mahler, F. T. Pokorny, B. Hou, M. Roderick, M. Laskey, M. Aubry, K. Kohlhoff, T. Kröger, J. Kuffner, and K. Goldberg, "Dex-net 1.0: A cloud-based network of 3d objects for robust grasp planning using a multi-armed bandit model with correlated rewards," in *2016 IEEE international conference on robotics and automation (ICRA)*. IEEE, 2016, pp. 1957–1964.

[32] A. Mousavian, C. Eppner, and D. Fox, "6-dof graspnet: Variational grasp generation for object manipulation," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2019, pp. 2901–2910.

[33] Z. Zhao, W. Shang, H. He, and Z. Li, "Grasp prediction and evaluation of multi-fingered dexterous hands using deep learning," *Robotics and Autonomous Systems*, vol. 129, p. 103550, 2020.

[34] C. Wu, J. Chen, Q. Cao, J. Zhang, Y. Tai, L. Sun, and K. Jia, "Grasp proposal networks: An end-to-end solution for visual learning of robotic grasps," *Advances in Neural Information Processing Systems*, vol. 33, pp. 13 174–13 184, 2020.

[35] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[36] M. Görner, R. Haschke, H. Ritter, and J. Zhang, "Moveit! task constructor for task-level motion planning," in *2019 International Conference on Robotics and Automation (ICRA)*. IEEE, 2019, pp. 190–196.

[37] D. Devaurs, T. Siméon, and J. Cortés, "Enhancing the transition-based rrt to deal with complex cost spaces," in *2013 IEEE International Conference on Robotics and Automation*. IEEE, 2013, pp. 4120–4125.

[38] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga *et al.*, "Pytorch: An imperative style, high-performance deep learning library," *Advances in neural information processing systems*, vol. 32, 2019.