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

Martina Lippi*1, Michael C. Welle*2, Petra Poklukar2, Alessandro Marino3, Danica Kragic2

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

Given a start observation of the system, the goal of visual action planning [1] 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. This aspect is crucial in applications where the state of the system cannot be easily described analytically, as for instance in the manipulation of deformable objects like wires in manufacturing settings, clothes in fashion industries or food in agricultural setups as in the European Project CANOPIES. The supporting visual plan additionally improves the interpretability of these methods [1], [2].

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 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 upon our Latent Space Roadmap (LSR) framework [3], [4] that learns a low-dimensional latent space from input images and builds a graph, in this latent space, that is used to perform planning. We tackle data scarcity by introducing the 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:

- 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. This is used in conjunction with a simple neural network that predicts the next latent state given the current state and desired action.
- We thoroughly analyze 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 [1] visual foresight plans for deforming a rope into desired configurations are generated with Context Conditional Causal InfoGANs. The learned rope inverse dynamics is then considered to reach the configurations in the generated plan. In [5] 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, the work in [6] 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 [7] where the main steps of the framework are improved.

To reduce the complexity of planning in the image space, low-dimensional latent spaces have been explored in several studies, e.g., [8]-[9]. A framework for global search in latent space is presented in [8], 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 [10] 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 [11] 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 Reinforcement Learning (RL) settings, like for example in [12] for model-based offline RL and in [13] for hierarchical RL. The combination of RL with graph structures in the latent space is explored in [9], where a node is created for each encoded observation. Building on [9], temporal closeness of the consecutive observations in the trajectories is also exploited in [2]. 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 for our framework and formalize the problem of visual action planning with data scarcity.

A. Dataset structure

Let $O$ be the space of all possible observations, i.e., images, of the system states. We consider a training dataset $T_o$ consisting of $q$ tuples $(O_1, O_2, \rho)$, where $O_1$ is an observation of the start state, $O_2$ an observation of the successor state, and $\rho$ 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 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 arrangement 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 $U$ 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 $P_v$ that lead the system from a given start $O_s \in O$ to a goal observation $O_g \in O$, i.e., such that $P_o = \{O_i = O_1, O_2, ..., O_N = O_g\}$ and $P_u = \{u_1, u_1, ..., u_{N-1}\}$, where $u_n \in U$ 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 $Z$ encoding $O$ that aims to capture the underlying states of the system.

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

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

Given these functions, one way to realize visual action planning is to map the start and goal observations in the latent space, obtaining $z_s = \xi(O_s)$, $z_g = \xi(O_g)$, and then perform planning directly in $Z$ by exploiting the latent dynamic function $f$. This leads to the definition of an action plan $P_u$ with corresponding latent plan $P_z = \{z_s = z_1, z_2, ..., z_N = z_g\}$, based on which the visual plan $P_o$ is generated through the observation generator $\omega$. Note that in practice the functions $\xi$, $\omega$ and $f$ are unknown and need to be approximated. The quality of these approximations depends on the available amount of observations $T_o$ of the system.
C. Problem Statement

When $T_o$ 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 $T_o$ as well as a start $O_s \in O$ and a goal observation $O_g \in O$, 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 $\eta : O \rightarrow U$ provides a subset $\eta(O_i) = U_i \subseteq U$ of actions that can be applied from an observation $O_i \in O$.

Given the suggestion function $\eta$ and latent dynamic function $f$, we define similar states as follows, in line with bisimulation theory [14].

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

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

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 [3], [4] by introducing the LSR framework, which we briefly recall in the following. The basic idea of this framework is to perform planning in the low dimensional latent space $Z$ 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 $T_o$ described in Section III-A into the latent space $Z$ to obtain a set of covered states $T_z = \{z_1, ..., z_q\} \subset Z$, i.e., $T_z = \xi(T_o)$, for which we make the following assumption as in [3], [4].

Assumption 1: Given a covered state $z \in T_z$, 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 $\varepsilon$-neighbourhoods of the covered states $z \in T_z$ as covered subspace

$$Z_{sys} = \bigcup_{z \in T_z} N_\varepsilon(z) \subset Z,$$

which can be rewritten as the union of $m$ path-connected components [4] called covered regions and denoted by $\{Z^i_{sys}\}_{i=1}^m$. 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 covered transition function $f_{sys}^i : Z_{sys} \times U \rightarrow Z_{sys}^i$ maps a point $z' \in Z_{sys}$ to a point $z'' \in Z_{sys}^i$ when applying an action $u \in U$, with $i, j \in \{1, 2, ..., m\}$ and $i \neq j$. Given $Z_{sys}$ and the covered transition functions $f_{sys}^i$, we then define the Latent Space Roadmap:

Definition 5: A Latent Space Roadmap is a directed graph $LSR = (V_{LSR}, E_{LSR})$ where each vertex $v_i \in V_{LSR} \subset Z_{sys}$ for $i \in \{1, ..., m\}$ is a representative of the covered region $Z_{sys}^i \subset Z_{sys}$, and each edge $e_{ij} = (v_i, v_j) \in E_{LSR}$ represents a covered transition function $f_{sys}^i j$ between the corresponding covered regions $Z_{sys}^i$ and $Z_{sys}^j$ for $i \neq j$.

Two main modules compose the LSR framework. First, a Mapping Module (MM) implements the mapping function $\xi$ and observation generator $\omega$ 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 $Z$ to approximate the covered regions $Z_{sys}^i$. Each obtained cluster is associated with a node in the LSR and edges among them are created using action pairs in the training dataset $T_o$. In this process, average action specificities are also endowed in the edges to retrieve the action plan $P_u$ (see the Action Averaging Baseline in [4] for details).

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 $T_o$, autonomously creating new similar pairs.
2) Create new unobserved connections in the latent space $Z$, increasing the set of covered transition functions $f_{sys}^i j$ and number of respective edges in the LSR.
3) Explore new regions of $Z$ in an efficient and guided manner, increasing the covered subspace $Z_{sys}$.

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

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

Regarding point 2), we use SM and LPM along with the covered subspace $Z_{sys}$ to identify previously unseen transitions $f_{sys}^i j$ that are possible to execute, called valid
transitions. These are added to the LSR in the form of new edges referred to as shortcuts.

Similarly, point 3) is realized by using the SM, that suggests the set of possible actions \( \mathcal{U}_i \) from the current state \( z_i \), and the LPM that predicts potential next states. Among the possible actions, the most promising one is chosen for exploration, 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), as detailed in Sec. VI. 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 considerations: 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 \( \eta \) with a Siamese network trained with a contrastive loss that encourages clustering of the states with respective colored dots. The modules only used offline are marked in grey, while the ones also used online are marked in green.

Fig. 3: Overview of the proposed architecture for visual action planning. The modules involved in the ACE phases 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.

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 \( \mathcal{T}_o \subset \mathcal{Z} \) (line 1) and initialize the augmented dataset \( \mathcal{T}_a \) (line 2). For each latent state \( z_i \in \mathcal{T}_o \), we check if a new similar pair can be identified.

When different action specifics are applied from \( O_1 \) and \( O_2 \) in \( \mathcal{T}_o \), i.e., if there exist \( (O_1, -1, \rho = (1, u_1)) \in \mathcal{T}_o \) and \( (O_2, -1, \rho = (1, u_2)) \in \mathcal{T}_o \) with \( u_1 \neq u_2 \). Note that training the Siamese network with the dataset \( \mathcal{T}_o^{SM} \) results in a latent space \( \mathcal{Z}' \) different from \( \mathcal{Z} \). The latent space \( \mathcal{Z}' \) is then clustered and each cluster \( C \subset \mathcal{Z}' \) is labeled with the set \( \mathcal{U}_c \subset \mathcal{U} \) containing all the actions that are executed starting from the states points in \( C \). We experimentally validate that the above procedure allows to achieve a good approximation of the suggestion function despite some possible erroneous similarity signals in \( \mathcal{T}_o^{SM} \).

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 \( C_i \) as visualized in Fig. 4.

### V. ACE Paradigm

In this section we present the individual components of our ACE paradigm and provide an overview of the full framework.

#### A. Augment

The proposed augmentation procedure builds new similar pairs based on the definition in 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 erroneously further apart in the latent space \( \mathcal{Z} \). This occurs in practice since the latent mapping \( \xi \) 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 \( \xi \) is re-learned to map the same underlying states close together.

Note that 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 \( \mathcal{T}_o \subset \mathcal{Z} \) (line 1) and initialize the augmented dataset \( \mathcal{T}_a \) (line 2). For each latent state \( z_i \in \mathcal{T}_o \), 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
Algorithm 1 Augmentation Algorithm

Require: Training dataset $T_0$, search radius $r$

1: $T_z \leftarrow \text{MM}(T_0)$
2: $T_o \leftarrow T_0$
3: for each $z_i \in T_z$ do
4: \hspace{1em} $U_i \leftarrow \text{SM}(O_i)$
5: \hspace{1em} $L_i \leftarrow$ search in radius($T_z, z_i, r$)
6: \hspace{1em} $L_i \leftarrow$ descent($L_i$)
7: \hspace{1em} found := False
8: \hspace{1em} for each $z_j \in L_i$ and not found do
9: \hspace{2em} $O_j \leftarrow$ get observation($T_o, j$)
10: \hspace{2em} $U_j \leftarrow \text{SM}(O_j)$
11: \hspace{2em} if $U_i \equiv U_j$ then
12: \hspace{3em} $Z^p_i, Z^p_j \leftarrow \text{LPM-R}(z_i, U_i), \text{LPM-R}(z_j, U_j)$
13: \hspace{3em} if $Z^p_i, Z^p_j$ nearest($T_z, Z^p_i$), nearest($T_z, Z^p_j$)
14: \hspace{3em} then $T_o := T_o \cup \{(O_i, O_j, a = 0)\}$
15: \hspace{1em} found := True
16: \hspace{1em} return $T_o$

states which are within the search radius $r$ (line 5), i.e., $L_i = \{z_j \in T_z \mid z_j \in N_\epsilon(z_i)\}$. This is followed by a descent sorting with respect to the distance of each $z_j \in L_i$ to $z_i$ (line 6). Note that we limit the search in a radius only for computational reasons. Since the latent space $Z$ already has a certain structure inferred from the non-augmented dataset $T_o$ 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 $z_j \in L_i$. 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 $U_j$ (line 10). If all the actions in the sets $U_i, U_j$ coincide, we obtain the sets of respective predicted states $Z^p_i, Z^p_j$ (line 12) by the LPM-R, which checks the reliability condition discussed in Sec. IV-A. Based on these sets, the sets $Z^p_i, Z^p_j$ consisting of closest covered latent states in $T_z$ with respect to $Z^p_i, Z^p_j$, respectively, are found. If $Z^p_i, Z^p_j$ coincide, a new similar pair $(O_i, O_j, a = 0)$ is added to the augmented dataset $T_o$, otherwise the next state $z_j \in L_i$ is analyzed.

B. Connect

A good connectivity of nodes is essential for the success of graph-based planning methods. 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 by using the SM and LPM modules. Note that it is crucial 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 $\epsilon$, we iterate over the states in the set of nodes $V_{LSR}$ of the LSR. For each state $z_i$ in $V_{LSR}$, we generate the respective observation $O_i$ through the observation generator $\omega$ of the MM (line 2) and obtain the set of potential actions $U_i$ by the SM. For each $u \in U_i$, LPM predicts the next state $z_n$ obtained from $z_i$ (line 5). If the predicted next state $z_n$ falls in the $\epsilon$-neighborhood of any other state $z_j \in V_{LSR}$ in the LSR with $i \neq j$, an edge between $z_i$ and $z_j$ is added in the edge set $E_{LSR}$ 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 $Z$ 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 $U_i$ 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 $Z$ at each exploration step.

The approach is summarized in Algorithm 3. Given the training dataset $T_z$ and the current state observation $O_i$, we first map both into $Z$ with the mapping function $\xi$ of the MM (lines 1-2). Then, we retrieve the set of potential actions $U_i$ from the current state $z_i$ through the SM. We initialize an empty auxiliary exploration list $L_e$. For each action $u \in U_i$, 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 $T_z$ (line 7). The tuple given by the action $u$ and distance $d_i$ is added to the exploration list $L_e$. Once all the actions in $U_i$ 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 $T_o$. The latent space is explored by executing Algorithm 3 $n_e$ times. Note that in case the LPM does not provide an accurate prediction for unseen states, this would only result in computing an imprecise distance $d_i$, and then choosing a less beneficial action to explore.
Algorithm 3 Exploration Algorithm

Require: Training dataset $T_o$, current observation $O_i$  
1: $T_z \leftarrow \text{MM}(T_o)$  
2: $z_i \leftarrow \ldots \text{V AE modelling the MM}$ is trained as in [4] with latent dimension $16$. A DBSCAN-based [15] clustering algorithm 
3: $U_i \leftarrow \text{SM}(O_i)$  
4: $L_c := \{\}$  
5: for each $u \in U_i$ do  
6: $z_u \leftarrow \text{LPM}(z_i, u)$  
7: $d_i \leftarrow \text{nearest}(T_z, z_u)$  
8: $L_c \leftarrow \text{add shortcut}(u, d_i)$  
9: $u_c \leftarrow \text{get action to furthest state}(L_c)$  
return $u_c$

D. LSR with ACE

In this section, we describe how the ACE components are combined within the LSR framework, summarized in Algorithm 4. Given an initial training dataset $T_o$, the hyper-parameters $r$ and $\varepsilon$ 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 $\overline{T}_o$ following Algorithm 1 with the search radius $r$ and use it to update the MM, LPM, and SM models. Thirdly, we performed the targeted exploration phase. For each exploration step $i \in \{1, \ldots, n_e\}$, 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 $\overline{T}_o$ (line 8). After completing the exploration, the LSR is built using the approach in [3] with neighborhood threshold $\varepsilon$ (line 9). Finally, we add the shortcuts as in Algorithm 2 (line 10) and the final LSR is returned. In case the system performance after executing Algorithm 4 is not satisfactory, this can be repeated multiple times to further improve the results.

Algorithm 4 Integration Algorithm

Require: Training dataset $T_o$, search radius $r$, neighborhood threshold $\varepsilon$, number of explorations $n_e$  
1: MM, LPM, SM $\leftarrow$ build models($T_o$)  
2: $\overline{T}_o \leftarrow$ augment dataset($T_o, r$) [Alg. 1]  
3: MM, LPM, SM $\leftarrow$ update models($\overline{T}_o$)  
4: for each $i \in \{1, \ldots, n_e\}$ do  
5: $O_i \leftarrow$ current observation  
6: $u_e \leftarrow$ get exploration action($\overline{T}_o, O_i$) [Alg. 3]  
7: $O_{i+1} \leftarrow$ perform action($u_e$)  
8: $T_o := T_o \cup \{(O_i, O_{i+1}, (\rho = (1, u_e)))\}$  
9: $\text{LSR} \leftarrow \text{build LSR}(T_o, \varepsilon)$ [3]  
10: $\text{LSR}_\text{ace} \leftarrow \text{build shortcuts}(\text{LSR}, \varepsilon)$ [Alg. 2]  
return $\text{LSR}_\text{ace}$

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 [4]. 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 $3 \times 3$ 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 $3 \times 3$ grid and the above rules, the system exhibits exactly 288 possible underlying states, and $|U| = 48$ 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 $T_{100}$ [4] consisting of 2500 pairs. We denote these sub-sampled dataset as $T_{800}$, $T_{750}$, $T_{600}$, $T_{500}$, $T_{400}$, and $T_{300}$, respectively. We compare the combined ACE-LSR with the $\varepsilon$-LSR in [3] 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 $A_0$-LSR, which generates similar pairs using the closest states in the latent space, and ii) a baseline exploration step, referred to as $E_0$-LSR, which is a random explorer selecting a random action from the set $U$ of the system actions and trying to apply it to the given state observation. For the sake of space, we omit the comparison of ACE-LSR to other existing methods, but this can be found for the general LSR framework in [4].

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 [4] with latent dimension 16. A DBSCAN-based [15] clustering algorithm
Fig. 5: Planning performance on the box stacking task in terms of % any using \( \varepsilon \)-LSR [3] (orange) and ACE-LSR (blue) trained on the subsampled datasets.

is used for building the LSR. The hyperparameter \( \varepsilon \) is set to \( \varepsilon = \mu_0 + w_x \cdot \sigma_0 \) as in [3] where \( \mu_0 \) and \( \sigma_0 \) 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 \( w_x \) is a scaling parameter. We perform a grid search for \( w_x \) in the interval \([-0.05, -0.05]\) 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 [16]. We train the Siamese network for 100 epochs and perform DBSCAN [17] clustering in the latent space \( \mathbb{Z}' \) 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 \( \varepsilon \)-LSR [3] 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 \( \varepsilon \)-LSR and ACE-LSR, respectively.

Table I shows the results of the ablation study for the components of the ACE paradigm compared with the \( \varepsilon \)-LSR. We report the complete score described in Sec. VI-A obtained using \( \mathcal{T}_{50} \) which consists of half the data used in [3]. We observe that data scarcity leads to unsatisfactory planning performance of the \( \varepsilon \)-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 A0-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 does 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), builds \( \approx 479 \) 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 baseline random explorer in E\( \mathcal{R} \)-LSR (row 5) are correct. This results in \( \approx 2.5 \% \) point enhancement of the planning performance in terms of % any compared to \( \varepsilon \)-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 \( \varepsilon \)-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 [3]. 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 [3] 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.
TABLE I: Evaluation results obtained on the box stacking task with $\mathcal{T}_{50}$ using $\varepsilon$-LSR and all the ACE components. The symbol $\pm$ denotes that the respective score is not relevant to the framework. See Sec. VI-A for details. Best results in bold.

| Framework | # pairs | % pairs | # edges | % edges | % explore | % trans. | % all | % any |
|-----------|---------|---------|---------|---------|-----------|----------|-------|-------|
| $\varepsilon$-LSR [3] | - | - | - | - | - | 60.5 ± 4.4 | 49.1 ± 5.9 | 49.6 ± 5.7 |
| A$_0$-LSR | 1654 ± 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 | - | - | 479.2 ± 52.1 | 95.1 ± 0.4 | - | 87.7 ± 0.6 | 57.1 ± 9.4 | 63.0 ± 7.9 |
| E$_0$-LSR | - | - | - | - | 7.8 ± 0.0 | 65.9 ± 0.2 | 51.4 ± 5.7 | 52.3 ± 5.3 |
| E-LSR | - | - | - | - | - | 97.7 ± 0.4 | 80.7 ± 2.0 | 62.2 ± 3.0 | 65.7 ± 3.1 |
| ACE-LSR | 15 ± 11.0 | 100 | 401.0 ± 15.3 | 96.6 ± 1.0 | 97.1 ± 1.6 | 93.1 ± 1.9 | 79.2 ± 4.2 | 82.6 ± 4.0 |

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

| Framework | fold 1 | fold 2 | fold 3 | fold 4 | fold 5 |
|-----------|--------|--------|--------|--------|--------|
| $\varepsilon$-LSR | 0/5 | 5/5 | 0/5 | 5/5 | 1/5 |
| ACE-LSR | 4/5 | 5/5 | 3/5 | 4/5 | 4/5 |

REFERENCES

[1] A. Wang, T. Kurutach, P. Abbeel, and A. Tamar, “Learning robotic manipulation through visual planning and acting,” in Robotics: Science and Systems, 2019.
[2] K. Liu, T. Kurutach, C. Tung, P. Abbeel, and A. Tamar, “Hallucinative topological memory for zero-shot visual planning,” in Int. Conf. Mach. Learn., 2020, pp. 6249-6270.
[3] 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.
[4] 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,” IEEE Trans. Robot., pp. 1–19, 2022.
[5] C. Finn and S. Levine, “Deep visual foresight for planning robot motion,” in IEEE Int. Conf. Robot. Autom., 2017, pp. 2786-2793.
[6] 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.
[7] 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.
[8] B. Ichter and M. Pavone, “Robot Motion Planning in Learned Latent Spaces,” IEEE Robot. Autom. Lett., vol. 4, no. 3, pp. 2407–2414, 2019.
[9] N. Savinov, A. Dosovitskiy, and V. Koltun, “Semi-parametric topological memory for navigation,” in Int. Conf. Learn. Represent., 2018.
[10] W. Yan, A. Vangipuram, P. Abbeel, and L. Pinto, “Learning predictive representations for deformable objects using contrastive estimation,” Conf. Robot. Learn., 2020.
[11] 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.
[12] R. Raffajew, T. Yu, A. Rajeswaran, and C. Finn, “Offline reinforcement learning from images with latent space models,” in Learning for Dynamics and Control, PMLR, 2021, pp. 1154–1168.
[13] 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, 2018, pp. 1851–1860.
[14] R. Givan, T. Dean, and M. Greig, “Equivalence notions and model minimization in markov decision processes,” Artificial Intelligence, vol. 147, no. 1, pp. 163–223, 2003, Planning with Uncertainty and Incomplete Information.
[15] 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, 1996, pp. 226–231.
[16] 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,” IEEE/RSJ Int. Conf. on Intelligent Robots and Systems, 2022.
[17] L. McMinn, J. Healy, and S. Astels, “IDBSCAN: Hierarchical density based clustering,” J. Open Source Software, vol. 2, no. 11, p. 205, 2017.

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 augmented the dataset to identify new similar pairs for training, identified new valid edges in the LSR to increase its Connectivity, and explored 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.

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