Augment-Connect-Explore: a Paradigm for Visual Action Planning with Data Scarcity

Martina Lippi*[1], Michael C. Welle*[2], Petra Poklukar[2], Alessandro Marino[3], Danica Kragic[2]

***Abstract*— Visual action planning particularly excels in applications where the state of the system cannot be computed explicitly, such as manipulation of deformable objects, as it enables planning directly from raw images. Even though the field has been significantly accelerated by deep learning techniques, a crucial requirement for their success is the availability of a large amount of data. In this work, we propose the Augment-Connect-Explore (ACE) paradigm to enable visual action planning in cases of data scarcity. We build upon the Latent Space Roadmap (LSR) framework which performs planning with a graph built in a low dimensional latent space. In particular, ACE is used to *i)* Augment the available training dataset by autonomously creating new pairs of datapoints, *ii)* create new unobserved Connections among representations of states in the latent graph, and *iii)* Explore new regions of the latent space in a targeted manner. We validate the proposed approach on both simulated box stacking and real-world folding task showing the applicability for rigid and deformable object manipulation tasks, respectively.**

I. Introduction

The performance of robotics tasks has been significantly accelerated by integration of deep learning techniques [1], [2], which have shown to be promising in visual action planning [3]. Given a start *observation* of the system, the goal of visual action planning is to produce *i)* an action plan comprised of the actions required to reach a desired state, and *ii)* a visual plan containing observations, i.e., images, of intermediate states that will be traversed during the execution of the planned actions. In this way, the planner can be given raw image observations which is crucial in applications where the state of the system cannot be easily described analytically, for instance as in manipulation of deformable objects. The supporting visual plan additionally improves the interpretability of these methods [3], [4].

While it has been shown that visual dynamics used for planning can be learned directly from images, several approaches considered planning in low-dimensional latent space (discussed in Sec. II). These methods reduce the complexity of planning in the image space but generally depend on vast amount of data to train reliable policies as they require a large amount of long rollouts for successful planning. In practice, this can hinder their applicability to real robotic hardware.

Therefore, in this work, we propose a method for performing visual action planning in case of data scarcity. We build on our Latent Space Roadmap (LSR) framework [5], [6] that first learns a low-dimensional latent space from input images and then builds a graph, called LSR, that is used to perform planning in this space. We tackle data scarcity with the proposed *Augment-Connect-Explore* (ACE) paradigm that is based on: *i) Augment:* creating new informative pairs of datapoints exploiting demonstrated actions to improve the latent space structure. *ii) Connect:* increasing the connectivity of the LSR by building new connections, i.e. *shortcuts*, among nodes to improve the capability to traverse the latent space. *iii) Explore:* proposing *targeted* exploratory actions by leveraging latent representations as well as collected actions to explore new states in an efficient manner. Our contributions can be summarized as follows:

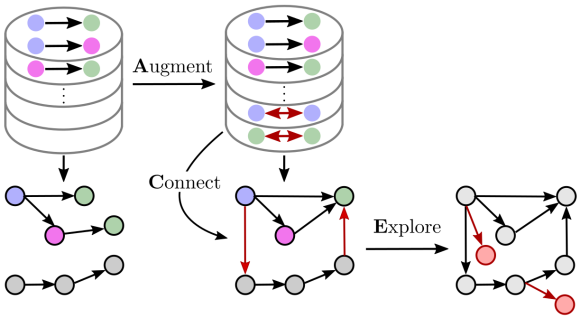

Fig. 1: Overview of our ACE paradigm: (1) gaining new similar datapairs by **A**ugmenting existing ones, (2) building unseen **C**onnections in the latent space, and (3) efficiently **E**xploring new regions. Color represent the underlying states of the system (see Sec. III for details).

- We introduce the ACE paradigm to address data scarcity in visual action planning by augmenting, connecting and exploring.
- To realize ACE, we design a novel Suggestion Module that proposes possible actions from a given state, which is used in conjunction with a simple neural network that predicts the next latent state given the current state and desired action.
- We thoroughly analyse the individual and cumulative effects of ACE components on a simulated box stacking task and demonstrate improved performance of the combined ACE framework on a real-world folding task under data scarcity.

II. Related Work

Several approaches learn the visual dynamics directly from images and use it for planning. In [3] visual foresight plans for deforming a rope into desired configurations are generated with Context Conditional Causal InfoGANs. The

*These authors contributed equally (listed in alphabetical order).
[1]Roma Tre University, Rome, Italy
[2]KTH Royal Institute of Technology Stockholm, Sweden
[3]University of Cassino and Southern Lazio, Cassino, Italy

learned rope inverse dynamics is then considered to reach the configurations in the generated plan. In [7] Long-Short Term Memory blocks are used to compose a video prediction model predicting the stochastic pixel flow from frame to frame given the action. This model is then integrated in a Model Predictive Control (MPC) framework to produce visual plans and push objects of interest. Building on the visual foresight frameworks, [8] proposes the VisuoSpatial Foresight which integrates the depth map information with the pure RGB data to learn the visual dynamic model of fabrics in a simulated environment. An extension of this approach is given in [9] where the main states of the framework are improved.

To reduce the complexity of planning in the image space, low-dimensional latent space have been explored in several studies, e.g., [10]-[11]. A framework for global search in latent space is presented in [10], where motion planning is performed directly in this latent space using an RRT-based algorithm with collision checking and latent space dynamics modelled as neural networks. Contrastive learning is used in [12] to derive a predictive model in the latent space that is exploited to find rope and cloth flattening actions. Latent space goal-conditioned predictors, formulated as hierarchical models, are introduced [13] to limit the search space to trajectories that lead to the goal configuration and thus to perform long-horizon visual planning. Latent planning has also been successfully applied in Reniforcement Learning (RL) settings, like for example in [14] for model-based offline RL and in [15] for hierarchical RL. The combination of RL with graph structures in the latent space is explored in [11], where a node is created for each encoded observation. Building on [11], temporal closeness of the consecutive observations in the trajectories is also exploited in [4]. However, the above methods generally require a large amount of long rollouts for successful planning. Therefore, in this work, we tackle visual action planning for scenarios with scarce training data.

III. Preliminaries and Problem Statement

In this section we provide preliminary notions about the considered dataset and the visual action planning. Then, we formalize the problem addressed in this work and present the LSR framework, which we extend to tackle data scarcity.

A. Dataset structure

Let $\mathcal{O}$ be the space of all possible observations, i.e., images, of the system's states. We consider a training dataset $\mathcal{T}_o$ consisting of tuples (O_1, O_2, ρ), where O_1 is an observation of the start state, O_2 an observation of the successor state, and ρ a variable denoting the respective action between the states. Here, an action is defined as a single transformation that brings the system to a new state different from the starting one. For example, in Fig. 2, an action corresponds to moving a box. The variable $\rho = (a, u)$ is composed of a binary variable $a \in \{0, 1\}$ indicating whether or not an action occurred and a variable u containing the task-dependent action-specific information (in case an action occurred, i.e. $a = 1$). We say that no action was performed, i.e., $a = 0$, if

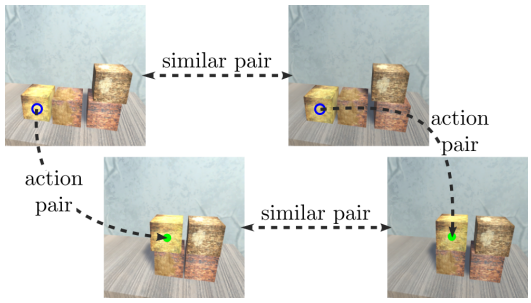

Fig. 2: Examples of similar and action pairs. The similar pairs show the same underlying state, while the observations in the action pairs show different underlying states.

observations O_1 and O_2 are different variations of the same (unknown) underlying state of the system. In the bottom row of Fig. 2, the observations exhibit lightning and slight positional variations, but correspond to the same underlying state of the system determined by the position of the boxes. We refer to a tuple in the form $(O_1, O_2, \rho = (1, u))$ as an *action pair* and $(O_1, O_2, a = 0)$ as a *similar pair* (shown in Fig. 2).

B. Visual Action Planning

Let $\mathcal{U}$ be the set of possible actions of the system. A visual action plan is the combination of an action plan P_u and a visual plan P_o that lead the system from a given start $O_s \in \mathcal{O}$ to a goal observation $O_g \in \mathcal{O}$, i.e., such that $P_o = \{O_s = O_1, O_2, ..., O_N = O_g\}$ and $P_u = \{u_1, u_1, ..., u_{N-1}\}$, where $u_n \in \mathcal{U}$ produces a transition between consecutive observations O_n and O_{n+1} for each $n \in \{0, ..., N-1\}$.

To reduce the complexity of the problem, we build a low-dimensional latent space $\mathcal{Z}$ encoding $\mathcal{O}$ that aims to capture the underlying states of the system.

Definition 1: The *latent mapping* function $\xi : \mathcal{O} \to \mathcal{Z}$ maps an observation $O_n \in \mathcal{O}$ into its latent representation $z_n \in \mathcal{Z}$. The *observation generator* function $\omega : \mathcal{Z} \to \mathcal{O}$ retrieves a possible observation $O_n \in \mathcal{O}$ associated with a latent representation $z_n \in \mathcal{Z}$.

Definition 2: The *latent dynamic function* $f : \mathcal{Z} \times \mathcal{U} \to \mathcal{Z}$ transitions the system through the latent space.

Given these functions, a way to perform visual action planning is then to map the start and goal observation into the latent space to obtain $z_s = \xi(O_s)$, $z_g = \xi(O_g)$, and then perform planning directly in $\mathcal{Z}$ by obtaining a latent plan $P_z = \{z_s = z_1, z_2, ..., z_N = z_g\}$. Finally, a visual plan P_o can be derived by mapping latent encodings, obtained when applying the latent dynamic function f on the action plan P_u, with the observation generator ω. Note that in practice the functions ξ, ω and f are unknown and need to be approximated. The quality of these approximations depends on the available amount of observations $\mathcal{T}_o$ of the system.

C. Problem Statement

When $\mathcal{T}_o$ is *scarce*, it might not contain all possible latent states associated with the system as well as possible

transitions among them. In this work we are interested in solving the following problem.

Problem 1: Given a *scarce* training dataset $\mathcal{T}_o$ as well as a start $O_s \in \mathcal{O}$ and a goal observation $O_g \in \mathcal{O}$, our objective is to find the related visual action plan (P_o, P_u).

To solve it, we exploit a simple insight that two latent states are similar if the same set of actions can be applied to both, and if these, in turn, also yield the same set of consecutive states. We first define the set of actions that can be applied to a given latent state z.

Definition 3: A *suggestion function* $\eta : \mathcal{O} \to \mathcal{U}$ provides a subset $\eta(O_i) = \mathcal{U}_i \subseteq \mathcal{U}$ of actions that can be applied from an observation $O_i \in \mathcal{O}$.

Given the suggestion function η and latent dynamic function f, we define similar states as follows.

Definition 4: The states $z_i, z_j \in \mathcal{Z}$ corresponding to the observations $O_i, O_j \in \mathcal{O}$ are said to be *similar* if

- $\eta(O_i) = \eta(O_j) = \widetilde{\mathcal{U}}$, and
- $\{f(z_i, u)\} = \{f(z_j, u)\}$ for every $u \in \widetilde{\mathcal{U}}$.

D. Latent Space Roadmap Framework

We addressed the problem of visual action planning for scenarios of *complete* training datasets, i.e., those covering all possible states and transitions among them, in our earlier works [5], [6] by introducing the Latent Space Roadmap (LSR) framework. The basic idea of this framework is to perform planning in the low dimensional latent space $\mathcal{Z}$ by *i)* structuring it to respect the underlying states of the system, and *ii)* building a graph directly in this latent space to guide the planning.

To address point *i)*, we define the concept of covered regions. We map the training dataset $\mathcal{T}_o$ described in Section III-A into the latent space $\mathcal{Z}$ to obtain a set of *covered* states $\mathcal{T}_z = \{z_1, ..., z_M\} \subset \mathcal{Z}$, i.e., $\mathcal{T}_z = \xi(\mathcal{T}_o)$, for which we make the following assumption as in [5], [6].

Assumption 1: Given a covered state $z \in \mathcal{T}_z$, there exists $\varepsilon > 0$ such that any other state z' in the $\varepsilon-$neighborhood $N_\varepsilon(z)$ of z can be considered as the same underlying state.

We define the union of ε-neighbourhoods of the covered states $z \in \mathcal{T}_z$ as *covered subspace*

$$\mathcal{Z}_{sys} = \bigcup\nolimits_{z \in \mathcal{T}_z} N_\varepsilon(z) \subset \mathcal{Z}, \tag{1}$$

which can be rewritten as the union of m path-connected components [6] called *covered regions* and denoted by $\{\mathcal{Z}^i_{sys}\}^m_{i=1}$. Note that in a well structured latent space, each covered region encodes a possible underlying state of the system. We define a set of transitions that connect covered regions. A *transition function* $f^{i,j}_z : \mathcal{Z}^i_{sys} \times \mathcal{U} \to \mathcal{Z}^j_{sys}$ maps a point $z^i \in \mathcal{Z}^i_{sys}$ to a point $z^j \in \mathcal{Z}^j_{sys}$ when applying an action $u \in \mathcal{U}$, with $i, j \in \{1, 2, ..., m\}$ and $i \neq j$. Given Z_{sys} and the transition functions $f^{i,j}$, we then define the Latent Space Roadmap:

Definition 5: A Latent Space Roadmap is a directed graph $LSR = (\mathcal{V}_{LSR}, \mathcal{E}_{LSR})$ where each vertex $v_i \in \mathcal{V}_{LSR} \subset \mathcal{Z}_{sys}$ for $i \in \{1, ..., m\}$ is a representative of the covered region $\mathcal{Z}^i_{sys} \subset \mathcal{Z}_{sys}$, and an edge $e_{i,j} = (v_i, v_j) \in \mathcal{E}_{LSR}$ represents a transition function $f^{i,j}_z$ between the corresponding covered regions $\mathcal{Z}^i_{sys}$ and $\mathcal{Z}^j_{sys}$ for $i \neq j$.

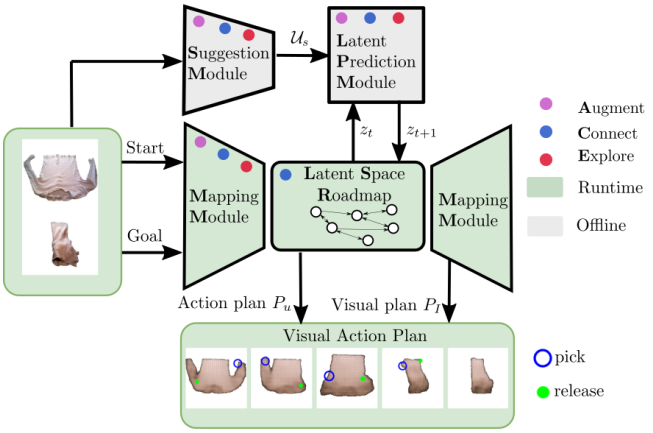

Fig. 3: Overview of the proposed architecture to generate visual action plans. The modules involved in each phase of the ACE paradigm are highlighted with respective colored dots. The modules only used offline are marked in grey while the ones also used online are marked in green.

Two main modules compose the LSR framework. First, a Mapping Module (MM) implements the mapping function ξ and observation generator ω defined in Def. 1 with a VAE framework. These are learned using a contrastive loss term, also called *action* term, which attracts states belonging to similar pairs and repels states belonging to action pairs to a minimum distance d_m. Second, an LSR module implements the LSR defined in Def. 5 by applying clustering in $\mathcal{Z}$ to approximate the covered regions $\mathcal{Z}^i_{sys}$. Each obtained cluster is associated with a node in the LSR and edges among them are created using action pairs in the training dataset $\mathcal{T}_o$. In this process, average action specifics are also endowed in the edges to retrieve the action plan P_u (see the Action Averaging Baseline in [6]). More details can be found in [5], [6].

IV. Overview of the Approach

In order to perform visual planning in case of data scarcity, the proposed ACE paradigm aims to:

1) Augment the available dataset $\mathcal{T}_o$, autonomously creating new similar pairs.
2) Create new unobserved connections in the latent space $\mathcal{Z}$, increasing the set of transition functions f^{ij}_z and number of respective edges in the LSR.
3) Explore new regions of $\mathcal{Z}$ in an efficient and guided manner, increasing the covered subspace $\mathcal{Z}_{sys}$.

To realise the ACE paradigm we extend the LSR framework with a Latent Prediction Model (LPM) and a Suggestion Module (SM). An overview of the overall ACE architecture including all the modules is shown in Fig. 3. We mark in green the modules used at run time to produce the visual action plan (bottom) from start O_s and goal observations O_g (left), and in grey the ones only used offline.

The LPM module approximates the latent dynamics function in Def. 2 which, given a latent state z_i and an action

$u \in \mathcal{U}$, predicts a potential next state z_{i+1}. Note that LPM implicitly assumes a given MM. The SM module approximates the suggestion function η in Def. 3 and, given an observation O_i, suggests a set of potential actions $\mathcal{U}_i$ that are possible to perform. The input image O_i can be either an observation of the current state or an observation generated by ω.

To realize point 1), we rely on Def. 4 and employ SM and LPM to find novel similar pairs that are added to $\mathcal{T}_o$ to obtain the augmented training dataset $\overline{\mathcal{T}_o}$. The latter is then used to obtain a new mapping function approximation $\bar{\xi}$ by updating the MM, which leads to an enhanced structure of the latent space $\mathcal{Z}$.

Regarding point 2), we use SM and LPM along with the covered subspace $\mathcal{Z}_{sys}$ to identify previously unseen transitions f_z^{ij} that are possible to execute, called *valid* transitions. These are added to the LSR in the form of new edges referred to as *shortcuts*.

Finally, point 3) is realized in a similar fashion where SM suggests the set of possible actions $\mathcal{U}_i$ from the current state z_i and LPM predicts potential next states out of which we select the most promising one as the exploratory action, as described in Sec. V-C. This enables exploring new regions of the latent space $\mathcal{Z}$ in a guided manner.

The ACE paradigm improves the individual components of the LSR framework which is then used to perform visual action planning. Note that even though we focus on the LSR framework, ACE is general and applicable to many other contexts, e.g., the proposed targeted exploration approach can be easily integrated into an RL setting.

A. Models for LPM and SM

We model LPM as a simple multilayer perceptron (MLP). During augmentation and connection phases, we also leverage the covered subspace $\mathcal{Z}_{sys}$ defined in (1): we consider a state z_j predicted by the LPM reliable only if it falls within the covered subspace, i.e., $z_j \in \mathcal{Z}_{sys}$. We refer to the LPM including the covered subspace check as reliable LPM (LPM-R) in the following.

The SM is built based on two core observations: *i)* several valid actions can be applied to the same state, and *ii)* the same action can be applied to different states. We model the suggestion function η with a Siamese network trained with a contrastive loss that encourages clustering of the states from which the same subset of actions can be performed.

In detail, we build the training dataset for the SM, denoted by $\mathcal{T}_o^{SM}$, by rearranging the observations in the training tuples in $\mathcal{T}_o$ depending on the actions. A similar pair $(O_1, O_2, s = 1)$, where s is the similarity signal, is added to $\mathcal{T}_o^{SM}$ if the same action specifics u is applied from O_1 and O_2 in $\mathcal{T}_o$, i.e., if there exist $(O_1, -, \rho = (1, u)) \in \mathcal{T}_o$ and $(O_2, -, \rho = (1, u)) \in \mathcal{T}_o$, where $-$ denotes any other observation. On the other hand, a dissimilar pair $(O_1, O_2, s = 0)$ is added to $\mathcal{T}_o^{SM}$ when different action specifics are applied from O_1 and O_2 in $\mathcal{T}_o$, i.e., if there exist $(O_1, -, \rho = (1, u_1)) \in \mathcal{T}_o$ and $(O_2, -, \rho = (1, u_2)) \in \mathcal{T}_o$ with $u_1 \neq u_2$. Training the Siamese network with the dataset $\mathcal{T}_o^{SM}$ results in a latent space $\mathcal{Z}'$ different from $\mathcal{Z}$, i.e., $\mathcal{Z}' \cap \mathcal{Z} = \emptyset$. The latent space $\mathcal{Z}'$ is then clustered and each cluster $\mathcal{C} \subset \mathcal{Z}'$ is labeled with the set $\mathcal{U}$ containing all the actions that are executed starting from the points contained in $\mathcal{C}$.

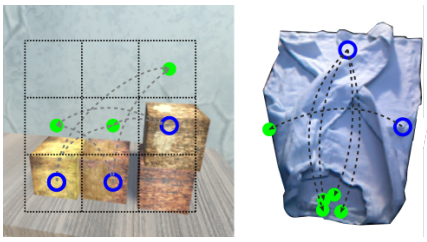

Fig. 4: Example of the set of suggested actions $\mathcal{U}$ using the SM for a box stacking (left) and T-shirt folding (right) task. The blue rings mark pick locations, while the green circles place locations.

At run time, a novel observation O_i is fed into the Siamese network to obtain its latent representation $z_i' \in \mathcal{Z}'$ and the set of suggested actions $\mathcal{U}_i$ associated with the closest cluster $\mathcal{C}$ as visualized in Fig. 4.

V. ACE Paradigm

In this section we present in detail the individual components of our ACE paradigm and then provide a summary of the full framework.

A. Augment

The proposed augmentation procedure builds new similar pairs based on Def. 4. In particular, if the same set of actions applied from different observations O_1 and O_2 leads to the same underlying states, we consider the two starting observations as a similar pair. In doing so, we discover similar pairs among states that are further apart in the latent space $\mathcal{Z}$. Note that this occurs in practice since the latent mapping ξ is only an approximation. Therefore, to improve the structure of $\mathcal{Z}$, it is crucial to identify more similar pairs in the dataset $\mathcal{T}_o$ such that $\bar{\xi}$ is re-learned to map the same underlying states close together.

Moreover, in our setting, no labels about the underlying states contained in the training observations that could be exploited for augmenting the dataset are provided. In contrast, we only have access to the information of whether two observations are similar or there is an action between them.

Algorithm 1 summarizes the augmentation procedure. Given the training dataset $\mathcal{T}_o$ and a search radius r determining the search area around covered states, we encode all observations $O_i \in \mathcal{T}_o$ to obtain $\mathcal{T}_z \subset \mathcal{Z}$ (line 1) and initialize the augmented dataset $\overline{\mathcal{T}}_o$ (line 2). For each latent state $z_i \in \mathcal{T}_z$, we check if a new similar pair can be identified. We first obtain the set $\mathcal{U}_i$ of possible actions from z_i using the SM (line 4). Then, we define the set $\mathcal{L}_i$ of covered latent states which are within the search radius r (line 5), i.e., $\mathcal{L}_i = \{z_j \in \mathcal{T}_z \,|\, z_j \in N_r(z_i)\}$. This is followed by a descent sorting with respect to the distance of each $z_j \in \mathcal{L}_i$ to z_i (line 6). Note that we limit the search in a radius only for computational reasons. Since the latent space $\mathcal{Z}$ already has

Algorithm 1 Augmentation Algorithm

Require: Training dataset $\mathcal{T}_o$, search radius r

1: $\mathcal{T}_z \leftarrow \text{MM}(\mathcal{T}_o)$
2: $\overline{\mathcal{T}}_o := \mathcal{T}_o$
3: **for each** $z_i \in \mathcal{T}_z$ **do**
4: $\quad \mathcal{U}_i \leftarrow \text{SM}(O_i)$
5: $\quad \mathcal{L}_i \leftarrow$ search in radius$(\mathcal{T}_z, z_i, r)$
6: $\quad \mathcal{L}_i \leftarrow$ descent sort$(\mathcal{L}_i)$
7: $\quad$ found := False
8: $\quad$ **for each** $z_j \in \mathcal{L}_i$ and not found **do**
9: $\quad\quad O_j \leftarrow$ get observation$(\mathcal{T}_o, j)$
10: $\quad\quad \mathcal{U}_j \leftarrow \text{SM}(O_j)$
11: $\quad\quad$ **if** $\mathcal{U}_i \equiv \mathcal{U}_j$ **then**
12: $\quad\quad\quad \mathcal{Z}_i^p, \mathcal{Z}_j^p \leftarrow$ LPM-R$(z_i, \mathcal{U}_i)$, LPM-R$(z_j, \mathcal{U}_i)$
13: $\quad\quad\quad \mathcal{Z}_i^n, \mathcal{Z}_j^n \leftarrow$ nearest$(\mathcal{T}_z, \mathcal{Z}_i^p)$, nearest$(\mathcal{T}_z, \mathcal{Z}_j^p)$
14: $\quad\quad\quad$ **if** $\mathcal{Z}_i^n \equiv \mathcal{Z}_j^n$ **then**
15: $\quad\quad\quad\quad \overline{\mathcal{T}}_o := \overline{\mathcal{T}}_o \cup \{(O_i, O_j, a = 0)\}$
16: $\quad\quad\quad\quad$ found := True

return $\overline{\mathcal{T}}_o$

Algorithm 2 Connection Algorithm

Require: $LSR = (\mathcal{V}_{LSR}, \mathcal{E}_{LSR})$, neighborhood size ε

1: **for each** $z_i \in \mathcal{V}_{LSR}$ **do**
2: $\quad O_i \leftarrow \text{MM}(z_i)$
3: $\quad \mathcal{U}_i \leftarrow \text{SM}(O_i)$
4: $\quad$ **for each** $u \in \mathcal{U}_i$ **do**
5: $\quad\quad z_n \leftarrow \text{LPM}(z_i, u)$
6: $\quad\quad$ **if** $\|z_j - z_n\|_1 < \varepsilon$ for $z_j \in \mathcal{V}_{LSR}, i \neq j$ **then**
7: $\quad\quad\quad \mathcal{E}_{LSR} \leftarrow$ create edge (z_i, z_j, u)

return LSR

a certain structure inferred from the non-augmented dataset $\mathcal{T}_o$ during training of MM, we avoid checking states that are too far away from the current and likely not similar to it.

At this point, we analyze the covered states $z_j \in \mathcal{L}_i$. Starting from the first z_j, we take the corresponding observation O_j in the training dataset (line 9) and obtain the set of potential actions $\mathcal{U}_j$ (line 10). If all the actions in the sets $\mathcal{U}_i, \mathcal{U}_j$ coincide, we consider the sets of respective predicted states $\mathcal{Z}_i^p, \mathcal{Z}_j^p$ (line 12) made by the LPM-R, where the reliability condition discussed in Sec. IV-A is checked. Lastly, the sets $\mathcal{Z}_i^n, \mathcal{Z}_j^n$ consisting of closest covered latent states in $\mathcal{T}_z$ with respect to $\mathcal{Z}_i^p, \mathcal{Z}_j^p$, respectively, are found. If $\mathcal{Z}_i^n, \mathcal{Z}_j^n$ coincide, a new similar pair $(O_i, O_j, a = 0)$ is added to the augmented dataset $\overline{\mathcal{T}}_o$, otherwise the next state $z_j \in \mathcal{L}_i$ is analyzed.

B. Connect

For graph-based planning methods it is essential to have a good connectivity of nodes. Although more connections can be built by collecting more data, a more efficient approach involves building *shortcuts*, i.e., connections between nodes that are not directly induced by the training set. In this work, we infer them using the SM and LPM modules. Note that it is important to add correct shortcuts as erroneous connections can be very detrimental for the graph planning capabilities, leading to unfeasible plans.

Algorithm 2 summarizes the proposed method for building shortcuts. The basic idea is that if an action u suggested by the SM in a certain state z_i leads to a covered state z_j, then the respective transition can be considered as valid and can be added to the LSR. In detail, given the LSR and the neighborhood size ε, we iterate over the states in the set of nodes $\mathcal{V}_{LSR}$ of the LSR. For each state z_i in $\mathcal{V}_{LSR}$, we generate the respective observation O_i through the observation generator ω of the MM (line 2) and obtain the set of potential actions $\mathcal{U}_i$ by the SM. For each $u \in \mathcal{U}_i$, LPM predicts the next state z_n obtained from z_i (line 5). If the predicted next state z_n falls in the ε-neighborhood of any other state $z_j \in \mathcal{V}_{LSR}$ in the LSR with $i \neq j$, an edge between z_i and z_j is added in the edge set $\mathcal{E}_{LSR}$ of the LSR (line 7). We also endow the edge with the new predicted action u for action planning purposes as discussed in III-D.

C. Explore

The challenges of finding suitable actions for exploration of the latent space $\mathcal{Z}$ are twofold: *i)* finding valid actions that can be performed in the current state, and *ii)* choosing the action that is most beneficial to the system.

The SM model provides a solution to the first problem as it outputs a set of valid actions $\mathcal{U}_i$ for an observation O_i corresponding to the current state z_i as described in Sec. IV-A. For the second problem, we propose to undertake the action that leads to the most unexplored area of the latent space $\mathcal{Z}$ at each exploration step.

The approach is summarized in Algorithm 3. Given the training dataset $\mathcal{T}_z$ and the current state observation O_i, we first map both into $\mathcal{Z}$ with the mapping function ξ of the MM (lines 1-2). Then, we retrieve the set of potential actions $\mathcal{U}_i$ from the current state z_i through the SM. We initialize an empty auxiliary exploration list $\mathcal{L}_e$. For each action $u \in \mathcal{U}_i$, we predict the next state z_n using the LPM (line 6) and compute the distance d_i from z_i to the nearest covered state in $\mathcal{T}_z$ (line 7). The tuple given by the action u and distance d_i is added to the exploration list $\mathcal{L}_e$. Once all the actions in $\mathcal{U}_i$ have been analyzed, we return the action u_e (line 9) that leads to the furthest latent state as the exploratory one. As described in the following, the observation O_{i+1} obtained after executing u_e is used to create a new action pair $(O_i, O_{i+1}, \rho = (1, u_e))$ that is added to $\mathcal{T}_o$. The latent space is explored by executing Algorithm 3 n_e times.

D. LSR with ACE

In this section, we describe how the ACE components are combined within the LSR framework, summarized in Algorithm 4. Given the training dataset $\mathcal{T}_o$, the hyperparameters r and ε as well as the number of total exploration steps n_e, we first build the models employed in the ACE paradigm (line 1). Secondly, we generate the augmented dataset $\overline{\mathcal{T}_o}$ following Algorithm 1 with the search radius r and use it to

Algorithm 3 Exploration Algorithm

Require: Training dataset $\mathcal{T}_o$, current observation O_i

1: $\mathcal{T}_z \leftarrow \text{MM}(\mathcal{T}_o)$
2: $z_i \leftarrow \text{MM}(O_i)$
3: $\mathcal{U}_i \leftarrow \text{SM}(O_i)$
4: $\mathcal{L}_e := \{\}$
5: **for each** $u \in \mathcal{U}_i$ **do**
6: $\quad z_n \leftarrow \text{LPM}(z_i, u)$
7: $\quad d_i \leftarrow \text{nearest}(\mathcal{T}_z, z_n)$
8: $\quad \mathcal{L}_e \leftarrow \text{add tuple}(u, d_i)$
9: $u_e \leftarrow \text{get action to furthest state}(\mathcal{L}_e)$

return u_e

update the MM, LPM, and SM models. Thirdly, we perform the targeted exploration phase. For each exploration step $i \in \{1, ..., n_e\}$, we get the current observation O_i, determine the most promising exploration action u_e using Algorithm 3 (line 6) and execute it (line 7) to reach the new observation O_{i+1}. The observed tuple $(O_i, O_{i+1}, (\rho = (1, u_e)))$ is added to the dataset $\overline{\mathcal{T}_o}$ (line 8). After the completion of the exploration, the LSR is built using the approach in [5] with neighborhood threshold ε (line 9). Finally, we add the shortcuts as in Algorithm 2 (line 10) and the final LSR is returned for planning.

Algorithm 4 Integration Algorithm

Require: Training dataset $\mathcal{T}_o$, search radius r, neighborhood threshold ε, number of explorations n_e

1: MM, LPM, SM $\leftarrow$ build models($\mathcal{T}_o$)
2: $\overline{\mathcal{T}}_o \leftarrow$ augment dataset($\mathcal{T}_o, r$) [Alg. 1]
3: MM, LPM, SM $\leftarrow$ update models($\overline{\mathcal{T}}_o$)
4: **for each** $i \in \{1, .., n_e\}$ **do**
5: $\quad O_i \leftarrow$ current observation
6: $\quad u_e \leftarrow$ get exploration action($\overline{\mathcal{T}_o}, O_i$) [Alg. 3]
7: $\quad O_{i+1} \leftarrow$ perform action(u_e)
8: $\quad \overline{\mathcal{T}_o} := \overline{\mathcal{T}_o} \cup \{(O_i, O_{i+1}, (\rho = (1, u_e)))\}$
9: $LSR \leftarrow$ build LSR($\overline{\mathcal{T}_o}, \varepsilon$) [5]
10: $LSR_{ace} \leftarrow$ build shortcuts(LSR, ε) [Alg. 2]

return LSR_{ace}

VI. Simulation results

To validate the proposed approach, we consider a simulated box stacking task, shown in Fig. 2 and referred to as *hard* stacking task in [6]. This setting allows to determine the true underlying state of each observation (exploited for evaluation purposes only) and therefore to automatically validate the effectiveness of each ACE component as well as of the entire ACE framework.

The box stacking task is composed of a 3×3 grid where four boxes can be stacked on top of each other. The underlying state is defined by the geometrical arrangement of the boxes, where each box is considered unique. The action specifics u is represented by the pick and place coordinates. In each observation, we induce different lighting conditions as well as $\approx 17\%$ random noise in the positioning of the boxes in each cell. The following rules apply: *i)* only one box can be moved at the time, *ii)* only one box can be placed in a single grid cell, *iii)* boxes cannot float, and *iv)* a box can only be picked if no other box is on top of it. Given the 3×3 grid and the above rules, the system exhibits exactly 288 possible underlying states, and $|\mathcal{U}| = 48$ possible actions.

A. Evaluation Criteria and Implementation Details

To evaluate the effectiveness of the ACE paradigm in scarce data settings, we randomly sub-sample 80%, 75%, 60%, 50%, 40%, and 30% of the original dataset $\mathcal{T}_{100}$ [6] consisting of 2500 pairs. We denote these sub-sampled dataset as $\mathcal{T}_{80}$, $\mathcal{T}_{75}$, $\mathcal{T}_{60}$, $\mathcal{T}_{50}$, $\mathcal{T}_{40}$, and $\mathcal{T}_{30}$, respectively. We compare the combined ACE-LSR with the ε-LSR in [5] as well as with the ablated versions of each component of ACE, namely A-LSR for the augmentation step, C-LSR for connection step, and E-LSR for the exploration step. We additionally implement: *i)* a baseline augmentation step, referred to as A_b-LSR, which generates similar pairs using the closest states in the latent space, and *ii)* a baseline exploration step, referred to as E_b-LSR, which is a random explorer that selects a random action from the set $\mathcal{U}$ of the system actions and tries to apply it to the given state observation. We omit the comparison of ACE-LSR to other existing methods which can be found in [6].

We score all frameworks by the planning performance on 1000 *novel* start and goal states randomly selected from a holdout dataset composed of 2500 observations. We report the percentage of correct transitions, and the percentage of cases where all plans are correct, and where at least one of the suggested plan is correct, denoted by *% trans.*, *% all*, and *% any*, respectively. Furthermore, to evaluate the augmentation component, we report the number of newly identified similar pairs, *# pairs*, and the percentage of correct pairs among them, *% pairs*, using the ground truth underlying states, The connection component is similarly scored by measuring the number of new edges built in the graph, *# edges*, as well as the percentage of correct edges, *% edges*. Finally, the exploration component is evaluated by performing $n_e = 500$ exploration steps from random initial states and defining the percentage of valid exploratory actions, *% explore*. For each score, we report mean and variance obtained with three different seeds for the MM model training.

The VAE modelling the MM is trained as in [6] with latent dimension 16. A DBSCAN-based [16] clustering algorithm is used for building the LSR. The hyperparameter ε is set to $\varepsilon = \mu_0 + w_\varepsilon \cdot \sigma_0$ as in [5] where μ_0 and σ_0 are the mean and standard deviation of the L_1 distances $\|z_1 - z_2\|_1$ among similar latent pairs $(z_1, z_2, a = 0)$, respectively, and w_ε is a scaling parameter. We perform a grid search for w_ε in the interval $[-0.65, -0.05]$ with step size 0.1. The LPM is a two-layer, 100 nodes MLP, while the Siamese network for the SM is a shallow two convolutional layer network with a latent space dimension 12 as in [17]. We train the Siamese network for 100 epochs and perform

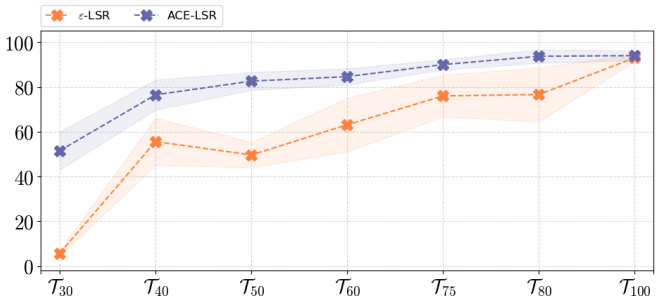

Fig. 5: Planning performance on the box stacking task in terms of *% any* using ε-LSR [5] (orange) and ACE-LSR (blue) trained on the subsampled datasets.

HDBSCAN [18] clustering in the latent space $\mathcal{Z}'$ of the model. We set search radius $r = \mu_0$ in Algorithm 1 since similar states should generally fall at a distance equal to the mean of similar states in the training dataset. While performing exploration, we additionally remove the action obtained by reversing the last action from the set of possible actions and apply a reset of the system state each time an invalid action is undertaken.

B. Evaluation Results

Figure 5 shows the planning performance in terms of *% any* score across the considered subsampled datasets when the proposed ACE paradigm is used (blue) and not (orange). Cross marks denote mean values, while the transparency represents the variance. We can observe that ACE-LSR boosts the planning performance compared to the ε-LSR [5] for each subsampled dataset and is particularly essential in case of very scarce datasets, e.g. $\mathcal{T}_{30}$-$\mathcal{T}_{50}$. For example, average improvements equal to $\approx 45,\ 21,\ 33\%$ are observed for $\mathcal{T}_{30}$,$\mathcal{T}_{40}$,$\mathcal{T}_{50}$, reaching $\approx 51.5, 76.5, 82.6\%$, respectively. Obviously, the improvement is much more significant with scarce datasets, while the performance is almost saturated with $\mathcal{T}_{100}$, reaching 93% and 94% with ε-LSR and ACE-LSR, respectively.

Table I shows the results of the ablation study for the components of the ACE paradigm compared with the ε-LSR. We report the complete scoring described in Sec. VI-A obtained using $\mathcal{T}_{50}$ which consists of half the data used in [5]. We observe that data scarcity leads to *unsatisfactory* planning performance of the ε-LSR, reaching only average *% any* score of 49.6% with *% trans* equal to 60.5%. No improvement but rather a decrease of performance is recorded with the baseline augmentation step, i.e., with A_b-LSR (row 2). This builds 1654 new similar pairs among which $\approx 95\%$ are correct. However, these new pairs deteriorate the structure of the latent space, resulting in *% any* equal to 33.1% only with a decrease of $\approx 16\%$. This suggests that simply adding new correct pairs do not necessarily induce improved performance if they are not carefully selected. In contrast, our augmentation algorithm A-LSR (row 3) produces only 15 new similar pairs on average that are 100% correct, thus boosting the planning performance in terms of *% any* to average 56.8%. Our connection algorithm in C-LSR (row 4).

Fig. 6: Start and five goal configurations of the T-shirt.

builds ≈ 479 new shortcuts that are $\approx 95\%$ correct. These yield to much higher planning scores, reaching average 87.7% and 63% for *% trans.* and *% any*, respectively. Concerning the exploration phase, only 7.8% of the moves (*% explore* score) attempted by the random explorer in E_b-LSR (row 5) are correct. This results in ≈ 2.5 percentage point enhancement of the planning performance in terms of *% any* compared to ε-LSR. On the other hand, a substantial improvement in planning performance is recorded when employing our exploration algorithm E-LSR (row 6). More specifically, $\approx 97.7\%$ of the $n_e = 500$ exploration moves are found to be valid, resulting in average 80.7% and 65.7% for *% trans.* and *% any* scores, respectively. This result suggests the effectiveness of the proposed SM models for proposing exploration actions, which are found almost always to be correct. Examples of suggested actions by the SM for the stacking task are reported in Fig. 4-left. Finally, the combined ACE-LSR approach (row 7) significantly outperforms all of the above mentioned frameworks, leading to an improvement $> 30\%$ in terms of *% trans.,% all,% any* compared to the ε-LSR and reaching a final *% any* performance of 82.6%.

VII. Experimental Results

To further validate the effectiveness of the ACE paradigm, we perform a real world T-Shirt folding task as in [5]. In this task, the goal is to generate and execute visual action plans from a start configuration to five different goal configurations, shown in Fig. 6.

Let $\mathcal{F}_{100}$ be the training dataset used in [5] containing a total of 1150 pairs, and let $\mathcal{F}_{50} \subset \mathcal{F}_{100}$ be a scarce dataset consisting of 50% randomly subsampled pairs. We use the same set of parameters and architectures as in Sec. VI-A unless otherwise specified. Since the action specifics u is composed of *pixel* position pick and place coordinates, the action space is much larger as in the simulation task. In order for the SM to be able to suggest meaningful actions, we discretize the action space into bins and use the mean actions of each bin. We choose a bin size of $\approx 15\%$ of the image space which results in 107 unique actions. Furthermore, we group the observations only based on the similarity of the pick action as this enables more flexible exploration. Examples of suggested actions for the T-shirt folding task are shown in Fig. 4-right. The SM, trained for 200 epochs, is then used in the augmentation step obtaining 13 new similar pairs (Algorithm 1). When applying the connection algorithm using the SM and LPM-R we obtain 55 novel edges in the graph. In order to obtain more novel connections, we increase the LPM-R reliability check by a factor of 1.5 since both the scarcity and diversity of the actions make a reliable prediction more challenging. Lastly, we execute Algorithm 3

Framework	*# pairs*	*% pairs*	*# edges*	*% edges*	*% explore*	*% trans.*	*% all*	*% any*
ε-LSR [5]	–	–	–	–	–	60.5 ± 4.4	49.1 ± 5.9	49.6 ± 5.7
A_b-LSR	$\mathbf{1654 \pm 9.0}$	95.18	–	–	–	58.8 ± 5.7	32.3 ± 19.2	33.1 ± 19.2
A-LSR	15 ± 11.0	**100**	–	–	–	64.0 ± 13.5	56.5 ± 12.1	56.8 ± 12.3
C-LSR	–	–	$\mathbf{479.2 \pm 52.1}$	95.1 ± 0.4	–	87.7 ± 0.6	57.1 ± 9.4	63.0 ± 7.9
E_b-LSR	–	–	–	–	7.8 ± 0.0	65.9 ± 0.2	51.4 ± 5.7	52.3 ± 5.3
E-LSR	–	–	–	–	$\mathbf{97.7 \pm 0.4}$	80.7 ± 2.0	62.2 ± 3.0	65.7 ± 3.1
ACE-LSR	15 ± 11.0	**100**	401.0 ± 15.3	$\mathbf{96.6 \pm 1.0}$	97.1 ± 1.6	$\mathbf{93.1 \pm 1.9}$	$\mathbf{79.2 \pm 4.2}$	$\mathbf{82.6 \pm 4.0}$

TABLE I: Evaluation results obtained on the box stacking task with $\mathcal{T}_{50}$ using ε-LSR as well as its combination with the baseline augmentation and exploration methods, the individual components of ACE paradigm and all the ACE components. The symbol – denotes that the respective score is not relevant to the framework. See Sec. VI-A for details. Best results in bold.

Framework	fold 1	fold 2	fold 3	fold 4	fold 5
ε-LSR	$0/5$	$\mathbf{5/5}$	$0/5$	$\mathbf{5/5}$	$1/5$
ACE-LSR	$\mathbf{4/5}$	$\mathbf{5/5}$	$\mathbf{3/5}$	$4/5$	$\mathbf{4/5}$

TABLE II: System performance results on the T-shirt folding task with $\mathcal{F}_{50}$ for ε-LSR and ACE-LSR on five different folds, each repeated five times. Best results in bold.

for $n_e = 20$ exploration steps. Note that exploration in the folding task is much less constrained, and therefore some explorations can lead to completely novel folding sequences not observed in the collected training dataset $\mathcal{T}_o$. Including the newly obtained action pairs yields the final ACE-LSR (built with $w_\varepsilon = 1$) that we compare with ε-LSR ($w_\varepsilon = 1.4$) trained on $\mathcal{F}_{50}$ in Tab. I. We repeat each fold five times and report the number of successful trials when the entire fold is performed successfully. The execution videos as well as the exploration can be seen on the project website[1].

The ACE-LSR outperforms the ε-LSR in all folds except for fold 4, and reaches a total system success rate over all five folds of 80%, matching the performance reported in [5] using only half the training data. We observe that the ε-LSR does not have enough data to distinguish fold 1 from fold 2 as it always performs fold 2 regardless of the fold goal state. On the contrary, ACE-LSR is able to successfully distinguish them and execute the correct fold most of the times. Furthermore, the ε-LSR is not able to reliably execute fold 5 as it is missing the final step to complete it, while ACE-LSR is able to perform it in $4/5$ cases.

VIII. Conclusions

In this work, we presented the ACE paradigm that addresses data scarcity problem for visual action planning. We built upon the Latent Space Roadmap framework and introduced *i)* a novel Suggestion Model (SM), that given an observation, suggests possible actions in that state, and *ii)* a Latent Prediction Model (LPM) that, given a latent state and an action, predicts the next latent state. Combining these modules, we **A**ugmented the dataset to identify new similar pairs for training, identified new valid edges in the LSR to increase its **C**onnectivity, and **E**xplored the latent space efficiently to reach potential undiscovered states. We validated the ACE paradigm on a simulated box stacking task and a real-world T-shirt folding task on several levels of data scarcity. As future work, we aim to extend this paradigm to different contexts, such as RL.

[1] `https://visual-action-planning.github.io/ace/`

References

[1] A. I. Károly, P. Galambos, J. Kuti, and I. J. Rudas, "Deep learning in robotics: Survey on model structures and training strategies," *IEEE Trans. Syst. Man Cybern.: Syst.*, vol. 51, no. 1, pp. 266–279, 2021.

[2] N. Sünderhauf, O. Brock, W. Scheirer, R. Hadsell, D. Fox, J. Leitner, B. Upcroft, P. Abbeel, W. Burgard, M. Milford, *et al.*, "The limits and potentials of deep learning for robotics," *Int. J. Robot. Res.*, vol. 37, no. 4-5, pp. 405–420, 2018.

[3] A. Wang, T. Kurutach, P. Abbeel, and A. Tamar, "Learning robotic manipulation through visual planning and acting," in *Robotics: Science and Systems*, 2019.

[4] K. Liu, T. Kurutach, C. Tung, P. Abbeel, and A. Tamar, "Hallucinative topological memory for zero-shot visual planning," in *Int. Conf. Mach. Learn.*, pp. 6259–6270, 2020.

[5] M. Lippi, P. Poklukar, M. C. Welle, A. Varava, H. Yin, A. Marino, and D. Kragic, "Latent space roadmap for visual action planning of deformable and rigid object manipulation," in *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, 2020.

[6] M. Lippi, P. Poklukar, M. C. Welle, A. Varava, H. Yin, A. Marino, and D. Kragic, "Enabling visual action planning for object manipulation through latent space roadmap," *arXiv preprint arXiv:2103.02554*, 2021.

[7] C. Finn and S. Levine, "Deep visual foresight for planning robot motion," in *IEEE Int. Conf. Robot. Autom.*, pp. 2786–2793, 2017.

[8] R. Hoque, D. Seita, A. Balakrishna, A. Ganapathi, A. Tanwani, N. Jamali, K. Yamane, S. Iba, and K. Goldberg, "VisuoSpatial Foresight for Multi-Step, Multi-Task Fabric Manipulation," in *Robotics: Science and Systems*, 2020.

[9] R. Hoque, D. Seita, A. Balakrishna, A. Ganapathi, A. K. Tanwani, N. Jamali, K. Yamane, S. Iba, and K. Goldberg, "Visuospatial foresight for physical sequential fabric manipulation," *Auton. Robots*, pp. 1–25, 2021.

[10] B. Ichter and M. Pavone, "Robot Motion Planning in Learned Latent Spaces," *IEEE Robot. Autom. Lett.*, vol. 4, no. 3, pp. 2407–2414, 2019.

[11] N. Savinov, A. Dosovitskiy, and V. Koltun, "Semi-parametric topological memory for navigation," in *Int. Conf. Learn. Represent.*, 2018.

[12] W. Yan, A. Vangipuram, P. Abbeel, and L. Pinto, "Learning predictive representations for deformable objects using contrastive estimation," *Conf. Robot Learn.*, 2020.

[13] K. Pertsch, O. Rybkin, F. Ebert, C. Finn, D. Jayaraman, and S. Levine, "Long-horizon visual planning with goal-conditioned hierarchical predictors," in *Adv. Neural Inf. Process. Syst.*, 2020.

[14] R. Rafailov, T. Yu, A. Rajeswaran, and C. Finn, "Offline reinforcement learning from images with latent space models," in *Learning for Dynamics and Control*, pp. 1154–1168, PMLR, 2021.

[15] T. Haarnoja, K. Hartikainen, P. Abbeel, and S. Levine, "Latent space policies for hierarchical reinforcement learning," in *Int. Conf. Mach. Learn.* (J. Dy and A. Krause, eds.), vol. 80, pp. 1851–1860, 2018.

[16] M. Ester, H.-P. Kriegel, J. Sander, X. Xu, *et al.*, "A density-based algorithm for discovering clusters in large spatial databases with noise.," in *Kdd*, vol. 96, pp. 226–231, 1996.

[17] C. Chamzas, M. Lippi, M. C. Welle, A. Varava, L. E. Kavraki, and D. Kragic, "Comparing reconstruction-and contrastive-based models for visual task planning," *arXiv preprint arXiv:2109.06737*, 2021.

[18] L. McInnes, J. Healy, and S. Astels, "HDBSCAN: Hierarchical density based clustering," *J. Open Source Software*, vol. 2, no. 11, p. 205, 2017.