

Spatial Reasoning via Deep Vision Models for Robotic Sequential Manipulation

Hongyou Zhou¹

Ingmar Schubert¹

Marc Toussaint^{1,2}

Ozgur S. Oguz³

Abstract—In this paper, we propose using deep neural architectures (i.e., vision transformers and ResNet) as heuristics for sequential decision-making in robotic manipulation problems. This formulation enables predicting the subset of objects that are relevant for completing a task. Such problems are often addressed by task and motion planning (TAMP) formulations combining symbolic reasoning and continuous motion planning. In essence, the action-object relationships are resolved for discrete, symbolic decisions that are used to solve manipulation motions (e.g., via nonlinear trajectory optimization). However, solving long-horizon tasks requires consideration of all possible action-object combinations which limits the scalability of TAMP approaches. To overcome this combinatorial complexity, we introduce a visual perception module integrated with a TAMP-solver. Given a task and an initial image of the scene, the learned model outputs the relevancy of objects to accomplish the task. By incorporating the predictions of the model into a TAMP formulation as a heuristic, the size of the search space is significantly reduced. Results show that our framework finds feasible solutions more efficiently when compared to a state-of-the-art TAMP solver.

I. INTRODUCTION

Task and Motion Planning (TAMP) usually combines a discrete search on a symbolic, logical level with a geometric planner. The discrete search is employed with finding the high-level symbolic action sequence, which in turn informs the geometric planner to solve for a feasible motion path that satisfies the goal state. Due to the huge computational overhead of solving geometric problems and the combinatorial complexity of discrete decisions, solving a TAMP problem often requires a large amount of computation [1, 2, 3, 4].

The process of task planning often depends on the number of objects that are available in the environment and the operations that the robot can perform. The algorithm builds the corresponding search tree based on the combination of objects and robot action set and expands the tree step by step (with action-object(s) tuples) until it reaches the goal state. Exponential growth rate of the search tree tremendously diminishes the efficiency of the TAMP methods. In reality, there are often multiple objects in the same environment, and a large number of operational possibilities by the autonomous agent, which impede the usability of TAMP solvers in such complex scenarios. Fortunately though, the task can usually be achieved by only considering a small subset of these objects.

Machine learning methods using visual data offer efficient ways to solve robotic manipulation problems [5, 6]. However,

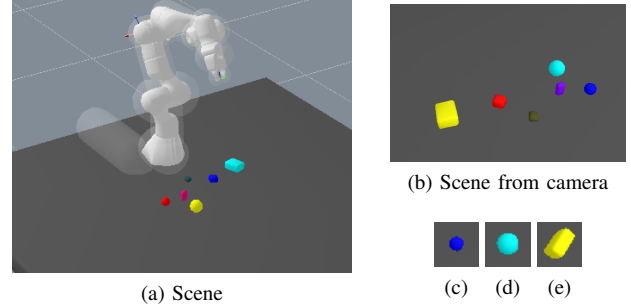


Fig. 1: Test scenario with six objects: (a) shows the overview of this test scenario, (b) is the view from the camera located on the robot end-effector, (c, d and e) are the canonical views of objects, the first two are related to the goal predicate, and the third is the object whose relevance is queried. Take the predicate (*on-left-side*) as an example, using these canonical views as inputs would query the model whether object in (e) should be considered when the goal is to *put object in (c) on the left-side of object in (d)*.

one challenge in integrating machine learning into TAMP is how to set up the problem (for example) by encoding objects in the scene as fixed-length inputs and then feeding them to the learning algorithm [7]. In general, learning methods are often integrated into robotic problems for a single task only, where fixed-size feature representation is often sufficient. On the other hand, TAMP formulations are designed to tackle a broad set of tasks in different environments with varying numbers of objects and goals. This hinders the usage of such fixed sized features for solving sequential robotic manipulation problems.

In this study, we argue that task-relevant objects can be inferred directly from the sensor space of the robot, instead of in a feature space. The sensor space not only has a fixed dimensionality, and hence is suitable for standard learning algorithms, but the sensory information also contains details about what the robot can extract from the current world state, except the history or other priors. Therefore, this paper aims to answer the following question: Is it possible to learn the relevance of an object in the scene given the goal state? This is critical because, as mentioned previously, the complexity of the TAMP algorithm belongs to the NP-hard class, and therefore the efficiency of the algorithm can be greatly improved if its search space can be reduced by learning methods.

We propose a deep neural network that takes a scene image, canonical views of objects included in the goal predicate, and the canonical view of another object in the scene to predict the relevance of that object for achieving the task (Fig. 1). The relevance of objects can then be used to improve the time complexity of mixed-integer programs, which is one way to realize TAMP, where the discrete action

¹Technical University of Berlin

²Science of Intelligence Cluster of Excellence, TUB

³Computer Eng. Dept., Bilkent University

(integer) variable represents the abstract decision and the resulting nonlinear trajectory optimization problem represents the geometric part. In the experiments, we consider the problem of grasping the target object with one robot arm and placing it on the position that is indicated by the goal predicate, as shown in Fig. 1. An object can be considered as irrelevant due to the non-interference of the trajectory of reaching the goal state. In contrast, if an object blocks the trajectory (i.e., there might be a collision), then the symbolic planner has to offer a different solution that instructs the robot to move this object out of the current place, which in turn makes this object relevant.

One of the experiment is shown in Fig. 1. As an example, if our goal is to place object c to the left of object d , then the symbolic planner, without prior knowledge, will traverse all possibilities and eventually find a path that satisfies the “object c is to the left of object d ” state, or if there is no possible solution, then it returns non-reachable. However, as humans, we can quickly evaluate that no other object needs to be moved except for these two objects, which is what we want to achieve with our trained classifier, thus greatly limiting the expansion of the search space. In general, completing the goal requires actions on very few objects, and the majority of the objects that appear in the scene are irrelevant to the goal state. Hence, we train several neural network architectures, including vision transformers (ViT) and residual neural networks (ResNet), to predict the relevancy of objects. The learned classifier then guides the discrete search. The main contributions are as follows:

- We propose using vision-based deep neural networks to predict the relevance of the objects from visual input for sequential manipulation planning.
- We propose a flexible and scalable *predicate extractor* to represent the spatial relationship.
- We develop the use of the network to guide the discrete search as a heuristic.

We support those contributions by showing that the trained models reach high accuracy, and our approach significantly reduces the number of motion planning problems in general, and thus improving computational efficiency of a state-of-the-art TAMP solver. By evaluating multiple deep neural network models on the same dataset, we did not observe significant performance differences, e.g., using a ViT does not appear to be a better choice [8].

II. RELATED WORK

A. Task and Motion Planning

A TAMP problem is typically specified in two parts: the first part is a symbolic discrete decision, and the second part is a continuous motion planner. The symbolic discrete decision proposes a discrete node to reach the goal state, while the continuous motion planner is intended to solve the feasibility problem between discrete states. There are three mainstream solutions for a motion planner: sampling based [9, 10, 11, 12, 13], discretization based on the configuration/action space [14, 15, 16, 17], and the optimization

based methods [18, 19]. The Logic Geometric Programming (LGP) [20] also falls into optimization-based category, where the logic imposes a skeleton of active constraints on a nonlinear program [21, 22]. Formalizing this hybrid nature of the contact and interaction modes in robotics results in a mixed-integer program [23, 24], and LGP is formulated as such for TAMP problems [2]. A major problem of LGP, though, is the combinatorial complexity that is introduced by the discrete variables (i.e., due to the number of actions or objects in the scene). In this study, we propose to address this complexity by reducing the search space using a vision-based deep learning model that predicts a small set of objects to be interacted with to realize a task.

B. Learning for Task and Motion Planning

Advances in deep networks have made it possible to build a planning system that is based on sensor inputs. For example, attention mechanisms offer unique opportunities to learn representations of the environment [25, 26, 27, 28]. Such models can be employed to reason about the spatial relationship between objects, thus the symbolic planner can leverage this information for high level planning [8, 29]. Our model is based on vision transformer and residual neural network [25, 30], and uses supervised learning to give the model the ability to infer task-relevant objects based on an image in the scene. The aim is to use visual inference to guide the symbolic planner to consider the option that is more likely to succeed in a given situation. The viability of image-based deep networks for direct service to planning and control systems has also been confirmed by many studies [5, 7, 6].

In spirit, the motivation with SORNET [8] is most similar to ours. The input to SORNET comprises image patches and canonical views of the objects to be queried. The output is the spatial relationship between the two objects queried in the scene. However, this information is often not directly used in task and motion planning, because knowing the spatial relationship between objects does not help much in motion planning. SORNET uses the embedding output of the vision transformer (ViT) to train additional networks that models the predicates. We, instead, use ViT to directly train a classifier that outputs the probability of relevancy of objects for a given goal. This allows us to directly integrate the model output into our TAMP formulation.

In general, the method provided in this study inferred the task-related objects on the basis of images, which can limit the space of the discrete search part of task and motion planning, thus improving the overall search efficiency.

III. MIXED-INTEGER PROGRAM FOR ROBOT TRAJECTORY PLANNING

In this section, we explain our task and motion planning formulation. Use cases for this formulation is two-fold: (i) we solve a large number of complex rearrangement problems for obtaining labels of task-relevancy of objects, (ii) we integrate our learned classifiers for improving the computational efficiency of solving such sequential manipulation problems.

Let $\mathcal{X} \subset \mathbb{R}^{n(S)} \times SE(3)^{m(s,S)} \times \mathbb{R}^{6 \cdot n_{cp}(s,S)}$ be the configuration space as a function of the scene S and the symbolic state variable $s \in \mathcal{S}(S)$. This configuration space contains the $n(S)$ -dimensional generalized coordinate of robot joints, the poses of $m(s,S)$ multiple rigid objects and six dimensional wrench contact interactions for each of the $n_{cp}(s,S)$ contact pairs. The idea is to find a globally consistent path $x : [0, KT] \rightarrow \mathcal{X}(s_{k(t)}, S)$ in this configuration space that minimizes

$$P(g, S) = \min_{\substack{K \in \mathbb{N} \\ x: [0, KT] \rightarrow \mathcal{X} \\ a_{1:K}, s_{1:K}}} \int_0^{KT} c(x(t), \dot{x}(t), \ddot{x}(t), s_{k(t)}, S) dt \quad (1a)$$

s.t.

$$\forall t \in [0, KT] : h_{eq}(x(t), \dot{x}(t), s_{k(t)}, S) = 0 \quad (1b)$$

$$\forall t \in [0, KT] : h_{ineq}(x(t), \dot{x}(t), s_{k(t)}, S) \leq 0 \quad (1c)$$

$$\forall k=1, \dots, K : h_{sw}(x(kT), \dot{x}(kT), a_k, S) = 0 \quad (1d)$$

$$\forall k=1, \dots, K : a_k \in \mathbb{A}(s_{k-1}, S) \quad (1e)$$

$$\forall k=1, \dots, K : s_k = succ(s_{k-1}, a_k) \quad (1f)$$

$$x(0) = \tilde{x}_0(S) \quad (1g)$$

$$s_0 = \tilde{s}_0(S) \quad (1h)$$

$$s_K \in \mathcal{S}_{goal}(g). \quad (1i)$$

The path x is assumed to be globally continuous ($x \in C([0, KT])$) and consists of $K \in \mathbb{N}$ phases (the number is part of the decision problem itself), each of fixed duration $T > 0$, in which we require smoothness $x \in C^2([(k-1)T, kT])$. These phases are also referred to as kinematic modes [2, 31]. Note that the number of degrees of freedom of the objects and the number of contact interactions depend on the symbolic state $s_{k(t)}$ with $k(t) = \lfloor t/T \rfloor$. Therefore, the dimension of the path may vary between the phases. For example, the symbolic state can express that a contact interaction should take place at a certain phase, which introduces a wrench interaction variable to the configuration space in that phase.

The functions c, h_{eq}, h_{ineq} , and hence the objectives in phase k , of the motion are parameterized by the discrete variable (or integers in mixed-integer programming) $s_k \in \mathcal{S}(S)$, representing the state of the symbolic domain. The possible discrete transitions between s_{k-1} and s_k are determined by the successor function $succ(\cdot, \cdot)$ which is a function of the previous state s_{k-1} and the discrete action a_k at phase k . Successor functions are defined by PDDL-like first-order logic-based rules. h_{sw} is a function that imposes transition constraints on the path between the kinematic modes. The quantity $\tilde{x}_0(S)$ is the scene dependent initial state. For fixed s , it is assumed that c, h_{eq} and h_{ineq} are differentiable.

The discrete actions a_k are of great importance for the rest of this paper. During tree search to find the action sequence, LGP expands the current node based on the possible action-object tuple. Once the scene becomes complex, the number of possible combinations of action-objects grows exponentially. This makes traversing the whole search space infeasible. Adding reasonable prior knowledge then becomes critical to prune irrelevant nodes.

The task or goal of the TAMP problem is defined sym-

bolically through the set $\mathcal{S}_{goal}(g)$ for the symbolic goal (a set of grounded literals) $g \in \mathbb{G}(S)$, (e.g., placing the yellow-box on the gray-table). A sequence of actions $a_{1:K}$ uniquely determines the sequence of symbolic states $s_{0:K}$ for a given initial symbolic state $s_0 = \tilde{s}_0(S)$. Therefore, we equivalently express that obtaining a solution to the LGP means finding an action sequence $a_{1:K}$ whose corresponding symbolic state sequence reaches the symbolic goal state and the corresponding nonlinear trajectory optimization problem is feasible. We define the feasibility of an action sequence $a_{1:K} = (a_1, \dots, a_K)$ via the existence of a respective path,

$$F_S(a_{1:K}) = \begin{cases} 1, & \text{if } \exists x : [0, KT] \rightarrow \mathcal{X} : (1b) - (1h) \\ 0, & \text{otherwise} \end{cases} \quad (2)$$

We also define a function that describes whether an object is involved in solving the task,

$$Z_S(o) = \begin{cases} 1, & \text{if } F_S(a_{1:K}) = 1 \rightarrow \exists a_k \in K : \theta(a_k) = o \\ 0, & \text{otherwise} \end{cases} \quad (3)$$

The $\theta(a)$ takes an action as parameter and retrieves the object involved in the action.

The complete LGP formulation (1) not only seeks to find a feasible solution, but also an action sequence that leads to the minimum trajectory costs (1a) compared to all other goal reaching sequences. We call a feasible solution a *solution to the TAMP/LGP problem*, whereas we call a solution that minimizes the cost an *optimal solution of the LGP*. Due to the vast number of possible discrete action sequences, obtaining optimal solutions is often intractable. Therefore, we mostly focus on obtaining a feasible solution.

The feasibility as defined in eq. 2 is a theoretical property of the resulting nonlinear program. In practice, we solve eq. 1 numerically by discretizing x with a finite number of collocation points in time, which leads to a finite dimensional optimization problem that we solve with an augmented-Lagrangian method that has the Gauss-Newton method in its inner loop. Therefore, $F_S(a_{1:K})$ is also determined numerically in the following way: If the accumulated constraint violations ($h_{eq} \neq 0, h_{ineq} > 0$) along the discretized trajectory are below a certain threshold, then the solution found by the optimizer is considered feasible, otherwise it is considered to be infeasible.

IV. VISUAL DEEP LEARNING MODELS AS PLANNING HEURISTICS

Here, we describe the deep learning models that we have developed. These models accept images as input and output information that assist TAMP to improve planning time.

A. Embedding Model

1) *The Vision Transformer (ViT)*: The inputs to the model are cropped image patches. These patches are of two types: the first type is the scene image and the second one is the canonical views of the objects that are queried (Fig. 2). The patches are first converted into an embedding, e.g., by a convolutional network. Then, the corresponding positional

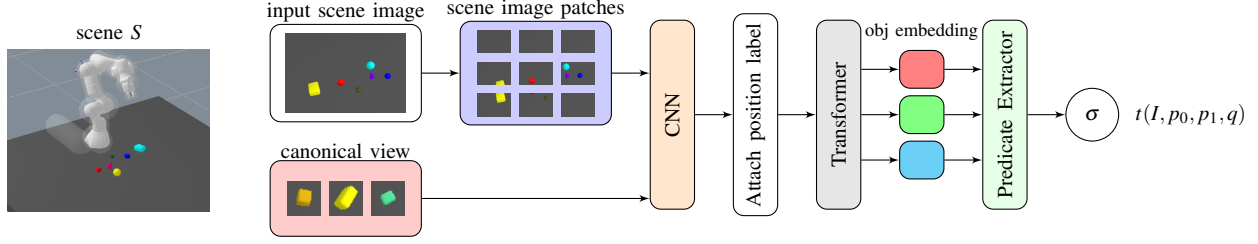


Fig. 2: Proposed neural net architecture based on transformer to predict the involvement of objects in the physical interaction for a given goal predicate.

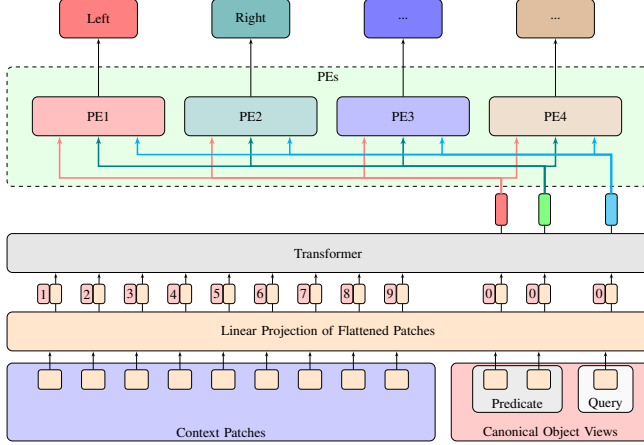


Fig. 3: The model structure. The input to the model consists of two major parts: (i) Context Patches: the cropped regions of the scene image, and (ii) Canonical Object Views: the descriptive image of the object to be queried. In the middle is the default Transformer model. Finally, there are four independent PEs, which correspond to different Predicates.

information is concatenated, which is later adopted in the transformer module.

In the default ViT architecture [26], the embedding at position zero is critical. It is not derived from the input patch, but instead can be learned and then used for the classification. This part is where our model differs from the original ViT. Our implementation extends the use of location labels to enable the model to perform the query task in a given situation, i.e., to determine whether the query object needs to be involved in the process of completing the task given the target state.

The location information and the embedding are then passed into the transformer. The transformer here is identical to the original transformer structure used for natural language processing [25]. However, unlike the traditional transformer, the vision transformer discards all the final embeddings of the patches from the scene image, and uses the final embedding of the zero position (Fig. 3). A multilayer perceptron is finally implemented to use this embedding to predict the relevancy of the queried object for several predicates - we call this component the Predicate Extractor (PE).

2) *The Residual Neural Network (ResNet)*: ResNets extract features by convolutional and residual blocks [30]. We built two versions (Fig. 4): For the first one ((D)efault-ResNet), two different sizes of ResNets are used to analyze the scene image and the canonical views separately. A fully linked network then produces three embeddings similar to the ViT-based model. The PE eventually receives the generated

embedding data to extract the relevance information for the query object given a task described by canonical object views. In the second one, which we named it as three dimensional ResNet, we first segment the scene image into patches as well, and stack them with the canonical view. Then, a residual neural network based on a 3D convolutional algorithm is used to extract the features (Fig. 4).

The features output from the residual-based neural network model can also be considered as embedded information and utilized by the *Predicate Extractor (PE)*.

B. Embedding Information Extraction

One option is to model *PE* to have the same output dimensionality as the number of the predicates that we are concerned with: for example, the predicates *on-the-left*, *on-the-right*, *in-front-of* and *behind*. In our current architecture, there is a single *PE* for each predicate as shown in Fig. 3.

We trained the model with batch polling method. This method is based on a polling approach (Fig. 6), where the data corresponding to different PEs are taken out in batches and used for the corresponding training, thus updating the corresponding PEs and the transformer they share. With the polling method, the transformer is updated with data from multiple predicates simultaneously at each gradient step, leading to a more reliable optimization.

C. Details of the Models

The goal of the present work is to learn a classifier t predicting an object set $O \subset \mathcal{O}$ that comprises the objects manipulated by the actions $a_{1:K}$ generated by a feasible nonlinear program $P(g, S)$. To realize this, the question is how the scene S and the goal configuration $S_{goal}(g)$ can be encoded as inputs to the classifier in a way that not only enables learning an accurate classifier but also generalizes to new scenes with (for example) varying numbers of objects. As mentioned previously, we argue that the sensor space, in this case camera images, has the potential to fulfill these requirements, while being a fixed size input. Formally, we assume that there is a generative process to produce an image $I \in \mathbb{R}^{w \times h \times 3}$ from the scene S , either via a rendering engine in simulation or a camera in the real world. We choose an image acquired by a tilted (virtual) camera to capture as much geometric information as possible.

Thanks to the attention mechanism, all we need to identify for our query is a canonical view per object $q \in \mathbb{R}^{32 \times 32}$ obtained in advance. Note that the dimensions of the canonical

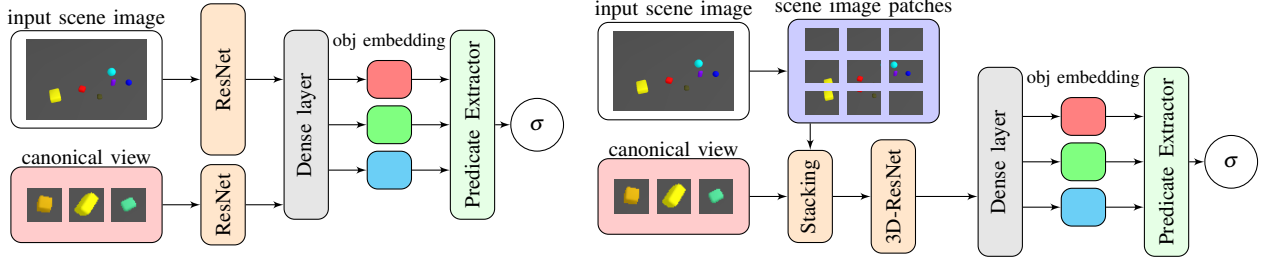


Fig. 4: Proposed architectures based on ResNet: (left) a model based on a conventional ResNet, and (right) a model based on a 3D-ResNet.

views here are same as the patches of scene image, which are also 32 by 32.

The input to the classifier $t(I, p_{0,\dots,L}, q_i)$ is the image I as three channel image $I \in \mathbb{R}^{w \times h \times 3}$ and canonical views for the objects, which, together with the goal predicate, describe the goal and the queried object, $p_{0,\dots,L} \in \mathbb{R}^{32 \times 32}$, q_i , respectively ($L = 1$ for our PEs in this study). Fig. 1 visualizes these input images, and Fig. 2 describes our architecture with a high level of abstraction. The network is trained to output the probability that the queried objects will be included in the manipulation plan which is the result of the nonlinear program $P(g, S)$

$$t(I, p_{0,\dots,L}, q_i) = p(Z_S(i) | F_S(a_{1:K}) = 1)_{o \in \mathcal{O}} \quad (4)$$

using the standard weighted binary cross-entropy loss

$$L(w) = \sum_{i \in \mathcal{O}} \eta Z_S(i) \log(t(I, p_{0,\dots,L}, q_i; w)) + (1 - Z_S(i)) \log(1 - t(I, p_{0,\dots,L}, q_i; w)). \quad (5)$$

The training data $\mathcal{D} = (I_i, P_i, Q_i)_{i=1}^d$ consists of scene images I_i with the canonical views of the predicate related objects P_i and query object Q_i . In our experiments the majority of the objects are irrelevant to the goal state. Therefore, the weighting factor $\eta \leq 1$ in eq. 5 turned out to be highly important to balance the loss, otherwise the network could achieve high accuracies by always predicting all objects to be irrelevant. If the prediction $t(I, p_{0,\dots,L}, q_i)$ for object i is higher than the threshold $\beta \in \mathbb{R}$, then we decide that $Z_S(i) = 1$, i.e., the object is relevant to the action sequence. In the experiments, we discuss the influence of this threshold.

For the ViT and the 3D-ResNet models, the scene image is first cropped into small patches of a predefined size, we use 32 by 32 in our experiments. For the ViT-based model, the patches of the scene images, along with the canonical views associated with the predicates and the query objects, are then fed into the first layer of the CNN network, which is used to transform them into an embedding with a size of 768 required by the transformer. The embedding converted from the scene image will then be attached with the corresponding position label, while the embedding generated from the canonical views will only be attached with position label zero. In essence, we only use the embeddings that capture the correlation between context patches and canonical views, and also between canonical views themselves, while discarding the correlation between context patches (for details,

see [26, 8]). These embeddings with labels attached are then fed into the standard transformer. Finally, we take out the output corresponding to the canonical views and feed them into the final fully connected layer, which has two layers and 512 neurons per layer (which we call the *Predicate Extractor*).

V. GUIDING MOTION PLANNING USING LEARNED VISUAL HEURISTICS

The learned classifier 4 can be utilized not only to predict the relevance of a given goal state represented by a goal predicate in a scene but also to guide the optimization of the mixed-integer program 1 as a heuristic, because it outputs a score which can be normalized and treated as probability of relevance. The goal of this heuristic is to find a set of objects $O \subset \mathcal{O}(S)$ such that the NLP $P(g, S)$ is feasible with all actions $a_{1:K}$ of the objects that are operated on $o \in O$ (i.e., $\theta(a) \in O$). In this way, the search space for finding a feasible NLP $P(g, S)$ could be reduced to a certain level, which improves the time complexity of the whole computation.

A. Admissible Heuristic

The classifier can be used as an admissible heuristic. Admissible in this context denotes that classification errors cannot prevent finding a solution to a feasible problem $P(g, S)$. This is realized by first evaluating $t(I, p_{0,\dots,L}, q_i) = p_i$ for all $o_i \in \mathcal{O}$, where q_i is the canonical view of o_i . The NLP $P(g, S)$ first searches for a feasible path with actions only considering the objects in $O \subset \mathcal{O}$. If it is feasible, then a solution will be given, otherwise the whole object set will be used for searching for a feasible path. However, for an infeasible problem, (i.e., a problem where there is no path which can lead to the goal state), all action-object tuples will be tested and, in this case, one therefore does not gain anything from using the learned network as an admissible heuristic.

B. Non-Admissible Heuristic

Similar to real-world settings, many objects are irrelevant for the goal state in our experiments. In addition, as long as the object set predicted by the classifier does not contain all of the objects that the NLP has to consider in order to find a feasible path, the NLP will clearly be unable to find a feasible path. In Sec. IV-C we mentioned the hyperparameter β as the threshold for the binary output $Z_S(i)$ (Eq. 4). In practice we can reduce β to a certain amount to increase the size of object set, and to have a higher probability to fully cover

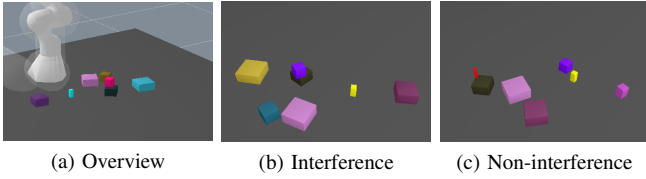


Fig. 5: An example of scene for the *on top* predicate.

the object set which is required by a feasible path finding procedure. Because the predicted object set is still a subset of the whole object set \mathcal{O} , the search space remains limited, thus the time complexity is expected to be improved. In the experiments, we evaluate and discuss the false infeasible rate and the time savings with respect to the feasibility threshold. For $\beta = 0$, this method becomes equivalent to the admissible heuristic of Section V-A.

VI. EXPERIMENTS

In our experiments, we consider two types of predicates: predicates describing objects on the same horizontal plane, such as *on-the-left* and *on-the-right*, and predicates describing objects in different planes, here we choose *on top*.

A. Data Generation

For all experiments, we place objects randomly on the table and a camera looking down 45 degrees to get the scene image. For the *on-top* predicate, also we randomly select a target object and then randomly place another object on top the target object. This is done in order to increase the likelihood that an object in the scene is task-relevant. Randomly distributing all objects would in almost all cases result in all objects being on the same horizontal plane, which would mean that for the *on-top* predicate, no other objects apart from those being part of the goal are relevant. Placing one object on top of the target ensures that the classifier is forced to make a decision whether it needs to be removed. This will depend on whether there is enough space for the goal object or not, and is therefore a nontrivial visual task. Fig. 1 depicts a typical scene for the same-plane case, where the task is to place object c relative to object d (e.g., on the left-hand side of object d). The classifier gets an extra input object e , and outputs the probability that object e has to be interacted with in order to complete the mentioned task. Fig 5 shows an example of the different-plane case.

Given this initial scene and a goal predicate indicating a spatial relationship between two objects, the LGP agent has to figure out both the sequence of high-level decisions and the motion plans that can realize those tasks jointly. If the agent successfully find a solution, the objects which have been manipulated according to the motion plan will be labeled as relevant and recorded. If no plan has been found, nothing is recorded.

As we mentioned before, for any given goal predicate, the majority of objects in the same scene S are irrelevant. Training a single model to output all 4 predicates simultaneously would mean that there are $2^4 = 16$ different labels, some of which (for example the *all true* case) never occur, resulting

in a very imbalanced dataset. We therefore decide to train 4 different model heads, the PEs, on a single binary label that is independently balanced. We still share weights of the model body, the details of which are described in section VI-B. This also means that the model scales gracefully to a large number of predicates.

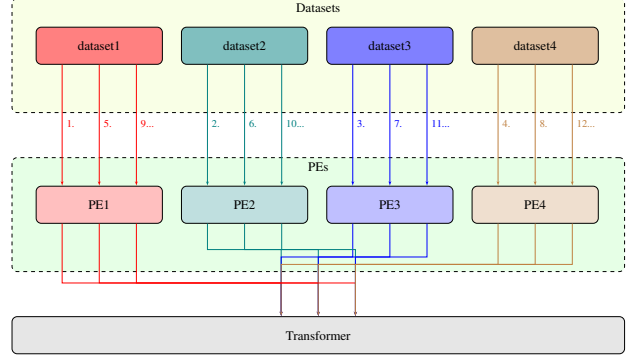


Fig. 6: The batch polling training Procedure

B. Training the multi-head model

We use 4 different model heads, one for each predicate. The body of the model is shared between these. The model is trained end-to-end in the following way by periodically taking a batch of data from each predicate's corresponding training set and train it, as shown in Fig. 6.

In an alternative approach, we also tried training each predicate head to convergence, before then moving on to the next. Unsurprisingly, the first approach we described works significantly better and used throughout the paper.

C. Predicates in the same plane

For the first phase of the experiments, we consider datasets for *on-the-left*, *on-the-right*, *in-front* and *behind*. Each dataset consists of 80,000 scenes in which there are between 3 and 10 objects in the scenes. Both of the training set and test set are generated with the same strategy. As mentioned before, the data imbalance is quite severe, with only about 14% of the queries being relevant and about 86% of the queries being irrelevant. Their data distribution is shown in the Table II. This imbalance imposes a challenge to the training dataset, we use $\eta = 0.86$. The network is trained with the **ADAM** optimizer (learning rate 0.00001, batch size 50).

D. On-top predicate

The training data contains 160,000 scenes, and the scenes also have various number of objects. 30% of the objects were relevant, 69% were irrelevant. The same imbalanced data issue arises here as well, we use $\eta = 0.66$. Again, the network is trained with the **ADAM** optimizer (learning rate 0.00001, batch size 50).

E. Efficiency Improvement on LGP

Finally, we compare the improvement brought by the model for sequential planning of actions using LGP. Because we found similar accuracies for the different models in

Test data	predicate	η	accuracy of classifier			performance on LGP		
			true rel.	true irrel.	total accuracy	runtime(s)	our runtime(s)	speed-up
same-plane	on-the-left	0.86	87%	94%	93%	3.06 ± 0.100	1.83 ± 0.065	40.2%
	on-the-right	0.86	88%	94%	93%	3.37 ± 0.104	1.87 ± 0.061	44.8%
	in-front	0.86	92%	96%	95%	3.64 ± 0.105	2.09 ± 0.062	42.6%
	behind	0.86	92%	95%	95%	3.36 ± 0.104	1.92 ± 0.065	42.8%
	total	0.86	90%	95%	94%	4.73	1.93	42.6%
different-plane	on	0.66	76%	89%	87%	4.47 ± 0.103	3.70 ± 0.101	17.2%

TABLE I: Accuracies on all validation sets and Running time comparison of LGP using vision transformer based deep learning model and original LGP. We report 67% confidence intervals.

	predicate	relevant (pct.)	irrelevant (pct.)
Training set	on-the-left	1235677 (13.7%)	7764323 (86.3%)
	on-the-right	1239533 (13.8%)	7760467 (86.2%)
	in-front	1239388 (13.8%)	7760612 (86.2%)
	behind	1248629 (13.9%)	7751371 (86.1%)
Test set	on-the-left	12448 (13.8%)	77552 (86.2%)
	on-the-right	12557 (14.0%)	77443 (86.0%)
	in-front	12401 (13.8%)	77599 (86.2%)
	behind	12495 (13.9%)	77505 (86.1%)
Training set	on	3650855 (30.4%)	8349145 (69.6%)
Test set	on	34689 (28.9%)	85311 (71.1%)

TABLE II: Distribution of labels in each dataset

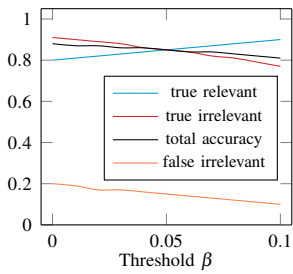


Fig. 7: Classifier accuracies

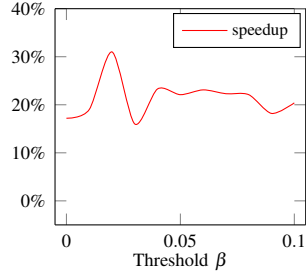


Fig. 8: LGP speed-up

Tbl. III are comparable and the runtime of LGP is only dependent on the accuracy of the model, we will focus on the results obtained using the model based on the Transformer. The results for both same-plane and different-plane datasets are shown in Tbl. I. We use the unmodified LGP as the baseline in our experiments. We do not show the impact of false irrelevant predictions of the classifier on the performance of the LGP. The influence of false irrelevant predictions on the runtime depends on other design choices like maximum runtime settings, which we factor out here.

The model performs better on the first dataset than on the second dataset, where we find an over 40% improvement on LGP. The second dataset has only 76 percent of true relevant due to the complexity of the predicates and the occlusion problem of the visual sensors, but the percentage of true relevant excluding false irrelevant, it still provides a 14% improvement.

As mentioned in section V-B, β directly affects the percentage of false irrelevant predictions, and the presence of false irrelevant prediction means that LGP has a high probability of not finding a feasible manipulation sequence.

We analyse how β would affect the runtime of LGP, and the results are shown in Fig. 7 and Fig. 8. The data in Fig. 7

is obtained by using the classifier trained on the data for the *on-top* predicate. As β increases, the model will adopt a more conservative strategy, so that more data will be marked as relevant. This also leads to a decrease in the number of false irrelevant cases. Since the action planning of LGP only considers the objects marked as "relevant", missing the objects that should be considered will result in LGP not finding a feasible plan. By adjusting Beta, we can avoid this situation to some extent.

F. Comparison of base models

In the last part of the experiments we compare the training results based on different visual deep learning models. All results in the experiments are obtained for the more complicated *on-top* predicate dataset.

Base Model	η	accuracy of classifier		
		true rel.	true irrel.	total acc.
Transformer	0.66	81%	89%	87%
3D-ResNet	0.66	79%	90%	87%
Default ResNet	0.66	81%	91%	88%

TABLE III: Comparison of accuracy rates using different base models

The best results are obtained using Default ResNet as the base model. This is followed by the 3D-ResNet-based model, and finally the Transformer-based model. It is worth noticing that even though we used very different vision models, the final results are not significantly different.

VII. CONCLUSION

As other visual models, our model can not circumvent occlusion and other ambiguities. Regardless of how the vision sensors are placed, it is possible for smaller objects to be obscured by larger objects, as is often the case for the *on-top* predicate. Furthermore, our dataset is relatively homogeneous in terms of object color selection, and the images are not affected by ambient lighting. While this can likely be addressed in a straightforward way by using photorealistic renderings and by randomizing lights and colors, it currently limits the number of scenarios the model can be used in.

Sequential manipulation planning becomes inefficient when applied to scenes where many non-relevant objects are present. We address this in this study by using a learned visual heuristic to classify objects into non-relevant and relevant objects, given a symbolic goal. We use a deep learning network that uses the scene image and canonical

views describing the goal and a queried object as input, and makes predictions about the involvement of this object in the scene. These predictions can be directly used in the process of sequential manipulation planning, thus directly improving the efficiency of planning.

We experimented on two different types of predicates: the first for spatial relations on the same plane and the second for spatial relations on different planes. They both perform well in terms of prediction accuracy and reduce the time required for sequential manipulation planning.

REFERENCES

- [1] 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.
- [2] M. A. Toussaint, K. R. Allen, K. A. Smith, and J. B. Tenenbaum, "Differentiable physics and stable modes for tool-use and manipulation planning," 2018.
- [3] I. Rodriguez, K. Nottensteiner, D. Leidner, M. Kaßbecker, F. Stulp, and A. Albu-Schäffer, "Iteratively refined feasibility checks in robotic assembly sequence planning," *IEEE Robotics and Automation Letters*, vol. 4, no. 2, pp. 1416–1423, 2019.
- [4] D. Driess, O. Oguz, and M. Toussaint, "Hierarchical task and motion planning using logic-geometric programming (hlgp)," in *RSS Workshop on Robust Task and Motion Planning*, 2019.
- [5] K. Kase, C. Paxton, H. Mazhar, T. Ogata, and D. Fox, "Transferable task execution from pixels through deep planning domain learning," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2020, pp. 10 459–10 465.
- [6] D. Driess, J.-S. Ha, R. Tedrake, and M. Toussaint, "Learning geometric reasoning and control for long-horizon tasks from visual input," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2021, pp. 14 298–14 305.
- [7] 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.
- [8] W. Yuan, C. Paxton, K. Desingh, and D. Fox, "Sornet: Spatial object-centric representations for sequential manipulation," in *Conference on Robot Learning*. PMLR, 2022, pp. 148–157.
- [9] S. Alili, A. K. Pandey, E. A. Sisbot, and R. Alami, "Interleaving symbolic and geometric reasoning for a robotic assistant," in *ICAPS Workshop on Combining Action and Motion Planning*, 2010.
- [10] 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.
- [11] L. de Silva, A. K. Pandey, M. Gharbi, and R. Alami, "Towards combining htn planning and geometric task planning," *arXiv preprint arXiv:1307.1482*, 2013.
- [12] N. T. Dantam, Z. K. Kingston, S. Chaudhuri, and L. E. Kavraki, "An incremental constraint-based framework for task and motion planning," *The International Journal of Robotics Research*, vol. 37, no. 10, pp. 1134–1151, 2018.
- [13] 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.
- [14] E. Erdem, K. Haspalamutgil, C. Palaz, V. Patoglu, and T. Uras, "Combining high-level causal reasoning with low-level geometric reasoning and motion planning for robotic manipulation," in *2011 IEEE International Conference on Robotics and Automation*. IEEE, 2011, pp. 4575–4581.
- [15] F. Lagriffoul, D. Dimitrov, A. Saffiotti, and L. Karlsson, "Constraint propagation on interval bounds for dealing with geometric backtracking," in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2012, pp. 957–964.
- [16] 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.
- [17] T. Lozano-Pérez and L. P. Kaelbling, "A constraint-based method for solving sequential manipulation planning problems," in *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2014, pp. 3684–3691.
- [18] Y. Shoukry, P. Nuzzo, I. Saha, A. L. Sangiovanni-Vincentelli, S. A. Seshia, G. J. Pappas, and P. Tabuada, "Scalable lazy smt-based motion planning," in *2016 IEEE 55th Conference on Decision and Control (CDC)*. IEEE, 2016, pp. 6683–6688.
- [19] D. Hadfield-Menell, C. Lin, R. Chitnis, S. Russell, and P. Abbeel, "Sequential quadratic programming for task plan optimization," in *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2016, pp. 5040–5047.
- [20] M. Toussaint, "Logic-geometric programming: An optimization-based approach to combined task and motion planning," in *Twenty-Fourth International Joint Conference on Artificial Intelligence*, 2015.
- [21] M. Toussaint, J.-S. Ha, and D. Driess, "Describing physics for physical reasoning: Force-based sequential manipulation planning," *IEEE Robotics and Automation Letters*, vol. 5, no. 4, pp. 6209–6216, 2020.
- [22] J.-S. Ha, D. Driess, and M. Toussaint, "A probabilistic framework for constrained manipulations and task and motion planning under uncertainty," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2020, pp. 6745–6751.
- [23] F. R. Hogan, E. R. Grau, and A. Rodriguez, "Reactive planar manipulation with convex hybrid mpc," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2018, pp. 247–253.
- [24] R. Deits and R. Tedrake, "Footstep planning on uneven terrain with mixed-integer convex optimization," in *2014 IEEE-RAS international conference on humanoid robots*. IEEE, 2014, pp. 279–286.
- [25] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.
- [26] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly *et al.*, "An image is worth 16x16 words: Transformers for image recognition at scale," *arXiv preprint arXiv:2010.11929*, 2020.
- [27] N. Carion, F. Massa, G. Synnaeve, N. Usunier, A. Kirillov, and S. Zagoruyko, "End-to-end object detection with transformers," in *European conference on computer vision*. Springer, 2020, pp. 213–229.
- [28] Y. Guo, S. Yang, W. Chen, L. Ma, D. Xie, and S. Pu, "2nd place solution for iccv 2021 vipriors image classification challenge: An attract-and-repulse learning approach," *ArXiv*, vol. abs/2206.06168, 2022.
- [29] S. Nguyen, O. S. Oguz, V. N. Hartmann, and M. Toussaint, "Self-supervised learning of scene-graph representations for robotic sequential manipulation planning," in *CoRL*, 2020, pp. 2104–2119.
- [30] 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.
- [31] M. Mason, "The mechanics of manipulation," in *Proceedings. 1985 IEEE International Conference on Robotics and Automation*, vol. 2. IEEE, 1985, pp. 544–548.