2023 IEEE International Conference on Robotics and Automation (ICRA 2023)
May 29 - June 2, 2023. London, UK

Learning to Predict Action Feasibility for Task and Motion Planning in 3D Environments

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

***Abstract*— In Task and motion planning (TAMP), symbolic search is combined with continuous geometric planning. A task planner finds an action sequence while a motion planner checks its feasibility and plans the corresponding sequence of motions. However, due to the high combinatorial complexity of discrete search, the number of calls to the geometric planner can be very large. Previous works [1] [2] leverage learning methods to efficiently predict the feasibility of actions, much like humans do, on tabletop scenarios. This way, the time spent on motion planning can be greatly reduced. In this work, we generalize these methods to 3D environments, thus covering the whole workspace of the robot. We propose an efficient method for 3D scene representation, along with a deep neural network capable of predicting the probability of feasibility of an action. We develop a simple TAMP algorithm that integrates the trained classifier, and demonstrate the performance gain of using our approach on multiple problem domains. On complex problems, our method can reduce the time spent on geometric planning by up to 90%.**

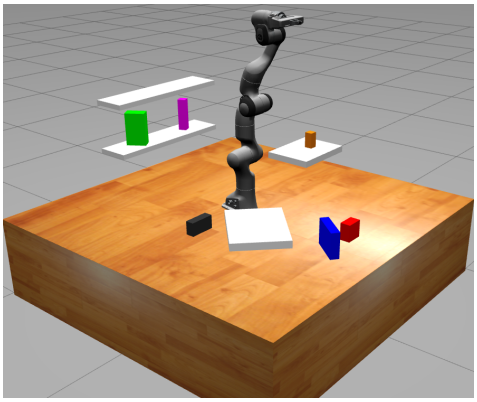


Fig. 1: An example of a 3D environment our approach is able to deal with. It includes multiple support surfaces and objects placed at different poses.

I. Introduction

Task and motion planning (TAMP) [3] [4] [5] [6] requires combining a discrete search over a space of symbolic decisions, with a continuous search over the space of associated geometric motions. A logical task planner searches for an action sequence leading to a symbolic goal state, while a motion planner ensures its geometric feasibility and plans the corresponding sequence of motions. Given the combinatorial complexity of discrete search, this results in a large number of calls to the motion planner before a geometrically feasible solution is found. In fact, motion planning is a bottleneck for combined task and motion planning [7].

In this work, we extend the approach proposed by Driess et al. [1] [2]. The idea is to predict the feasibility of discrete actions during task planning, so that the search over the symbolic space can be informed. Hence, the number of calls to the geometric planner can be greatly reduced, since the task planner will prioritize promising solutions first. This can be done by training a classifier to predict the probability of feasibility given the state of the environment and the considered action.

The method proposed in [1] [2] is however limited to tabletop environments and problems in which only one object has a goal pose. Our goal is to generalize this approach to 3D environments such as the one shown in Figure 1, with different numbers and poses of support surfaces and object, thus covering the whole workspace of the robot. Also, we aim at solving a wider range of problems, in which multiple objects can have a goal pose. In order to achieve this, some challenges have to be tackled. First, existing learning models require inputs with a fixed dimensionality. Therefore, there is a need for fixed-size representations of the 3D environment, the object of interest and the considered action. Another challenge is the integration of the trained classifier to a TAMP algorithm. The latter must take advantage of feasibility prediction in order to accelerate the search for a geometrically feasible solution, while ensuring its completeness.

In order to tackle these challenges, we propose a new 3D scene representation consisting of a multi-channel image, each channel corresponding to a specific view of the scene. We use depth images rather than regular images in order to incorporate the 3 dimensions in each view and generalize our approach to real world scenarios. Objects of interest are encoded as multi-channel masks in the same way as the 3D environment (Section IV). We also propose an algorithm that makes use of the predictions to accelerate task and motion planning, and can solve a wide range of problems (Section V).

In summary, the contributions of this paper are:

- We generalize the work proposed in [1] [2] to 3D environments, introducing a new approach for efficient 3D scene representation.
- We propose a TAMP algorithm which takes advantage of the classifier to accelerate the planning process, and is able to solve a wider range of complex problems involving the displacement of multiple objects to their goal pose.

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

II. Related Work

A. Manipulation Planning

Early works in TAMP viewed the problem from a geometrical perspective [8] [9] [10] [11]. The main idea is to search for a solution in a manipulation graph, which maps the intersection between the sets of valid grasps and valid placements, as well as motions connecting them. These methods suffer from a combinatorial explosion if the number of objects in the scene increases. [12] extends this approach to include non prehensile actions such as pushing or pulling. Works such as [13] [14] [15] [16] generalize this class of methods as multi-modal motion planning.

Some recent methods [17] [18] [19] focus on representing actions using constraints that define manifolds. The goal is to find a sequence of manifolds in a constraint graph that brings the environment from an initial to a goal state. However, if the number of objects increases, the constraint graph can be very complex to build. Also, a known issue with these approaches is the crossed foliation issue [20].

B. Classical TAMP

Most of existing TAMP methods combine symbolic planning with geometric planning [4] [5] [21] [6] [22] [23]. A logical task planner is used to explore symbolic states of the environment and possible actions, while a motion planner checks their feasibility and plans the corresponding motions. If an action is infeasible, then geometric backtracking is performed. As the number of movable objects increases, the combinatorial complexity of the symbolic search increases exponentially. This results in a very high number of calls to the geometric planner, most of these calls being for infeasible actions. Works such as [24] [25] propose an informed backtracking strategy using feedback from the motion planner, e.g. which obstacle causes a collision or which kinematic constraints are violated. This method allows to sort possible actions by relevance to the task. Although this accelerates the planning process, motion planning problems still have to be solved to get information on the relevance of actions.

C. Learning for TAMP

Recent works leverage learning methods to accelerate TAMP algorithms [7] [1] [2] [26] [27] [28] [29] [30] [31]. The goal of these methods is to provide a learned heuristic to the task planner in order to prioritize promising actions, and hence reduce the number of calls to the geometric planner. Kim et al. [27] propose a method that uses experience from previous plannings, using a score-space representation. Xu et al. [28] propose a reinforcement learning model which predicts action affordances. These are then used to select the most promising action sequences. However, these approaches are domain-specific, meaning that each time the problem domain changes, the learning model has to be retrained.

Works such as [7] [1] [2] [26] propose generalizable learning approaches which can be applicable to multiple domains. Wells et al. [7] use SVMs to predict the geometrical feasibility of actions in tabletop environments containing two movable objects. Used as a heuristic, this prediction reduces considerably the number of calls to the motion planner. However, several SVMs have to be trained, each one specialized in a pair (action, grasp). Also, the environment is represented using manually designed feature vectors. Thus, applying this approach to scenes with more than two objects requires calling the SVMs on each pair of objects.

Driess et al. [1] [2] solve this issue by representing the environment using a top-view depth image, which can encode multiple objects at once. They also propose a neural network which provides a probability of feasibility. The actions can then be sorted using this probability for exploration. However, this method is limited to tabletop environments. Also, it aims at solving problems involving the displacement of one object only to a target location. Xu et al. [26] made a first contribution towards a generalization to 3D environments, by providing the relative position of the support surface to the robot's base as input to the classifier. However, only a small area from the scene in represented. It is also not applicable to environments containing multiple support surfaces, especially when they act as obstacles. Our approach generalizes the work of Driess et al. [1] [2] to cover the whole workspace of the robot and hence predict action feasibility in 3D environments. These can include support surfaces such as shelves or cupboards, as well as objects that are placed on higher surfaces than the table. Moreover, we aim at covering more complex problems, in which the goal is to bring multiple objects to their target placements.

III. Problem Description

In this work, we focus on manipulation problems involving sequences of $Pick$ and $Place$ actions as solutions. The manipulation environment E is composed of n_O movable objects and n_S stable support surfaces. Let S be the state of the environment and the robot, representing the configuration of all movable objects in the planning scene as well as the state of the robot $\{Free, Holding\}$. Let an action be $a_{(O,G,\mathbf{q}_O)}$, where $a \in \{Pick, Place\}$, O is the considered object, G is a discrete grasp by which the object is (or was) picked, and $\mathbf{q}_O$ is the current pose of O in case of a $Pick$ action, or the pose at which O should be placed in case of $Place$ action. For an easier read, we simplify the notation $a_{(O,G,\mathbf{q}_O)}$ to a.

Furthermore, let $\pi(S, a)$ be the motion plan associated with applying action a at state S, and $F_\pi(\pi(S, a), E)$ a function that checks the feasibility of a motion plan such that:

$$F_\pi(\pi(S,a),E) = \begin{cases} 0, & \text{if } \pi(S,a) \text{ is infeasible} \\ 1, & \text{if } \pi(S,a) \text{ is feasible} \end{cases} \quad (1)$$

Here, infeasibility can be due to either a collision with another object, a collision with a support surface, the kinematic constraints of the robot or the lack of a feasible motion.

The goal of TAMP is to find a geometrically feasible task plan $\tau = \{a_0, a_1, ..., a_K\}$ and the corresponding sequence of motions $\Pi = \{\pi_0, \pi_1, ..., \pi_K\}$, which brings the environment from an initial state S_0 to a goal state S_g. Since symbolic task planners do not have any built-in motion planning or

geometric reasoning capabilities, they have no way of finding Π or verifying (1). Hence, there is a need for a geometric planner as well. However, the combinatorial complexity of discrete search results in a high number of calls to the latter. Moreover, geometric planning can be computationally expensive depending on the feasibility of an action as well as its difficulty. Previous works [7] [26] have shown that in TAMP, most of planning time is spent on geometric planning, mostly checking infeasible actions.

IV. Action Feasibility Prediction

In order to overcome these challenges, as in [1] [2], we seek to provide a heuristic to the task planner, allowing it to get fast feedback on the feasibility of an action, and thus call the motion planner on promising action sequences first. To accomplish this, we propose to train a classifier f to predict the feasibility given an action a and a specific state of the environment S. However, as mentioned previously, we first need a fixed-size and generalizable representation of the state of the environment.

A. 3D Scene Representation

We extend the method proposed by Driess et al. [1] [2] for tabletop scene representation to 3D environments. Their idea is to use fixed-size, top view depth images that can represent scenes with different numbers of objects and a single support surface. Although this representation is efficient for tabletop environments, it has a major issue in the case of 3D environments. Indeed, contrarily to the tabletop case, 3D scenes can have support surfaces at different heights, which might create occlusions. For example, if a scene contains a table, a shelf, and one movable object placed underneath the shelf, the object will not be visible in a top view depth image of the scene. As a result this representation does not encode any information about the shape, size and pose of the object.

In order to solve this issue, we propose to represent the 3D scene as 5-channel depth images, with each channel corresponding to a specific view of the scene (top, front, rear, left, right) as shown in Figure 3. This representation reduces greatly the effect of occlusions, because unless the object is hidden from all sides of the scene, it will be visible in at least one view, and information about that object is encoded. Using this approach, the environment E and the state S which dimensionality is clearly dependent on n_O and n_S, becomes $I_{scene} \in \mathbb{R}^{5\times w\times h}$ which has a fixed dimensionality, w and h being the dimensions of the images. I_{scene} represents not only information about movable objects, but also the support surfaces in the scene[1].

B. Action Representation

Following the approach proposed in [2], we decompose actions into two parts. The first is $\hat{a}(G)$ representing a $Pick$ or $Place$ with a specific grasp G. The second is the pair $(O, \mathbf{q}_O)$ corresponding to the considered object and its pose in the environment. This way, we can have a representation

[1]Note that the depth images do not come from depth cameras, but are rather internally constructed from the scene's model.

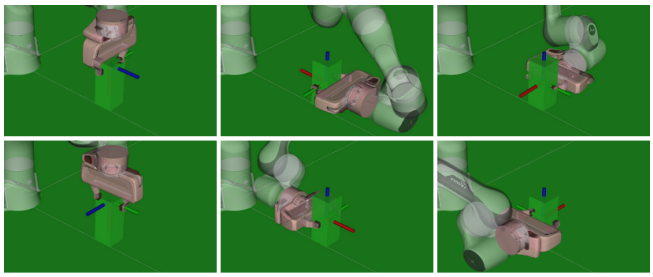


Fig. 2: Visualization of the 6 discrete grasps considered.

for the object of interest, and one for the action in question separately.

For the pair (O, q_O), we complement the 3D scene representation by adding a 5-channel mask $I_{object} \in \mathbb{R}^{5\times w\times h}$ showing only the object of interest at the pose q_O. This incorporates all information related to the object that is relevant to the classifier, i.e. its shape, size, and pose. Combining I_{scene} and I_{object}, we obtain a single 10-channel image $I(S, E, O) \in \mathbb{R}^{10\times w\times h}$. This representation is highly compact compared to existing 3D environment representations such as point clouds, voxel-based or cell decomposition-based representations.

Regarding the action $\hat{a}(G)$, we consider 6 discrete types of grasps as shown Figure 2, each one corresponding to a specific side of the object in the perspective of the robot (e.g. approach from the right or from the rear). These discrete grasps are more general than the ones proposed in [1] [2]. It is important to note that these are types of grasps and not grasp poses, meaning that it specifies the side from which the object has to be grasped, but the height and depth of the grasp are free parameters chosen by the geometric planner. As a result, we have 12 discrete actions corresponding to $Pick$ or $Place$ tasks with one of the 6 discrete grasps. In general, it is not advisable to use a single categorical variable as input to learning models. For that reason, we use one hot encoding to generate a 12-dimension vector $\hat{\mathbf{a}}_{ohe} \in \mathbb{R}^{12}$, where each dimension corresponds to a discrete action $\hat{a}(G)$.

C. Neural Network Architecture

The complete proposed architecture is detailed in Figure 3. Given the scene-object and action representations, our proposed model is a classifier $f(I(S, E, O), \hat{\mathbf{a}}_{ohe})$ which approximates $F_\pi(\pi(S, a), E)$. In our case, the latter is the output of a sampling-based motion planner. As in, [1], our Action Feasibility Prediction Neural Network (**AFP-Net**) has an encoder-decoder architecture. Since we are working with images, the most natural choice for a scene-object encoder is a convolutional neural network (CNN). In our model, we use the ResNet architecture [32] which has a good performance on image classification and object detection tasks, and is robust to neural network training issues such as the vanishing gradient problem. We modify the first layer of the CNN in order for it to accept 10-channel image inputs rather than classic 3-channel RGB images. Also we add a flattening layer

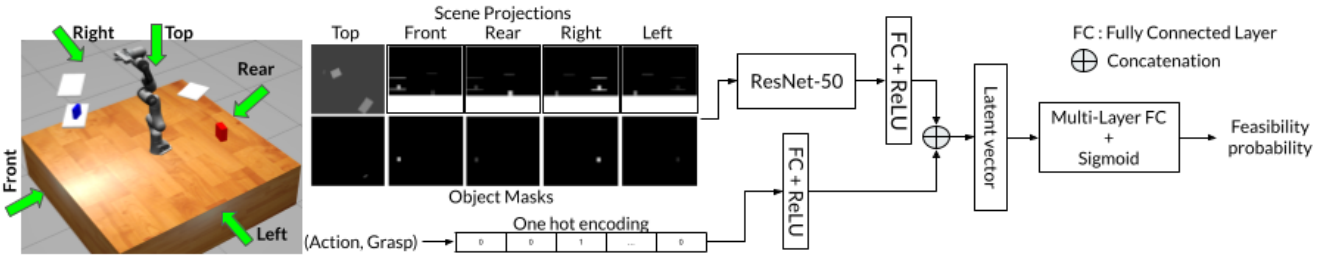


Fig. 3: Detailed architecture of the proposed neural network **AFP-Net**.

and a fully connected layer with a ReLU activation function at the end of the CNN.

The action vector $\hat{\mathbf{a}}_{ohe}$ is encoded using a fully connected layer with a ReLU activation function as well. Each encoder outputs a latent vector, these are then concatenated into a single vector containing encoded information from the 3D scene, the considered object, the action as well as the type of grasp to be checked for feasibility. This vector is then fed to a multi-layer perceptron followed by a Sigmoid activation function. The output of our model is a probability of feasibility p of an action given the state of the environment and the robot.

V. Task and Motion Planning

In order to test the efficiency of our learned heuristic, we developed a simple TAMP algorithm which integrates the trained neural network in order to accelerate the search for a geometrically feasible solution. The idea is to reduce the number of calls to the sampling-based motion planner by testing promising action sequences first. The main algorithm (Algorithm 1) is a best-first tree search with a backtracking capability. A set Q of nodes to expand is maintained. These nodes are equivalent to the states S defined in Section III. They represent the configuration of all objects in the scene, the state of the robot.

At each iteration of the main loop, the node S with the minimum cost and maximum probability of feasibility p is popped from the set Q. If S is a goal state, we retrieve the complete action sequence τ. We then call the motion planner to verify its geometric feasibility and find its corresponding sequence of motions Π. If it is feasible, then a geometrically feasible solution was found and the planning is finished. Otherwise, we perform backtracking by pruning all children of the first infeasible node in the task plan from the tree.

If S is not a goal node, then it is expanded by calling $findChildren()$ described in Algorithm 2. Depending on the state of the robot (i.e free or holding an object), we generate all possible $Pick$ or $Place$ actions, and find the resulting children nodes. We then compute the cost of each child, which is a weighted sum of the number of actions leading to S and the number of objects that have not reached their goal placement yet. After that, we use the trained classifier to get the probability of feasibility p of each child. Finally, all children are added to the set Q.

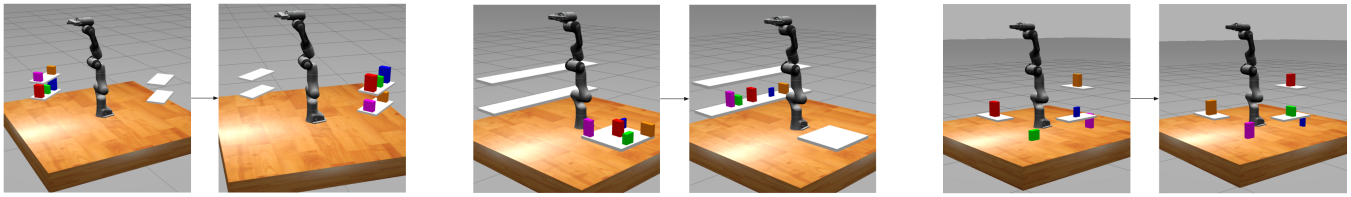
With this approach, the trained classifier is used to sort the set Q. This way, the nodes with the highest probability of feasibility are expanded first. In case of misclassifications, false positives (infeasible actions given a high probability of feasibility) may result in an infeasible task plan that would be tested and dismissed by the geometric planner. If the neural network always classifies actions as feasible, the algorithm becomes a classical TAMP method without any geometric intuition. In case of false negatives (feasible actions given a low probability of feasibility), other actions predicted as feasible will be expanded first, which might hurt the performance of the algorithm. Note that Algorithm 1 is complete, since using the classifier as a heuristic does not discard any state from the search tree. Hence, in worst case

Algorithm 1 Task and motion planner

Input: E, S_0, S_{goal} ▷ Initial and goal states
$Q \leftarrow \{S_0\}$ ▷ Set of nodes to expand
while Solution not found and Q not empty **do**
  $S \leftarrow argmax_p(argmin_{cost} \quad Q)$
  if $S = S_{goal}$ **then**
    $\tau \leftarrow retrieveTaskPlan(S)$
    $\Pi \leftarrow constructMotionSequence(\tau, E)$
    if Π is feasible **then**
      solution found
      break
    else
      $S_{infeasible} \leftarrow getFirstInfeasibleNode(\Pi)$
      $Q \leftarrow Q \setminus pruneChildren(S_{infeasible})$
      continue
    end if
  end if
  $Q \leftarrow Q \cup findChildren(S, E)$
end while

Algorithm 2 findChildren

Input: S, E
$children \leftarrow \emptyset$
for all a applicable at S **do**
  $child \leftarrow nextState(S, a)$
  $child.cost \leftarrow computeCost(child)$
  $child.p \leftarrow predictFeasibility(child)$
  $children \leftarrow children \cup child$
end for
return $children$



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

Fig. 4: Visualization of the initial states and goal states for the 5-object version of the three problem domains used to measure the performance of our approach.

scenario, all states and actions will be explored, and if there is a feasible solution, it will be found.

VI. Experiments

We focus on problems with one robot arm with a parallel gripper (Franka Emika Panda), and scenes involving one table, a variable number of support surfaces (shelves), and multiple box-shaped objects, as shown in Figure 1.

A. Data Generation and Annotation

We build our dataset on the assumption that the neural network is able to generalize to scenes with multiple objects, even though it is trained on scenes containing two objects only as in [1] [2]. We generate 3000 scenes for training. Each scene has a large table on which the robot is placed at the center. Also, a number of up to 4 support surfaces are generated, which sizes and poses are sampled randomly. Two box-shaped objects with sizes sampled randomly are then generated. In order to incorporate the different challenges of $Pick$ and $Place$ actions, we define 3 placement types. The first is random placement, in which the object is placed at a random pose on a random support surface. The second is a proximity placement, in which the object is placed near another object. The last placement type is underneath a support surface. As a final step, we perform a collision checking ensuring that the generated scenes are valid.

After scenes generation, we proceed to the construction of the dataset. For each object in the scene, we generate 6 $Pick$ actions, one per discrete grasp type. We also generate 6 $Place$ actions, two per placement type defined previously. Since we are not interested in trivial infeasibility, we disregard grasps from object sides larger than the gripper's width. This will result in at most 12 datapoints per object. Using the same method and different random seeds, we construct a validation set and a test set of 1000 scenes each. We use an adapted version of Moveit! Task Constructor [33] with a BiTRRT motion planner [34] for annotation. Data generation and annotation takes nearly 15 hours.

B. Neural Network Training/Testing

We implemented and trained our neural network using the Pytorch library [35]. The model is trained using a weighted binary cross entropy loss to account for class imbalance. We use the ADAM optimizer with a learning rate of 0.0005 and a batch size of 128. To prevent overfitting, we use dropout with a probability of 0.2, and weight decay of 0.0001.

We test the performance of our classifier on the test set. As our dataset is imbalanced, we use the F1 score, the ROC-AUC, the true positive rate (TPR) and the true negative rate (TNR) as metrics to measure the performance of our model. These metrics are known to be robust to class imbalance.

C. Test Environments for the TAMP Approach

In order to measure the gain of using the classifier as a heuristic for the TAMP algorithm, we define three problem domains presenting different challenges to classical TAMP methods. The first is the $Reorder$ domain (Figure 4a), where a number of objects are initially placed on a two-shelves cupboard, and have to be placed on another cupboard in different order. This problem allows us to test our algorithm when faced with cases where the grasp choice is important, since picking or placing objects underneath a shelf from the top is infeasible.

The second domain is the $Unpack$ domain (Figure 4b), in which objects are initially on a tray-like surface and have to be unpacked and ordered into a cupboard. The challenge here is that initially, the objects are close to one another, so the choice of grasp is very important in order to achieve a feasible $Pick$ action. This difficulty is increased by the fact that the goal placement of the objects is underneath a shelf, meaning that top grasps are not allowed.

The last problem is the $Swap$ domain (Figure 4c), in which a set of objects are placed either on the table or on higher support surfaces, and the goal is to swap their poses. Here, the difficulty comes from the fact that the goal poses of objects are already occupied, hence intermediary placements are necessary.

VII. Results

We analyse first the performance of **AFP-Net** on the validation and test sets. We then demonstrate the performance gain of using the neural network as a heuristic on the test problem domains.

A. Neural Network Performance

Our network achieves an F1-score of approximately 90% on both the validation and test sets. It also has an AUC of 96%. For comparison, the AUC of a random classifier is 50%. This shows that the model is able to predict accurately the feasibility of actions in 3D environments even though it is trained on imbalanced data. It also shows that our proposed

TABLE I: Comparison of the performance of the TAMP algorithm with and without using **AFP-Net** on the 5-object version of the three problem domains. Results are averaged over 10 runs.

Domain	Heuristic	Nb Infeasible Task Plans	Nb Motion Planning calls	Nb Infeasible Motion Plannings	Nb Expanded Nodes
Reorder	None	5.0	19.8	9.8	46.4
	AFP-Net	**0.4**	**10.5**	**0.5**	**10.4**
Unpack	None	11.8	29.5	19.5	85.2
	AFP-Net	**3.3**	**17.1**	**6.3**	**33.3**
Swap	None	30.6	67.7	53.7	200.4
	AFP-Net	**3.9**	**22.0**	**7.8**	**85.3**

approach for 3D scene representation is capable of encoding enough information about the objects and support surfaces in the environment. Moreover, Using a probability threshold of 0.5, **AFP-Net** has a true feasible rate of 91.5% and a true infeasible rate of 84.3% on the test set. This is a comparable performance to the one obtained by Driess et al. [1] even though our test data is not limited to tabletop environments.

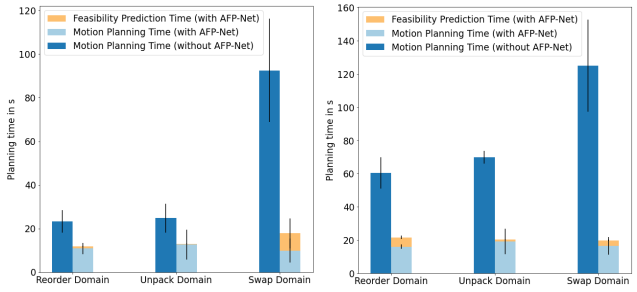
B. Performance on Test Domains

We ran our TAMP algorithm on the three experiment scenarios for 10 runs each. In order to test the generalizability of our method to scenes with more than 2 objects, we ran the algorithm on two versions of each scenario: one containing 2 objects only, and the other containing 5 objects. We compare our approach to one which does not use any heuristic on geometrical feasibility of actions.

Figure 5 shows the improvement in planning time when **AFP-Net** is used as a heuristic for the TAMP algorithm. Using feasibility prediction reduces considerably the time spent on geometric planning. Indeed, for 2-object problems, our approach reduces motion planning time by 50% on the $Reorder$ and $Unpack$ domains and 90% on the $Swap$ domain. On the other hand, for 5-object problems, motion planning time is 74% lower using **AFP-Net** as a heuristic for the $Reorder$ and $Unpack$ domains, and 87% lower for the $Swap$ domain. These results show that the higher the complexity of the problem is, the higher the performance gain from using our approach will be. Note that unlike a non-informed TAMP algorithm, the planning time using our proposed method remains quasi-constant across problems with different levels of difficulty. Moreover, the added computational cost of feasibility prediction does not hurt the total planning time. The average feasibility prediction time is $50ms$, including the construction of scene projections and querying the neural network.

One interesting aspect of our approach is that once the scene-object pair is encoded by the CNN, the obtained latent vector can be stored. Hence, next time an action is tested in the same scene configuration, only the decoder part of the network has to be queried, thus saving projection generation and neural network execution time. Using this method, the average feasibility prediction time is reduced to $10ms$. The reported results were obtained using this optimization method.

Table I shows the efficiency of our approach in terms of reduction of the number of calls to the geometric planner. For the $Reorder$ domain, the first generated task plan using our approach is a feasible one in most of the runs. Hence, the number of infeasible calls to the geometric planner is 0 in most runs. For the $Unpack$ domain, even though there has been some misclassifications resulting in 3 infeasible task plans, the performance is still better than when no heuristic is used. The largest gain can however be observed on the $Swap$ problem, where the number of infeasible solutions found using **AFP-Net** is 3.9, compared to 30.6 without the use of a heuristic. These results show that our approach can reduce by up to 9 times the number of infeasible calls to the motion planner, while being robust to classification errors.



(a) 2-object problems (b) 5-object problems

Fig. 5: Detailed planning time of our algorithm on the three defined problem domains with and without using **AFP-Net**, averaged over 10 runs.

VIII. Conclusion

In this work, we demonstrated an approach for generalizing action feasibility prediction to 3D environments. In addition to being computationally efficient, a 3D scene representation using 5 views of the scene as depth images is capable of encoding enough information about the size, shape and pose of object as well as support surfaces in the environment. Using our proposed neural network as a heuristic in a TAMP algorithm decreases the number of infeasible task plans generated, thus the number of calls to the motion planner and the total planning time can be greatly reduced.

As a future work, the neural network can be trained and tested on more realistic environments, including obstacles and objects with different shapes. It would also be interesting to extend our method to non-prehensile actions such as pushing or pulling, as well as multi-robot TAMP problems.

REFERENCES

[1] 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.

[2] 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.

[3] 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.

[4] 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.

[5] 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.

[6] 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.

[7] 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.

[8] 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.

[9] 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.

[10] 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.

[11] 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.

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

[13] 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.

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

[15] 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.

[16] 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.

[17] J. Mirabel, S. Tonneau, P. Fernbach, A.-K. Seppälä, M. Campana, N. Mansard, and F. Lamiraux, "Hpp: A new software for constrained motion planning," in *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2016, pp. 383–389.

[18] F. Lamiraux and J. Mirabel, "Prehensile manipulation planning: Modeling, algorithms and implementation," *IEEE Transactions on Robotics*, 2021.

[19] 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.

[20] J. Mirabel and F. Lamiraux, "Manipulation planning: addressing the crossed foliation issue," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2017, pp. 4032–4037.

[21] 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.

[22] 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.

[23] 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.

[24] 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.

[25] 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.

[26] 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.

[27] 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.

[28] 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.

[29] 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.

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

[31] 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.

[32] 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.

[33] 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.

[34] 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.

[35] 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.