Enabling Visual Action Planning for Object Manipulation through Latent Space Roadmap

Martina Lippi*1,2, Petra Poklukar*1, Michael C. Welle*1, Anastasia Varava1, Hang Yin1, Alessandro Marino3, and Danica Kragic1

Abstract—We present a framework for visual action planning of complex manipulation tasks with high-dimensional state spaces, focusing on manipulation of deformable objects. We propose a Latent Space Roadmap (LSR) for task planning which is a graph-based structure globally capturing the system dynamics in a low-dimensional latent space. Our framework consists of three parts: (1) a Mapping Module (MM) that maps observations given in the form of images into a structured latent space extracting the respective states as well as generates observations from the latent states, (2) the LSR which builds and connects clusters containing similar states in order to find the latent plans between start and goal states extracted by MM, and (3) the Action Proposal Module that complements the latent plan found by the LSR with the corresponding actions. We present a thorough investigation of our framework on simulated box stacking and rope/box manipulation tasks, and a folding task executed on a real robot.

I. INTRODUCTION

In task and motion planning, it is common to assume that the geometry of the scene is given as input to the planner. In contrast, modern representation learning methods are able to automatically extract state representations directly from high-dimensional raw observations, such as images or video sequences [1]. This is especially useful in complex scenarios where explicit analytical modeling of states is challenging, such as in manipulation of highly deformable objects which is recently gaining increasing attention by the research community [2], [3].

Unsupervised State Representation Learning. Given raw observations, state representation learning is commonly performed in an unsupervised way using for example Autoencoders (AEs) [4] or Variational Autoencoders (VAEs) [5]. In these frameworks, two neural networks – an encoder and a decoder – are jointly trained to embed the input observation into a low-dimensional latent space, and to reconstruct it given a latent sample. The resulting latent space can be used as a low-dimensional representation of the state space, where the encoder acts as a map from a high-dimensional observation (an image) into the lower-dimensional state (a latent vector).

However, to be useful for planning, it is desirable to have a particular structure in the latent space: states that are similar should be encoded close to each other, while different states should be separated. Such information does not always coincide with the similarity of the respective images: two observations can be significantly different with respect to a pixel-wise metric due to task-irrelevant factors of variation such as changes in the lighting conditions and texture, while the underlying state of the system (e.g., the shape and the pose of the objects) may be identical. The opposite is also possible: two observations may be relatively close in the image space, because the respective change in the configuration of the scene does not significantly affect the pixel-wise metric, while from the task planning perspective the two states are fundamentally different.

Challenges of State Representation Learning for Planning. For planning, the system dynamics should also be captured in the latent space. We therefore identify three main challenges when modeling the state space representation for planning: i) it needs to be low dimensional, while containing the relevant information from high-dimensional observations; ii) it needs to properly reflect similarities between states; and iii) it needs to efficiently capture feasible transitions between states allowing to perform complex tasks such as deformable object manipulation.

In this work, we address i) by extracting the low-dimensional states directly from images of the scene through a Mapping Module (MM). For this, we deploy a VAE framework and compare it to AE. We address ii) by explicitly encouraging the encoder network to map the observations that correspond to different states further away from each other despite their visual similarity. This is done by providing a weak supervision signal: we record a small number of actions between observation pairs, and mark the observations as “same” or “different” depending on whether or not an action

Fig. 1: Examples of visual action plans for a stacking task (top), a rope/box manipulation task (middle) and a shirt folding task (bottom).
is needed to bring the system from one state to the successor one. We use this action information in an additional loss term to structure the latent space accordingly. Finally, we tackle iii) by building the Latent Space Roadmap (LSR), which is a graph-based structure in the latent space used to plan a manipulation sequence given a start and goal image of the scene. The nodes of this graph are associated with the system states, and the edges model the actions connecting them. For example, as shown in Fig. 1 these actions can correspond to moving a box or a rope, or folding a shirt. We identify the regions containing the same underlying states using hierarchical clustering [6] which accounts for differences in shapes and densities of these regions. The extracted clusters are then connected using the weak supervision signals. Finally, the action specifics are obtained from the Action Proposal Module (APM). In this way, we capture the global dynamics of the state space in a data-efficient manner without explicit state labels, which allows us to learn a state space representation for complex long-horizon tasks.

Contributions. Our contributions can be summarized as follows: i) we define the Latent Space Roadmap that enables to generate visual action plans based on weak supervision; ii) we introduce an augmented loss function with dynamic parameter to favourably structure the latent space; iii) we validate our framework on simulated box stacking tasks involving rigid objects, a combined rope and box manipulation task involving both deformable and rigid objects, and on a real-world T-shirt folding task involving deformable objects. Complete details can be found on the website [7].

This work is an extensively revised version of our earlier conference paper [2], where we first introduced the notion of Latent Space Roadmap. The main novelties of the present work with respect to [2] are:

1) Extension of the LSR building algorithm with an outer optimisation loop improving its performance;
2) New training approach for the MM with a dynamic adjustment of the key hyperparameter used in the additional loss term;
3) Large scale simulation campaigns investigating the effect of the additional loss term and hyperparameter choices;
4) Restructuring of the framework into three main components leading to a more modular setup;
5) Introduction of a more challenging box stacking task and a task involving manipulation of a rope and two boxes, enabling a thorough ablation study on all components of our framework;
6) Comparison with the state-of-the-art solutions in [8] and [9] on the simulation tasks as well as comparison of the improved framework with its predecessor [7] on the T-shirt folding task performed on a real robot;
7) Comparison with other potentially suitable clustering algorithms used to build the LSR;
8) Comparison of VAE and AE for the mapping module;
9) Comparison of different realizations of the APM.

https://visual-action-planning.github.io/lsr-v2/

II. RELATED WORK

The state representation, in terms of shape and poses of objects in the robot's workspace, is generally assumed to be known in task and motion planning and represents an input to the planner. As an example, robot and grasping knowledge was exploited in [10] to carry out sequential manipulation tasks, while semantic maps were designed in [11] to integrate spatial and semantic information and perform task planning. In addition, the notion of belief space, representing all possible distributions over the robot state space, was introduced in [12] to tackle partially observable control problems. Alternatively, sampling-based planning approaches, such as Rapidly exploring Random Tree (RRT) or Rapidly exploring Dense Tree (RDT) [13], can be adopted to probe the state space when an explicit representation of it is difficult to achieve.

However, traditional planning approaches, like the ones mentioned above, are generally prone to fail when high-dimensional states, such as images, are involved in the system [14]. For this reason, data-driven low-dimensional latent space representations for planning are gaining increasing attention, which, nonetheless, introduce problems to properly capture the global structure and dynamics of the system, as discussed in Sec. I. To tackle these, imitation learning can be leveraged as in [15], [16]. In particular, a latent space Universal Planning Network was designed in [15] to embed differentiable planning policies. The process is learned in an end-to-end fashion from imitation learning and gradient descent is used to find optimal trajectories. A controller based on random forests was built in [16] to perform cloth manipulation assuming the presence of optimal demonstrations provided by experts. Alternatively, a motion planning network with active learning procedure was developed in [17] to reduce the data for training and actively ask for expert demonstrations only when needed. In particular, the approach relies on an encoder network, that maps the robot surrounding environment in a low-dimensional state space, and on a prediction network, that is iteratively queried to provide states in the planned path on the basis of environment encoding and the robot initial and goal states.

In contrast to imitation learning methods, self-supervised learning approaches can be applied to mitigate the reduced generalization capabilities of the former at the expense of a costly data collection phase. Existing methods leverage physical simulators to acquire interaction data and then adapt skills from simulation to reality [18]. New simulation techniques have also opened up research on manipulating cloth-like objects involving multiple actions [19], [20]. However, these methods are often limited to handling objects of a simple shape, such as a square cloth, and simplify the interaction modeling with kinematic constraints [18], [19], [20]. Simulating complex dynamics of deformable objects/robot interaction remains an open problem. This was the motivation in [21] which learned folding directly on the real robotic platforms by meticulously designing convolutional observation features and a discrete action space, managing to generalize to objects of untrained sizes. Still, the scalability of collecting real interactive data undermines the applicability of self-supervised
learning approaches. This problem becomes even more severe when dealing with long-horizon task planning, where the range of possible variations in the state space is large. In this context, a framework for global search in latent space was designed in [22] which is based on three components: i) a latent state representation, ii) a network to approximate the latent space dynamics, and iii) a collision checking network. Motion planning is then performed directly in the latent space by an RRT-based algorithm. Embed-to-Control (E2C) [23] pioneers in learning a latent linear dynamical model for planning continuous actions. Variational inference is used to infer a latent representation and dynamical system parameters that can reconstruct a sequence of images. The extracted latent variables were shown to be well aligned to the underlying states in a visual inverted pendulum domain. In addition to estimating transition and observation models, [9] proposed a deep planning network which also learns a reward function in the latent space. Then, the latter is used to find viable trajectories resorting to a Model Predictive Control (MPC) approach. A comparison between our method and a baseline inspired by this approach can be found in Sec. [IX-C1] Reinforcement Learning (RL) using self play was employed in [24], where a VAE encodes trajectories into the latent space that is optimized to minimize the KL-divergence between proposed latent plans and those that have been encountered during self-play exploration. In contrast to using full plans, we optimized the structure of the latent space using pairs of states. The manipulation of a deformable rope from an initial start state to a desired goal state was analyzed in [25]. It builds upon [26] and [27] where 500 hours worth of data collection were used to learn the rope inverse dynamics, and then produced an understandable visual foresight plan containing the intermediate steps to deform the rope using a Context Conditional Causal InfoGAN (C3IGAN). Similarly, contrastive learning was used in [28] to learn a predictive model in the latent space, which is used to plan rope and cloth flattening actions. Goal-conditioning was then introduced in [29] to carry out long-horizon visual planning by reducing the search space and performing hierarchical optimization. Visual foresight was also realized in [30] where a video prediction model based on Long-Short Term Memory blocks was employed to predict stochastic pixel flow from frame to frame. Trained on video, action and state sequences, the model provides an RGB prediction of the scene that is then used to perform visual model predictive control. The data was collected using ten identical real world setups with different camera angles. In this work, instead of performing imitation learning from experts or exploring the latent space in a self-supervised manner, we leverage a weak supervision given by demonstrated actions in the dataset to capture the global structure of the state space and its dynamics in a data-efficient manner.

Finally, the idea of employing graphs as data representation has recently been investigated in several works. In [31], the authors introduced contrastive learning for structured world models. An end-to-end framework was presented that uses a Convolutional Neural Network (CNN) extractor to segment objects from the given scene observations and a Multi-Layer Perceptron encoder to produce their representations. A graph neural network (GNN) was then learned to model the relations and transitions given the representations of objects in the scene. Moreover, as graphs naturally decompose a long-horizon task into short-horizon tasks, authors of [32] improved RL performance on long-horizon tasks by building a graph whose nodes are the states contained in the replay buffer. Combining RL with the idea of connecting states in the latent space via a graph was proposed in Semi-Parametric Topological Memory (SPTM) framework [8], where an agent explores the environment and encodes observations into a latent space using a retrieval network. Each encoded observation forms a unique node in a memory graph built in the latent space. This graph is then used to plan an action sequence from a start to a goal observation using a locomotion network. As discussed in Sec. [IX-C1] where we compare our method with the SPTM framework, the latter is optimized for the continuous domain, having action/observation trajectories as input and building on the assumption that each observation is mapped to a unique latent code. Finally, the graph representation was also used in [33] for learning block stacking. The scene/object relations are expected to be captured by a GNN so as to boost the efficiency of curriculum learning and to generalize to stacking untrained number of blocks. In this work, we use graphs to capture the structure and the dynamics of the system, where nodes represent states of the system and edges the possible transitions between them.

### III. Problem Statement and Notation

| Variable | Meaning |
|----------|---------|
| $I$      | Space of observations, i.e., images |
| $U$      | Space of actions |
| $Z$      | Low-dimensional latent space |
| $P_i, P_u, P_z$ | Planned sequence of images, actions and latent states from assigned start and goal observations, respectively |
| $Z_{sys}^i$ | Covered region $i$ of the latent space |
| $Z_{sys}$  | Overall covered region of the latent space |
| $\rho$  | Specifics of the action that took place between two images $I_1$ and $I_2$ |
| $T_I, T_z$ | Datasets containing image tuples $(I_1, I_2, \rho)$ and their embeddings $(z_1, z_2, \rho)$, respectively |
| $\xi$  | Latent mapping function from $I$ to $Z$ |
| $\omega$ | Observation generator function from $Z$ to $I$ |
| $d_{in}$ | Minimum distance encouraged among action pairs in the latent space |
| $p$  | Metric $L_p$ |
| $\tau$  | Clustering threshold for LSR building |
| $c_{max}$ | Maximum number of connected components of the LSR |
| $N_{\varepsilon_z}(z)$ | The $\varepsilon_z$-neighbourhood of a covered state $z$ containing same covered states |
| $\varepsilon^i$ | $\varepsilon_z$ associated with all the states in the covered region $Z_{sys}^i$, i.e., $\varepsilon^i = \varepsilon_z \forall z \in Z_{sys}^i$ |

Table I: Main notations introduced in the paper.
The goal of visual action planning, also referred to as “visual planning and acting” in \cite{20}, can be formulated as follows: given start and goal images, generate a path as a sequence of images representing intermediate states and compute dynamically valid actions between them. We now formalize the problem and provide notation in Table \ref{tab:notation}.

Let \( I \) be the space of all possible observations of the system’s states represented as images with fixed resolution and let \( U \) be the set of possible control inputs or actions.

**Definition 1:** A visual action plan consists of a visual plan represented as a sequence of images \( P_1 = \{ I_{\text{start}} = I_0, I_1, ..., I_N = I_{\text{goal}} \} \) where \( I_{\text{start}}, I_{\text{goal}} \in I \) are images capturing the underlying states and goal states of the system, respectively, and an action plan represented as a sequence of actions \( P_u = \{ u_0, u_1, ..., u_{N-1} \} \) where \( u_n \in U \) generates a transition between consecutive states contained in the observations \( I_n \) and \( I_{n+1} \) for each \( n \in \{0, ..., N - 1\} \).

To retrieve the underlying states represented in the observations as well as to reduce the complexity of the problem, we map \( I \) into a lower-dimensional latent space \( Z \) such that each observation \( I_n \in I \) is encoded as a point \( z_n \in Z \) extracting the state of the system captured in the image \( I_n \). We call this map a latent mapping and denote it by \( \xi : I \rightarrow Z \). In order to generate visual plans, we additionally assume the existence of a mapping \( \omega : Z \rightarrow I \) called observation generator.

Let \( T_z = \{ I_1, ..., I_M \} \subset I \) be a finite set of input observations inducing a set of covered states \( T_z = \{ z_1, ..., z_M \} \subset Z \), i.e., \( T_z = \xi(T_I) \). In order to identify a set of unique covered states, we make the following assumption on \( T_z \).

**Assumption 1:** Let \( z \in T_z \) be a covered state. Then there exists \( \varepsilon_z > 0 \) such that any other state \( z' \) in the \( \varepsilon_z \)-neighborhood \( N_{\varepsilon_z}(z) \) of \( z \) can be considered as the same underlying state.

This allows both generating a valid visual action plan and taking into account the uncertainty induced by imprecisions in action execution. Let

\[
Z_{\text{sys}} = \bigcup_{z \in T_z} N_{\varepsilon_z}(z) \subset Z
\]

be the union of \( \varepsilon_z \)-neighbourhoods of the covered states \( z \in T_z \). Given \( Z_{\text{sys}} \), a visual plan can be computed in the latent space using a latent plan \( P_z = \{ z_{\text{start}} = z_0, z_1, ..., z_N = z_{\text{goal}} \} \), where \( z_n \in Z_{\text{sys}} \), which is then decoded with the observation generator \( \omega \) into a sequence of images.

To obtain a valid visual plan, we study the structure of the space \( Z_{\text{sys}} \) which in general is not path-connected. As we show in Fig. \[2\] on the right, linear interpolation between two states \( z_1 \) and \( z_2 \) in \( Z_{\text{sys}} \) may result in a path containing points from \( Z - Z_{\text{sys}} \) that do not correspond to covered states of the system and are therefore not guaranteed to be meaningful. To formalize this, we define an equivalence relation in \( Z_{\text{sys}} \)

\[
z \sim z' \iff z \text{ and } z' \text{ are path-connected in } Z_{\text{sys}},
\]

which induces a partition of the space \( Z_{\text{sys}} \) into \( m \) equivalence classes \( [z_1], ..., [z_m] \). Each equivalence class \( [z_i] \) represents a path-connected component of \( Z_{\text{sys}} \)

\[
Z_{\text{sys}}^i = \bigcup_{z \in [z_i]} N_{\varepsilon_z}(z) \subset Z_{\text{sys}}
\]
called covered region. To connect the covered regions, we define a set of transitions between them:

**Definition 2:** A transition function \( f^{i,j}_z : Z_{\text{sys}}^i \times U \rightarrow Z_{\text{sys}}^j \)

maps any point \( z \in Z_{\text{sys}}^i \) to an equivalence class representative \( z_j^i \in Z_{\text{sys}}^j \), where \( i, j \in \{1, 2, ..., m\} \) and \( i \neq j \).

Equivalence relation (2) and Assumption 1 imply that two distinct observations \( I_1 \) and \( I_2 \) which are mapped into the same covered region \( Z_{\text{sys}}^i \) contain the same underlying state of the system, and can be represented by the same equivalence class representative \( z^i \). Given a set of covered regions \( Z_{\text{sys}}^i \) in \( Z_{\text{sys}} \), and a set of transition functions connecting them we can approximate the global transitions of \( Z_{\text{sys}} \) as shown in Fig. \[2\] on the left. To this end, we define a Latent Space Roadmap (see Fig. \[2\] in the middle):

**Definition 3:** A Latent Space Roadmap is a directed graph \( \text{LSR} = (\mathcal{V}_{\text{LSR}}, \mathcal{E}_{\text{LSR}}) \) where each vertex \( v_i \in \mathcal{V}_{\text{LSR}} \subset Z_{\text{sys}} \) for \( i \in \{1, 2, ..., m\} \) is an equivalence class representative of the covered region \( Z_{\text{sys}}^i \subset Z_{\text{sys}} \), and an edge \( e_{i,j} = (v_i, v_j) \in \mathcal{E}_{\text{LSR}} \) represents a transition function \( f^{i,j}_z \) between the corresponding covered regions \( Z_{\text{sys}}^i \) and \( Z_{\text{sys}}^j \) for \( i \neq j \). Moreover, weakly connected components of an LSR are called graph-connected components.

**IV. METHODOLOGY**

We first present the structure of the training dataset and then provide an overview of the approach.

**A. Training Dataset**

We consider a training dataset \( T_I \) consisting of generic tuples of the form \((I_1, I_2, \rho)\) where \( I_1 \subset I \) is an image of the start state, \( I_2 \subset I \) an image of the successor state, and \( \rho \) a variable representing the action that took place between the
two observations. Here, an action is considered to be a single transformation that produces any consecutive state represented in \( I_2 \) different from the start state in \( I_1 \), i.e., \( \rho \) cannot be a composition of several transformations. On the contrary, we say that no action was performed if images \( I_1 \) and \( I_2 \) are observations of the same state, i.e., if \( \xi(I_1) \approx \xi(I_2) \) with respect to the equivalence relation \( (2) \). The variable \( \rho = (a, u) \) consists of a binary variable \( a \in \{0, 1\} \) indicating whether or not an action occurred as well as a variable \( u \) containing the task-dependent action-specific information. The latter, if available, is used to infer the transition functions \( f^{g,a} \). We call a tuple \( (I_1, I_2, \rho = (1, u)) \) an action pair and \( (I_1, I_2, \rho = (0, u)) \) a no-action pair. For instance, Fig. 4 shows an example of an action pair (top row) and a no-action pair (bottom row) for the folding task. In this case, the action specifics \( u \) contain the pick and place coordinates. To achieve the transition from the state captured by \( I_1 \) to the state captured by \( I_2 \), while the no-action pair images are different observations of the same underlying state of the system represented by slight perturbations of the sleeves. When the specifics of an action \( u \) are not needed, we omit them from the tuple notation and simply write \((I_1, I_2, a)\). By abuse of notation, we sometimes refer to an observation \( I \) contained in any of the training tuples as \( I \in \mathcal{T}_I \). Finally, we denote by \( \mathcal{T}_z \) the encoded training dataset \( \mathcal{T}_z \) consisting of latent tuples \((z_1, z_2, \rho)\) obtained from the input tuples \((I_1, I_2, \rho) \in \mathcal{T}_I \) by encoding the inputs \( I_1 \) and \( I_2 \) into the latent space \( \mathcal{Z}_{sys} \) with the latent mapping \( \xi \). The obtained states \( z_1, z_2 \in \mathcal{Z}_{sys} \) are called covered states.

Remark 1: The dataset \( \mathcal{T}_I \) is not required to contain all possible action pairs of the system but only a subset of them that sufficiently cover the dynamics, which makes our approach data efficient.

B. System Overview

Generation of visual action plans consists of three components visualized in Fig. 3:

- **Mapping Module (MM)** used to both extract a low-dimensional representation of a state represented by a given observation, and to generate an exemplary observation from a given latent state (Sec. V);
- **Latent Space Roadmap (LSR)** built in the low dimensional latent space and used to plan (Sec. VI);
- **Action Proposal Module (APM)** used to predict action specifics for executing a latent plan found by the LSR (Sec. VII).

The MM consists of the latent mapping \( \xi : \mathcal{I} \rightarrow \mathcal{Z} \) and the observation generator \( \omega : \mathcal{Z} \rightarrow \mathcal{I} \). To find a visual plan between a given start observation \( I_{\text{start}} \) and goal observation \( I_{\text{goal}} \), the latent mapping \( \xi \) first extracts the corresponding lower-dimensional representations \( z_{\text{start}} \) and \( z_{\text{goal}} \) of the underlying start and goal states, respectively. Ideally, \( \xi \) should exactly extract the underlying state of the system such that different observations containing the same state are mapped into the same latent point. In practice, however, the unknown true latent embedding \( \xi \) is approximated with a neural network which implies that different observations containing the same state could be mapped to different latent points. In order to perform planning in \( \mathcal{Z} \), we thus build the LSR which is a graph-based structure identifying the latent points belonging to the same underlying state and approximating the system dynamics. This enables finding the latent plans \( \mathcal{P}_z \) between the extracted states \( z_{\text{start}} \) and \( z_{\text{goal}} \). For the sake of interpretability, latent plans \( \mathcal{P}_z \) are decoded into visual plans \( \mathcal{P}_I \), consisting of a sequence of images, by the observation generator \( \omega \).

We complement the generated visual plan \( \mathcal{P}_I \) with the action plan \( \mathcal{P}_u \) produced by the APM, which proposes an action \( u_i \) that achieves the desired transition \( f_z^{i,i+1}(z_i, u_i) = z_{i+1} \) between each pair \((z_i, z_{i+1})\) of consecutive states in the latent plan \( \mathcal{P}_z \) found by the LSR.

The visual action plan produced by the three components can be executed by any suitable framework.
Remark 2: If open loop execution is not sufficient for the task, as for deformable object manipulation, a re-planning step can be added after every action. This implies that a new visual action plan is produced after the execution of each action until the goal is reached. A visualization of the re-planning procedure is shown in Fig. 12 on the T-shirt folding task presented in Sec. X.

Remark 3: Our method is able to generate a sequence of actions \( \{u_0, \ldots, u_{N-1}\} \) to reach a goal state in \( I_N \) from a given start state represented by \( I_0 \), even though the tuples in the input dataset \( T_i \) only contain single actions \( u \) that represent the weak supervision signals.

V. MAPPING MODULE (MM)

The mappings \( \xi : I \rightarrow Z \) and \( \omega : Z \rightarrow I \) as well as the low-dimensional space \( Z \) can be realized using any encoder-decoder based algorithms, for example VAEs, AEs or Generative Adversarial Networks (GANs) combined with an encoder network. The primary goal of MM is to find the best possible approximation \( \xi \) such that the structure of the extracted states in the latent space \( Z \) resembles the one corresponding to the unknown underlying system. The secondary goal of MM is to learn an observation generator \( \omega \) which enables visual interpretability of the latent plans. Since the quality of these depends on the structure of the latent space \( Z \), we leverage the action information contained in the binary variable \( a \) of the training tuples \( (I_1, I_2, a) \) to improve the quality of the latent space. We achieve this by introducing a contrastive loss term \( \ell_{\text{loss}} \) which can be easily added to the loss function of any algorithm used to model the MM.

More precisely, we introduce a general action term

\[
\ell_{\text{action}}(I_1, I_2) = \begin{cases} 
\max(0, d_m - \|z_1 - z_2\|_p) & \text{if } a = 1 \\
\|z_1 - z_2\|_p & \text{if } a = 0
\end{cases}
\]  

(4)

where \( z_1, z_2 \subset Z_{\text{sys}} \) are the latent encodings of the input observations \( I_1, I_2 \subset T_i \), respectively, \( d_m \) is a hyperparameter, and the subscript \( p \in \{1, 2, \infty\} \) denotes the metric \( L_p \). The action term \( \ell_{\text{action}} \) naturally imposes the formulation of the covered regions \( Z_{\text{sys}}^{i} \) in the latent space. On one hand, it encodes identical states contained in the no-action pairs close by. On the other hand, it encourages different states to be encoded in separate parts of the latent space via the hyperparameter \( d_m \).

As we experimentally show in Sec. XI-B, the choice of \( d_m \) has a substantial impact on the latent space structure. Therefore, we propose to learn its value dynamically during the training of the MM. In particular, \( d_m \) is increased until the separation of action and no-action pairs is achieved. Starting from 0 at the beginning of the training, we increase \( d_m \) by \( \Delta d_m \) every \( k \)th epoch as long as the maximum distance between no-action pairs is larger than the minimum distance between action pairs. The effect of dynamically increasing \( d_m \) is shown in Fig. 5 where we visualize the distance \( \|z_1 - z_2\|_1 \) between the latent encodings of every action training pair (in blue) and no-action training pair (in green) obtained at various epochs during training on a box stacking task. It can be clearly seen that the parameter \( d_m \) is increased as long as there is an intersection between action and no-action pairs.

Detailed investigation of this approach as well as its positive effects on the structure of the latent space are provided in Sec. XI-B. Note that the dynamic adaptation of the parameter \( d_m \) eliminates the need to predetermine its value as in our previous work [7].

We use a VAE such that its latent space represents the space \( Z \), while the encoder and decoder networks realize the mappings \( \xi \) and \( \omega \), respectively. We validate this choice in Sec. XI-B3 by comparing it to AE. In the following, we first provide a brief summary of the VAE framework [5], [35] and then show how the action term can be integrated into its training objective. Let \( I \subset T_I \) be an input image, and let \( z \) denote the unobserved latent variable with prior distribution \( p(z) \). The VAE model consists of encoder and decoder neural networks that are jointly optimized to represent the parameters of the approximate posterior distribution \( q(z|I) \) and the likelihood function \( p(I|z) \), respectively. In particular, VAE is trained to minimize

\[
\ell_{\text{vae}}(I) = E_{z \sim q(z|I)}[\log p(I|z)] + \beta \cdot D_{KL}(q(z|I)||p(z))
\]  

(5)

with respect to the parameters of the encoder and decoder neural networks. The first term influences the quality of the reconstructed samples, while the second term, called Kullback-Leibler (KL) divergence term, regulates the structure of the latent space. The trade-off between better reconstructions or a more structured latent space is controlled by the parameter \( \beta \), where using a \( \beta > 1 \) favors the latter [36], [37]. The action term (4) can be easily added to the VAE loss (5) as follows:

\[
\ell(I_1, I_2) = \frac{1}{2}(\ell_{\text{vae}}(I_1) + \ell_{\text{vae}}(I_2)) + \gamma \cdot \ell_{\text{action}}(I_1, I_2)
\]  

(6)

where \( I_1, I_2 \subset T_i \) and the parameter \( \gamma \) controls the influence of the distances among the latent encodings on the latent space structure. Note that the same procedure applies for integrating the action term (4) into any other framework that models the MM.

Fig. 5: An example showing histograms of distances \( \|z_1 - z_2\|_1 \) for latent action (in blue) and no-action pairs (in green) obtained at epochs 1, 5 and 50 during the training of VAE on the hard box stacking task (more details in Sec. XI). The figure shows the separation of the action and no-action distances induced by dynamically increasing the minimum distance \( d_m \) in \( \ell_{\text{action}} \).

VI. LATENT SPACE ROADMAP (LSR)

The Latent Space Roadmap, defined in Definition 2, is built in the latent space \( Z \) obtained from the MM. LSR is a graph that enables planning in the latent space which identifies sets of latent points associated with the same underlying state and viable transitions between them. Each node in the roadmap is associated with a covered region \( Z_{\text{sys}}^{i} \). Two nodes are connected by an edge if there exists an action pair.
that builds the LSR using the encoded training data and discussed in Sec. VI-A. It relies on a clustering algorithm L

transition 1

A. LSR Building

Require:

Algorithm 1 determine it using the procedure described in Sec. VI-B. (V

and visualized on the left of Fig. 6. Its set of vertices the edges that later induce the transition functions f

the latent action pairs. It serves as a look-up graph to preserve the latent states in Z

set of all the latent states in Zsys. It results in a reference graph built on the encoded dataset.

The LSR building procedure is summarized in Algorithm 1 and discussed in Sec. VI-A. It relies on a clustering algorithm that builds the LSR using the encoded training data Tz and a specified metric Lp as inputs. The input parameter τ is inherited from the clustering algorithm and we automatically determine it using the procedure described in Sec. VI-B.

Algorithm 1 LSR building

Require: Dataset Tz, metric Lp, clustering threshold τ

Phase 1

1: init graph G = (V, E) := (∅, ∅)
2: for each (z1, z2, a) ∈ Tz do
3:  V ← create nodes z1, z2
4:  if a = 1 then
5:  E ← create edge (z1, z2)

Phase 2

1: M ← Average-Agglomerative-Clustering(Tz, Lp)
2: W ← get-Disjoint–Clusters(M, τ)
3: Zsys ← (∅, ∅)
4: for each W ∈ W do
5:  εi ← get-Cluster-Epsilon(Wi)
6:  Zsys := ∪w∈W,wi:=Nεi,w(w)
7: Zsys := Zsys ∪ {Zsysi}

Phase 3

1: init graph LSR = (VLSR, ELSR) := (∅, ∅)
2: for each Zsysi ∈ Zsys do
3:  w∗i := 1/|Zsysi| ∑w∈Zsysi w
4:  z′sysi := argminz∈Zsysi ∥z − w∥p
5:  VLSR ← create node z′sysi
6: for each edge e = (v1, v2) ∈ E do
7:  find Zsys, Zsys containing v1, v2, respectively
8:  ELSR ← create edge (z′sys, Zsys)

return LSR

A. LSR Building

Algorithm 1 consists of three phases. In Phase 1 (lines 1.1–1.5), we build a reference graph G = (V, E) induced by Tz and visualized on the left of Fig. 6. Its set of vertices V is the set of all the latent states in Tz, while edges exist only among the latent action pairs. It serves as a look-up graph to preserve the edges that later induce the transition functions f1,i,2.

In Phase 2, Algorithm 1 identifies the covered regions Z′sys, Zsys. We achieve this by first clustering the training samples and then retrieving the covered regions from these clusters. We start by performing agglomerative clustering on the encoded dataset Tz (line 2.1). Agglomerative clustering is a hierarchical clustering scheme that starts from single nodes of the dataset and merges the closest nodes, according to a dissimilarity measure, step by step until only one node remains. It results in a stepwise dendrogram M, depicted in the middle part of Fig. 6 which is a tree structure visualizing the arrangement of data points in clusters with respect to the level of dissimilarity between them. We choose to measure this inter-cluster dissimilarity using the unweighted average distance between points in each cluster, a method also referred to as UPGMA [38]. More details about other possible clustering algorithms and dissimilarity measures are discussed in Sec. 1X-C4. Next, the dissimilarity value τ, referred to as clustering threshold, induces the set of disjoint clusters W, also called flat or partitional clusters [39], from the stepwise dendrogram M (line 2.2). Points in each cluster V

i are then assigned a uniform εi (line 2.5), i.e. the neighbourhood size from Assumption 1 of each point z ∈ V

i is εz = εi. We discuss the definition of the εi value at the end of this phase. The union of the εi-neighbourhoods of the points in V

i then forms the covered region Z′sysi (line 2.6). Illustrative examples of covered regions obtained from different values of τ are visualized on the right of Fig. 6 using various colors. The optimization of τ is discussed in Appendix C. The result of this phase is the set of the identified covered regions Zsys = {Zsysi} (line 2.7).

We propose to approximate εi as

εi = μi + σi

(7)

where μi and σi are the mean and the standard deviation of the distances ∥zi,j − z′i,j∥p among all the training pairs (zi,j, z′i,j) ∈ Tz belonging to the ith cluster. The approximation in (7) allows to take into account the cluster density such that denser clusters get lower εi. In contrast to our previous work [7], we now enable clusters to have different ε values. We validate the approximation in (7) in Secs. 1X-C5 and 1X-C6, where we analyze the covered regions identified by the LSR.

In Phase 3, we build the LSR = (VLSR, ELSR). We first compute the mean value w∗i of all the points in each cluster V

i (line 3.3). As the mean itself might not be contained in the corresponding path-connected component, we find the equivalence class representative z′sysi ∈ Z′sysi that is the closest (line 3.4). The found representative then defines a node v∗i ∈ VLSR representing the covered region Zsysi (line 3.5). Lastly, we use the set of edges E in the reference graph built in Phase 1 to infer the transitions f1,i,2,j between the covered regions identified in Phase 2. We create an edge in LSR if there exists an edge in E between two vertices in V that were allocated to different covered regions (lines 3.6–3.8). The right side of Fig. 6 shows the final LSRs, obtained with different values of the clustering threshold τ.

Note that, as in the case of the VAE (Sec. V), no action-specific information a is used in Algorithm 1 but solely the binary variable a indicating the occurrence of an action.

B. Optimization of LSR Clustering Threshold τ

The clustering threshold τ, introduced in Phase 2 of Algorithm 1, heavily influences the number and form of the resulting clusters. Since there is no inherent way to prefer one cluster configuration over another, finding its optimal value is a non-trivial problem and subject to ongoing research [40, 41, 42]. However, in our case, since the choice of τ subsequently influences the resulting LSR, we can leverage the information about the latter in order to optimize τ.

As illustrated in Fig. 6 the number of vertices and edges in LSRτ changes with the choice of τ. Moreover, the resulting LSRs can have different number of graph-connected components. For example, LSRτ1 in Fig. 6 has 2 graph-connected components, while LSRτ2 and LSRτ3 have only a
single one. Ideally, we want to obtain a graph that exhibits both good connectivity which best approximates the true underlying dynamics of the system, and has a limited number of graph-connected component. Intuitively, high number of edges increases the possibility to find latent paths from start to goal state. At the same time, this possibility is decreased when the graph is fragmented into several isolated components, which is why we are also interested in limiting the maximum number of graph-connected components.

While we cannot analyze the clusters themselves, we can evaluate information captured by the LSR that correlates with the performance of the task, i.e., we can assess a graph by the edges increases the possibility to find latent paths from start to goal state. At the same time, this possibility is decreased when the graph is fragmented into several isolated components, which is why we are also interested in limiting the maximum number of graph-connected components.

We again analyze the clusters and try to find a graph that achieves the transition $f^{i+1}$ from $z_i$ to $z_{i+1}$. Such functionality can be realized by any method that is rather robust to the choice of this parameter.

**Algorithm 2** LSR input optimization

```
Require: Dataset $T_z$, metric $L_p$, search interval $[\tau_{\text{min}}, \tau_{\text{max}}]$, $c_{\text{max}}$
1: $\tau \leftarrow \text{init}(\tau_{\text{min}}, \tau_{\text{max}})$
2: while $|\tau_{\text{max}} - \tau_{\text{min}}|$ not small enough do
3: $\text{LSR}_{\tau} \leftarrow \text{LSR-building}(T_z, L_p, \tau)$ [Algorithm 1]
4: $\psi \leftarrow \text{Evaluate}(\text{LSR}_{\tau})$ [Eq. (8)]
5: $\tau, \tau_{\text{min}}, \tau_{\text{max}} \leftarrow \text{Brent-update}(\psi)$ [43]
6: $\tau^* \leftarrow \tau$
return $\text{LSR}_{\tau^*}$
```

C. Visual plan generation

Given a start and goal observation, a trained VAE model and an LSR, the observations are first encoded by $\xi$ into the VAE's latent space $Z$ where their closest nodes in the LSR are found. Next, all shortest paths in the LSR between the identified nodes are retrieved. Finally, the equivalence class representatives of the nodes comprising each of the found shortest path compose the respective latent plan $P_z$, which is then decoded into the visual plan $P_I$ using $\omega$.

VII. ACTION PROPOSAL MODULE (APM)

The final component of our framework is the Action Proposal Module (APM) which is used to complement a latent plan, produced by the LSR, with an action plan that can be executed by a suitable framework. The APM allows to generate the action plans from the extracted low-dimensional state representations rather than high-dimensional observations. The action plan $P_z$ corresponding to a latent plan $P_z$ produced by the LSR is generated sequentially: given two distinct consecutive latent states $(z_i, z_{i+1})$ from $P_z$, APM predicts an action $u_i$ that achieves the transition $f^{i+1}(z_i, u_i) = z_{i+1}$. Such functionality can be realized by any method that is suitable to model the action specifics of the task at hand.

We model the action specifics with a neural network called Action Proposal Network (APN). We deploy a multi layer perceptron and train it in a supervised fashion on the latent action pairs obtained from the enlarged dataset $T_z$ as described below. We validate this choice in Sec. X-D, where we compare it to different alternatives that produce action plans rather than extracted low-dimensional states.
The training dataset \( \mathcal{T}_c \) for the APN is derived from \( \mathcal{T}_f \) but preprocessed with the VAE encoder representing the latent mapping \( \xi \). We encode each training action pair \((I_1, I_2, \rho = (1, u)) \in \mathcal{T}_f \) into \( \mathcal{Z} \) and obtain the parameters \( \mu_i, \sigma_i \) of the approximate posterior distributions \( q(z|I_i) \sim \mathcal{N}(\mu_i, \sigma_i) \), for \( i = 1, 2 \). We then sample \( 2S \) novel points \( z_1^s \sim q(z|I_1) \) and \( z_2^s \sim q(z|I_2) \) for \( s \in \{0, 1, \ldots, S\} \). This results in \( S+1 \) tuples \((\mu_1, \mu_2, \rho)\) and \((z_1^s, z_2^s, \rho)\), \( 0 \leq s \leq S \), where \( \rho = (1, u) \) was omitted from the notation for simplicity. The set of all such low-dimensional tuples forms the APN training dataset \( \mathcal{T}_c \).

Remark 4: It is worth remarking the two-fold benefit of this preprocessing step: not only does it reduce the dimensionality of the APN training data but also enables enlarging it with novel points by factor \( S+1 \). Note that the latter procedure is not possible with non-probabilistic realizations of \( \xi \).

VIII. ASSUMPTIONS, APPLICABILITY AND LIMITATIONS OF THE METHOD

In this section, we briefly overview our assumptions, describe tasks where our method is applicable, and discuss its limitations. In order for our method to successfully perform a given visual action planning task, the observations contained in the training dataset \( \mathcal{T}_f \) should induce the covered states (defined in Sec. [IV-A]) that are considered in the planning. Furthermore, it is required that sufficiently many transitions among them are observed such that the obtained LSR adequately approximates the true underlying system dynamics. For example, the training datasets \( \mathcal{T}_f \) in the box stacking tasks consist of \( 2500 \) pairs of states of the system instead of all (i.e., \( 41616 \)) possible combinations. On the other hand, if the system contains many feasible states, it can be challenging to collect a dataset \( \mathcal{T}_f \) that covers sufficiently many states and transitions between them. Even though the performance of the LSR would deteriorate with such incomplete dataset, we do not consider this as the limitation of the method itself as this can be mitigated with online learning approaches, e.g., [44], that dynamically adapt the LSR based on the interaction with the environment.

Given the assumptions on the format of the dataset \( \mathcal{T}_f \) introduced in Sec. [IV-A] our method is best applicable to visual action planning tasks where feasible states of the system are finite and can be distinguished in \( \mathcal{T}_f \) such that meaningful unambiguous actions to transition among them can be defined.

Therefore, our approach does not generalize well to entirely novel states of the system not contained in the training set. This is expected, as the model has no prior knowledge about the newly appeared state, such as, for example, an entirely new fold of a T-shirt or a new piece of garment. Such generalization could be achieved by integrating active learning approaches which is indeed an interesting future direction.

We emphasise that the proposed method is not limited by the dimensionality of the system’s states since that is reduced via MM.

IX. SIMULATION RESULTS

We experimentally evaluated our method on three different simulated tasks: two versions of a box stacking task (Fig. 7 left) and a combined rope and box manipulation task (Fig. 7 right), which we refer to as rope-box manipulation task. We considered the initial box stacking task used in our previous work [2] (top left), and a modified one where we made the task of retrieving the underlying state of the system harder. We achieved this by i) using more similar box textures which made it more difficult to separate the states, and ii) by introducing different lighting conditions which made observations containing the same states look more dissimilar. We refer to the original setup as the normal stacking task denoted by ns, and to the modified one as the hard stacking task denoted by hs.

In the rope-box manipulation task (Fig. 7 right), denoted by rb, a rope connects two boxes constraining their movement. To challenge the visual action planning, we again introduced different lighting conditions as well as the deformability of the rope.

These three setups enable automatic evaluation of the structure of the latent space \( Z_{sys} \), the quality of visual plans \( P_t \) generated by the LSR and MM, and the quality of action plans \( P_a \) predicted by the APN. Moreover, they enable to perform a more thorough ablation studies on the introduced improvements of our framework which were not possible in our earlier version of the LSR [2] since the resulting visual action plans achieved a perfect evaluation score.

All setups were developed with the Unity engine [45] and the resulting images have dimension \( 256 \times 256 \times 3 \). In the stacking tasks, four boxes with different textures that can be stacked in a \( 3 \times 3 \) grid (dotted lines in Fig. 7). A grid cell can be occupied by only one box at a time which can be moved according to the stacking rules: i) it can be picked only if there is no other box on top of it, and ii) it can be released only on the ground or on top of another box inside the \( 3 \times 3 \) grid. In both versions of the stacking task, the position of each box in a grid cell was generated by introducing \( \sim 17\% \) noise along \( x \) and \( y \) axes which was applied when generating both action and no-action pairs. The action-specific information \( u \), shown in Fig. 7 left, is a pair \( u = (p, r) \) of pick \( p \) and release \( r \) coordinates in the grid modelled by the row and column indices, i.e., \( p = (p_r, p_c) \) with \( p_r, p_c \in \{0, 1, 2\} \), and equivalently for \( r = (r_r, r_c) \).

In the rope-box manipulation task, two boxes and a rope can be moved in a \( 3 \times 3 \) grid with 4 pillars according to the following manipulation rules: i) a box can only be pushed one cell in the four cardinal directions but not outside the grid, ii) the rope can be lifted over the closest pillar, iii) the rope cannot be stretched over more that two cells, meaning the boxes can never be more than one move apart from being adjacent. In this task, the action-specific information \( u \), shown in Fig. 7 right, denotes whether the rope is moved over the closest pillar (top) or a box is moved in the grid (bottom) with respective pick \( p \) and release \( r \) coordinates.

According to the above rules, the training datasets \( \mathcal{T}_f \) for stacking tasks contain all possible 288 different grid configurations, i.e., the specification of which box, if any, is contained in each cell. In case of the rope-box manipulation task, \( \mathcal{T}_f \) contains 157 different grid configurations comprising the position of the rope and boxes. These 288/157 grid configurations represent the covered states in these tasks. Note that the exact
number of underlying states is in general not known. Given a pair of states and the task rules, it is possible to analytically determine whether or not an action is allowed between them. In addition, we can determine the grid configuration associated with an image (i.e., its underlying state) contained in the produced visual plan \( P_T \) using classifiers. These were trained on the decoded images and achieved accuracy greater than 98.8% on a holdout dataset composed of 750 samples for both versions of the stacking task and the rope-box task. All the implementation details can be found on our code repository.²

A. Experiment Objectives and Implementation Details

Our experiments are designed to answer the following questions:

1) MM What is the impact of the action term \( 4 \) in the augmented loss function \( 6 \) on the structure of the latent space? How do the respective parameters (e.g., minimum distance) influence the overall LSR performance? Lastly, how does the VAE framework perform compared to the AE one for modelling the mappings \( \xi \) and \( \omega \) in the MM?

2) LSR What is the performance of the LSR compared to state of the art solutions like \( 8 \) and \( 9 \)?, and what is the influence of the action term \( 4 \) on it? How do the respective LSR parameters (e.g., number of components) and the choice of the clustering algorithm impact the overall LSR performance? How good is the LSR approximation of the covered regions?

3) APM What is the performance of the APN model?

In this section, we present the implementation details and introduce the notation used to easily refer to the models in consideration. For VAEs (used in MM), each model is annotated by \( \text{VAE}_{ld \text{-} task \text{-} d} \) where \( ld \) denotes the dimension of the latent space, \( task \) denotes the version of the task and is either \( ns \), \( hs \) or \( rb \) for the normal stacking task, hard stacking tasks or rope-box manipulation task, respectively. The parameter \( d \) indicates whether or not the model was trained with the action loss term \( 4 \). We use \( d = b \) to denote a baseline VAE trained with the original VAE objective \( 5 \), and \( d = L_1 \) to denote an action VAE trained with the loss function \( 6 \) including the action term \( 4 \) using metric \( L_1 \). Compared to \( 7 \), we consider only \( L_1 \) metric in our simulated experiments due to its superior performance over the \( L_2 \) and \( L_\infty \) metrics established in \( 7 \).

All VAE models used a ResNet architecture \( 46 \) for the encoder and decoder networks. They were trained for 500 epochs on a training dataset \( T_T \) composed of 65% action pairs and 35% no-action pairs for stacking tasks, and 50% action pairs and 50% no-action pairs for rope-box manipulation task. For each combination of parameters \( ld, task, \) and \( d \), we trained 5 VAEs initialized with different random seeds. Same seeds were also used to create training and validations splits of the training dataset. The weight \( \beta \) in \( 5 \) and \( 6 \) was gradually increased from 0 to 2 over 400 epochs, while \( \gamma \) was fixed to 100. In this way, models were encouraged to first learn to reconstruct the input images and then to gradually structure the latent space. The minimum distance \( d_m \) was dynamically increased every fifth epoch starting from 0 using \( \Delta d_m = 0.1 \) as described in Sec. \( V \). The effect of this dynamic parameter increase is shown in Fig. \( 5 \).

For LSR, we denote by \( \text{LSR} \cdot L_1 \) a graph built using the metric \( L_1 \) in Algorithm ⁴. The parameters \( \tau_{\text{min}} \) and \( \tau_{\text{max}} \) in the LSR optimization \( 9 \) were set to 0 and 3, respectively. Unless otherwise specified, we fixed \( ld = 12 \) for all tasks. Moreover, the number of graph-components \( c_{\text{max}} \) in the optimization of the clustering threshold \( 8 \) was set to 1 for \( ns \), and 20 for \( hs \) and \( rb \). These choices are explained in detail in the following sections. Given an LSR, we evaluated its performance by measuring the quality of the visual plans found between 1000 randomly selected start and goal observations from an unseen holdout set containing 2500 images. To automatically check the validity of the found paths, we used the classifiers on the observations contained in the visual plans to get the respective underlying states. We then defined a checking function (available on the code repository) that, given the states in the paths, determines whether they are allowed or not according to the the stacking or the manipulation rules. In the evaluation of the planning performance we considered the following quantities: \( i) \) percentage of cases when all shortest paths from start to goal observations are correct, denoted as \( \% \text{All} \), \( ii) \) percentage of cases when at least one of the proposed paths is correct, denoted as \( \% \text{Any} \), and \( iii) \) percentage of correct single transitions in the paths, denoted as \( \% \text{Trans} \). We refer to the \( \% \text{Any} \) score in \( ii \) as \( \text{partial scoring} \), and to the combination of scores \( i-iii \) as \( \text{full scoring} \). Mean and standard deviation values are reported over the 5 different random seeds used to train the VAEs.

For APNs, we use the notation \( \text{APN}_{ld \text{-} task \text{-} d} \) analogous to the VAEs. The APN models were trained for 500 epochs on the training dataset \( T_T \) obtained following the procedure described in Sec \( VII \) using \( S = 1 \). Similarly as for LSR, we report the mean and standard deviation of the performance obtained over the 5 random seeds used in the VAE training.
B. MM Analysis

In this section, we validate the MM module answering the questions in point 1) of Sec. IX-A. In the first experiment, we investigated the influence of the dynamic parameter \( d_m \) on the LSR performance. We then studied the structure of the latent space by analyzing the distance between encodings of different states. Lastly, we compared the LSR performance when modelling MM with an AE framework instead of a VAE.

1) Influence of dynamic \( d_m \): A key parameter in the action term (4) is the minimum distance \( d_m \) encouraged among the action pairs. We considered the hard stacking and rope-box manipulation tasks and validated the approach proposed in Sec. V which dynamically increases \( d_m \) to separate action and no-action pairs (see Fig. 5). At the end of the training, the approach results in \( d_m = 2.3 \pm 0.1 \) and \( d_m = 2.6 \pm 0.2 \) for the hard stacking and rope-box tasks, respectively.

Figure 8 shows the performance of the LSR using partial scoring on the hard stacking task (blue) and rope-box manipulation task (orange) obtained for the dynamic \( d_m \) (solid lines), and a selected number of static \( d_m \) parameters (cross markers with dashed lines) ranging from low \((d_m = 0)\) to high \((d_m = 100)\) values. Among the latter, we included the static \( d_m = 11.6 \) and \( d_m = 6.3 \) obtained using our previous approach in \( \text{V} \) on the stacking and the rope-box tasks, respectively. We observed that: i) the choice of \( d_m \) heavily influences the LSR performance, where same values of \( d_m \) can lead to different behavior depending on the task (e.g., \( d_m = 11.6 \)), ii) the dynamic \( d_m \) leads to nearly optimal performance regardless of the task compared to the grid searched static \( d_m \). Note that even though there are static \( d_m \) values where the performance is higher than in the dynamic case (e.g., \( d_m = 3 \) with 93.1% for stacking and \( d_m = 9 \) with 91.2% for the rope-box task), finding these values a priori without access to ground truth labels is hardly possible.

This approach not only eliminates the need for training the baseline VAEs as in \( \text{V} \) but also reaches a value of \( d_m \) that obtains a better separation of covered regions \( Z_{\text{sys}} \) without compromising the optimization of the reconstruction and KL terms. In fact, as discussed in Sec. \( \text{V} \) the reconstruction, KL and action terms in the loss function (5) have distinct influences on the latent space structure which can be in contrast to each other. The proposed dynamic increase of \( d_m \) results in a lower \( d_m \) value than in \( \text{V} \), which in turn yields small distances between the action pair states while still being more beneficial than a simple static \( d_m = 0 \). Such small distances in the action term are desirable as they do not contradict the KL term. This can explain why the LSRs with higher values of \( d_m \) reach worse performance compared to the dynamic one.

2) Separation of the states: We investigated the effect of the action loss (4) on the structure of the latent space by analyzing the separation of the latent points \( z \in T_z \) corresponding to different underlying states of the system. For simplicity, we report only results for the normal stacking task but we observed the same conclusions for the hard stacking and the rope-box manipulation tasks. Recall that images in \( T_i \) containing the same state looked different because of the introduced positioning noise in the stacking tasks (and different lightning conditions in the case of \( hs \) as well as the deformability of the rope in \( rb \)).

Let \( z_s \) be the centroid for state \( s \) defined as the mean point of the training latent samples \( \{ z_{s,i} \} \subset T_z \) associated with the state \( s \). Let \( d_{\text{inter}}(z_{s,i}, z_p) \) be the intra-state distance defined as the distance between the latent sample \( i \) associated with the state \( s \), namely \( z_{s,i} \), and the respective centroid \( z_s \). Similarly, let \( d_{\text{inter}}(\bar{z}_s, \bar{z}_p) \) denote the inter-state distance between the centroids \( z_s \) and \( \bar{z}_p \) of states \( s \) and \( p \), respectively.

Figure 9 reports the mean values (bold points) and the standard deviations (thin lines) of the inter- (in blue) and intra-state (in orange) distances for each state \( s \in \{1, \ldots, 288\} \). In the normal stacking task when using the baseline model \( \text{VAE}_{12-ns-b} \) (top) and the action model \( \text{VAE}_{12-ns-L_1} \) (bottom). In case of the baseline VAE, we observed similar intra-state and inter-state distances. This implies that samples of different states were encoded close together in the latent space which can raise ambiguities when planning. On the contrary, when using \( \text{VAE}_{12-ns-L_1} \), we observed that the inter- and intra-state distances approach the values 5 and 0, respectively. These values were imposed with the action term (4) as the minimum distance \( d_m \) reached 2.6. Therefore, even when there existed no direct link between two samples of different states, and thus the action term for the pair was never activated, the VAE was able to encode them such that the desired distances in the latent space were respected. Similar conclusions also hold for the hard stacking and the rope-box manipulation tasks, whose plots are omitted for the interest of space.

Finally, we analyzed the difference between the minimum inter-state distance and the maximum intra-state distance for each state. The higher the value the better separation of states in the latent space since samples of the same state are in this case closer to each other than samples of different states. When the latent states were obtained using the baseline \( \text{VAE}_{12-ns-b} \), we observed a non-negative distance for 0/288 states with an average value of \( \approx -1.2 \). This implies that only weak separation occurred in the latent space for samples of different states. On the other hand, when calculated on points encoded with \( \text{VAE}_{12-ns-L_1} \), the difference became non-negative for 284/288 states and its mean value increased to \( \approx 0.55 \), thus achieving almost perfect separation. In the hard stacking
task, we similarly found that VAE_{12}-hs-b reached an average difference of $-5.86$ (being non-negative for 0/288 states), while the action model VAE_{12}-hs-L_1 reduced the average difference to $-0.04$ (being non-negative for 121/288 states). This result demonstrates the difference in the difficulty between the two versions of the box stacking task and highlights the challenges of visual action planning on the harder stacking task where worse separation of states was achieved. For the rope-box manipulation task we obtained, coherently with the box stacking results, an average difference of $-2.95$ (being non-negative for 37/157 states) with the baseline model, which improved to 0.15 with the action model VAE_{12}-rb-L_1 (being non-negative for 100/157 states).

In Appendix-B we performed an ablation study on the latent space dimension, justifying the choice $ld = 12$ in our simulations. For each considered latent dimension, we then additionally analyzed the influence of the action loss on the structure of the latent space by calculating relative contrast [47, 48] in Appendix-A, which evaluates the significance of distances among points in the latent space. We observed higher values for action VAEs compared to the baseline VAEs, showing that the action loss leads to higher relative contrast in the latent space.

We conclude that the action term $\gamma$ and the dynamic $d_m$ contribute to a better structured latent space $Z_{sys}$.

3) VAE compared to AE: VAE framework is only one of the possible models for the MM. We justify this modeling choice by comparing it to the AE framework. Similarly as VAE, an AE model consists of an encoder and a decoder network which are jointly trained to minimize the the Mean Squared Error (MSE) between the original input and its decoded output. In contrast to VAEs, the two networks in AEs do not model a probability distribution. Since the KL divergence in VAE acts as a regularization term, we employed the stable weight-decay Adam optimizer from [49] with default parameters to make the comparison more fair. We denote the model by AE-b. Analogously to VAE, the original AE loss was augmented with the action loss $\gamma$, which we denote by AE-L_1. Note that L_1 refers only to the metric in [4] and not in the calculation of MSE.

We modelled the AE encoder and decoder networks using the same ResNet [46] architecture as in case of VAEs. We set $ld = 12$, $\gamma = 1000$ and increased the minimum distance $d_m$ dynamically every fifth epoch starting from 0 using $\Delta d_m = 1$, as described in Sec. [4]. The LSR was built using the same $\tau_{min} = 0$ and $\tau_{max} = 3$ (Algorithm 2).

Table II shows the LSR performance using partial scoring on all simulated tasks when MM was modelled as an AE (top two rows) and as a VAE (bottom row). Not only we observed a superior performance of VAE compared to the AE but once again the effectiveness of the action term $\gamma$ on all the tasks as it increased the average AE performance from 0.1% to 36.3% for ns, from 0.1% to 33.6% for hs, and 0.1% to 9.8% for rb. This comparison shows that the probabilistic modeling adopted by VAEs resulted in a latent space that is more adequate for visual action planning with respect to the considered AEs.

| Model           | ns [%] | hs [%] | rb [%] |
|-----------------|--------|--------|--------|
| AE-b+LSR-L_1    | 0.1±0  | 0.0±0  | 0.1±0  |
| AE-L_1+LSR-L_1  | 36.3±26.9 | 33.6±10.3 | 9.8±5.4 |
| VAE-L_1+LSR-L_1 | 100.0±0 | 92.1±2.9 | 90.4±2.9 |

Table II: Comparison of the LSR performance using partial scoring when modelling MM with an AE (top two rows) and a VAE (bottom row) framework on all the simulated tasks. Best results in bold.

C. LSR Analysis

In this section, we analyze the LSR performance by answering the questions stated in point [2] of Sec. [IX-A]. Firstly, we compared the LSR performance to the method in [8] and one inspired by [9]. Secondly, we investigated the influence of the action term $\gamma$ on the LSR performance. Thirdly, we investigated the influence of the upper bound on the number of connected components $c_{max}$ used in (8). Next, we performed an extensive comparison of the LSR algorithm using different clustering algorithms in Phase 2 of Algorithm 1. Finally, we analyzed the covered regions determined by the LSR.

1) LSR comparison: We compared the performance of the LSR on all simulated tasks with two benchmark methods introduced below. In all the experiments, we considered the baseline models VAE_{12}-b and the action model VAE_{12}-L_1 trained with the action term $\gamma$. We compared our method with Semi-Parametric Topological Memory (SPTM) framework presented in [8] and an MPC-based approach inspired by [9].

In SPTM, we connected action pairs (treated as one-step trajectories) and no-action pairs (considered temporarily close) in the latent memory graph. As in [8], we added $N_{sc}$ more shortcut edges connecting the encodings that are considered closest by the retrieval network to the memory graph. In the localization step, we used the median of $k = 5$ nearest neighbours of the nodes in the memory graph as recommended in [8]. To select the waypoint, we performed a grid search over $s_{reach} \in \{0.75, 0.9, 0.95\}$ and chose $s_{reach} = 0.95$. We also performed a grid search over $N_{sc} \in \{0.2 \cdot 10^2, 1 \cdot 10^4, 1 \cdot 10^5, 1 \cdot 10^6, 1.5 \cdot 10^6, 2 \cdot 10^6\}$ and used the values $N_{sc} = 1.0 \cdot 10^9, 1.5 \cdot 10^6, 2.0 \cdot 10^6$ for ns, hs and rb, respectively. We used high number of shortcuts compared to $N_{sc} = 2 \cdot 10^2$ in [8] because we only had access to one-step trajectories instead of full roll-outs.
Using low number of shortcuts resulted in a memory graph consisting of large amount of disconnected components which impeded planning. For example, in hard stacking task using $N_{sc} = 2 \cdot 10^2$ yielded a graph with 2243 connected components which led to almost zero correct transitions over the 1000 test paths. A higher number of shortcuts instead improved the connectivity of the graph and thus its planning capabilities.

The MPC-inspired baseline is composed of a learned transition model $f_t(\cdot)$ and a learned action validation model $f_a(\cdot)$, both taking the current latent state $z_1$ and the applied action $a$ as inputs. The transition model then predicts the next state $z_2 = f_t(z_1, a)$, while the validation model $f_a(z_1, u)$ predicts whether the given action $u$ was allowed or not.

These models are used in an MPC-style approach, where first a search tree is constructed for a given start state $z_1$ by iterating over all allowed action using $f_a(z_1, u)$ with $u \in U$ and predicting the consecutive states with the transition model $f_t(\cdot)$. The search is performed at each time step and until the search tree has reached a specified horizon $N$. Lastly, the path in the built tree leading to the state closest to the goal using $L_1$ distance is selected and the first action in the sequence is applied. This procedure is repeated until all proposed actions lead further from the goal. In our case, the resulting state and action sequence is decoded into a visual action plan and evaluated in the same way as the LSR.

We implemented $f_t$ and $f_a$ as a three layer MLP-regressor and MLP-classifier, respectively, with 100 hidden units. For a fair comparison, we trained $f_t$ and $f_a$ using training encodings $T_z$ from the same MM that was used for building the LSR. As $T_z$ only includes allowed actions, we augmented the training data for $f_a(\cdot)$ with an equal amount of negative examples by randomly sampling $u \in U$. We used horizon $N = 4$. The trained $f_t$ models achieved $R^2$ coefficient of determination [50] of 0.96, 0.96, and 0.88 (highest 1) for the normal, hard stacking and rope-box datasets, respectively. The $f_a(\cdot)$ model was evaluated on 1000 novel states and by applying all possible actions on each state. It achieved an accuracy score of $88.5 \pm 1.8, 97.3 \pm 0.2$, and $87.4 \pm 0.8$ for the normal, hard stacking and rope-box datasets, respectively. Note that the normal and hard stacking tasks has exactly 48 unique actions with $\approx 9.4\%$ of them being allowed on average. The rope-box task on the other hand has 25 unique actions with an average of $\approx 17.1\%$ being allowed per state.

Table [III] shows the result of our method (VAE-L1 + LSR-L1), the SPTM framework and the MPC-based approach (VAE-L1 + MPC) evaluated on the full scoring on the normal box stacking (top), hard box stacking (middle), and rope-box manipulation task (bottom). We observed that the proposed approach (VAE-L1 + LSR-L1) significantly outperformed the considered benchmark methods. This can be explained by the fact that SPTM- and MPC-based methods are more suited for tasks where the provided data consists of rolled out trajectories in which small state changes are recorded in consecutive states. In contrast, as discussed in Sec. [VIII] our method is best applicable when actions lead to distinguishable different observations. This allows to consider only pairs of observations as input dataset instead of requiring entire trajectories. Moreover, a core difference between our approach and SPTM is that we do not assume that each observation maps into a unique underlying state, but rather, as described in Sec. [IV] we structure and cluster observations in such a way that observations associated with the same underlying state are grouped together. We reiterate that this approach is best suited for tasks with finite and distinguishable states, which differ from continuous RL setting used by SPTM.

2) Influence of the action term: We investigated how the LSR performance is affected by the action term [4] by comparing it to the variant where MM was trained without it (VAE-b + LSR-L1). The results on the full scoring for all the tasks are shown in Table [III]. We observed deteriorated LSR performance when using baselines VAE12-b compared to the action VAEs regardless the task. This indicates that VAEs were not able to separate states in $Z_{sys}$. In addition, the results underpin the different level of difficulty of the tasks as indicated by the drop in the LSR performance on $h_s$ and $r_b$ compared to $n_s$ using the action VAE-L1.

In summary, this simulation campaign demonstrates the effectiveness of the LSR on all the considered simulated tasks involving both rigid and deformable objects compared to existing solutions, as well as supports the integration of the action term in the VAE loss function.

| Task | Model | % All | % Any | % Trans. |
|------|-------|-------|-------|----------|
| $n_s$ | VAE-L1 + MPC | 2.3 ± 0.3 | 2.3 ± 0.3 | 69.2 ± 1.0 |
| | SPTM [8] | 0.2 ± 0.1 | 0.5 ± 0.3 | 51.9 ± 1.4 |
| | VAE-L1 + LSR-L1 | 2.5 ± 0.5 | 4.1 ± 1.0 | 59.7 ± 4.9 |
| $h_s$ | VAE-L1 + MPC | 2.1 ± 0.4 | 2.1 ± 0.4 | 67.8 ± 0.3 |
| | SPTM [8] | 0.0 ± 0.0 | 0.0 ± 0.0 | 23.6 ± 7.7 |
| | VAE-L1 + LSR-L1 | 0.2 ± 0.1 | 0.2 ± 0.1 | 38.0 ± 2.0 |
| $r_b$ | VAE-L1 + MPC | 6.2 ± 0.5 | 6.2 ± 0.5 | 73.8 ± 0.8 |
| | SPTM [8] | 0.0 ± 0.0 | 0.4 ± 0.3 | 25.2 ± 9.7 |
| | VAE-L1 + LSR-L1 | 0.2 ± 0.1 | 0.2 ± 0.1 | 0.2 ± 0.1 |

Table III: Planning performance using full scoring for the normal (top) and hard (middle) box stacking tasks and rope-box manipulation task (bottom) using MPC and SPTM [8] methods, baseline VAE-b and action VAE-L1. Best results in bold.

3) Influence of the maximum number of connected components: The optimization method described in Sec. [VI-B requires setting an upper bound on the number of graph-connected components $c_{max}$ of the LSR. Table [IV] shows how different choices of upper bounds influence the LSR performance on all simulated tasks.

| $c_{max}$ | $n_s$ [%] | $h_s$ [%] | $r_b$ [%] |
|-----------|-----------|-----------|-----------|
| 100 | 100.0 ± 0.0 | 65.3 ± 24.6 | 4.5 ± 5.6 |
| 50 | 99.5 ± 0.4 | 88.6 ± 5.4 | 55.8 ± 28.8 |
| 100 | 99.0 ± 0.3 | 91.5 ± 3.8 | 80.4 ± 10.6 |
| 20 | 97.5 ± 0.5 | 92.1 ± 2.9 | 90.4 ± 2.9 |

Table IV: LSR performance on all simulated tasks for different $c_{max}$ values. Best results in bold.
We observed that the results are rather robust with respect to the $c_{\text{max}}$ value. For all tasks, the performance dropped for a very high $c_{\text{max}}$, such as $c_{\text{max}} = 100$, while in the hard stacking task and especially in the rope-box manipulation task, we additionally observed a drop for a very low $c_{\text{max}}$, such as $c_{\text{max}} = 1$. This behavior can be explained by the fact that the lower the $c_{\text{max}}$ the more the system is sensitive to outliers, while the higher the $c_{\text{max}}$ the greater the possibility that the graph is disconnected which potentially compromises its planning capabilities. For example, in the hard stacking task, outliers arise from different lightning conditions, while in the rope-box manipulation task they arise from the deformability of the rope. In contrast, no outliers exist in the normal stacking task which is why a single connected component is sufficient for the LSR to perform perfectly. For all further evaluation, we set $c_{\text{max}} = 1$ for ns and $c_{\text{max}} = 20$ for hs and rb.

This result demonstrates the robustness of the approach with respect to $c_{\text{max}}$ as well as justifies the choices of the $c_{\text{max}}$ values in the rest of simulations.

4) Comparing different clustering methods for Phase 2: We showcase the effect of the outer optimization loop described in Algorithm 2 on several different clustering methods used in Phase 2 in Algorithm 1 on the hard stacking task. We considered Epsilon clustering used in our earlier work [7], Mean-shift [51], OPTICS [52], Linkage (single, complete and average) [53] and HDBSCAN [54] algorithms. We provide a summary of the considered algorithms in Appendix C. The performance of the considered clustering methods (except for HDBSCAN) depends on a single input scalar parameter that is hard to tune. However, as described in Sec. VI-B we are able to optimize it by maximizing the objective in (8).

Table VI reports the LSR performance with different clustering algorithms when performing grid search to determine their input scalar parameters (left) and when using our automatic optimization (right). Partial scoring using VAE$_{12}$-hs-L1 is shown. Note that the grid search was only possible in this problem setting as the ground truth can be retrieved from the trained classifiers but it is not generally applicable. Firstly, the results show that average-linkage, used for our LSR in Sec. VI-A together with our automatic input parameter optimization outperformed the other alternatives. The results of the grid search show that the automatic criteria for identifying different cluster densities, adopted by OPTICS and HDBSCAN, did not effectively retrieve the underlying covered regions. Meanshift performed better but its approximation of spherical clusters did not lead to the optimal solution. Similar performance to Meanshift was obtained with single- and complete-linkage algorithms showing that the respective distance functions are not either suited for identifying covered regions. The same applies for the epsilon clustering.

Concerning the optimization results, they highlight the effectiveness of the optimization procedure in Algorithm 2 as they are comparable to the ones obtained with the grid search for all clustering methods. Note that grid search led to a slightly lower performance than the optimization for meanshift, complete-linkage and average-linkage. In these cases, the grid was not fine enough which points out the difficulty of tuning the respective parameters.

This investigation demonstrates the effectiveness of our proposed optimization loop and shows that the average-linkage clustering algorithm led to the best LSR performance among considered alternatives for the hard box stacking task.

| Clust. method    | Grid Search [%] | Optimization [%] |
|------------------|-----------------|-------------------|
| Epsilon          | 83.5 ± 4.8      | 65.8 ± 12.2       |
| Meanshift        | 78.2 ± 3.3      | 80.2 ± 5.9        |
| OPTICS           | 44.3 ± 8.7      | 40.8 ± 6.1        |
| HDBSCAN          | 16.1 ± 5.7      | -                 |
| Single-linkage   | 79.3 ± 8.8      | 65.8 ± 12.2       |
| Complete-linkage | 79.1 ± 6.4      | 81.4 ± 4.8        |
| Average-linkage  | 91.1 ± 2.5      | 92.1 ± 2.9        |

Table V: Comparison of the LSR performance for different clustering algorithms for the hard box stacking task. Partial scoring is reported when applying grid search (left column) and when using the optimization in Algorithm 2 (right column). Best results in bold.

5) Covered regions using LSR: To show that the LSR captures the structure of the system, we checked if observations corresponding to true underlying states of the system, that have not been seen during training, are properly recognized as covered. Then, we checked if observations from the datasets of the remaining simulated tasks as well as from the 3D Shapes dataset [55] are marked as uncovered since they correspond to out-of-distribution observations. The covered regions $Z^c_{sys}$ were computed using the epsilon approximation in (7).

Table VI reports the results of the classification of covered states obtained by the models trained on normal (first row) and hard (second row) box stacking tasks and rope-box manipulation task (third row). Holdout datasets for each simulated task were used. For the normal stacking task, results show that the LSR almost perfectly recognized all the covered states (ns column) with the average recognition equal to 99.5%, while it properly recognized on average 4694/5000 samples (93.9% - hs column) hard version. An almost perfect average recognition was also obtained on the rope-box manipulation task (99.6% - rb column). For out-of-distribution observations, the lower the percentage the better the classification. Table VI shows that the models trained on ns (first row, columns hs, rb, 3D Shapes) and hs (second row, columns ns, rb, 3D Shapes) were able to perfectly identify all non-covered states, while worse performance was observed for the rope-box models which misclassified $\approx 10\%$ of the uncovered datasets (third row, columns ns, hs, 3D Shapes). This decrease in performance could be explained by the fact that capturing the state of a deformable object is much more challenging than rigid objects.

We conclude that LSR provides a good approximation of the global structure of the system as it correctly classified most of the observations representing possible system states as covered, and out-of-distribution observations as not covered.

D. APM Analysis

We evaluated the accuracy of action predictions obtained by APN-L1 on an unseen holdout set consisting of 1611, 1590 and 948 action pairs for the normal stacking, hard stacking and rope-box manipulation tasks, respectively. As the predicted actions can be binary classified as either true or false, we
calculated the percentage of the correct proposals for picking and releasing, as well as the percentage of pairs where both pick and release proposals were correct. For rope-box task, we additionally calculated the percentage of the correct proposal for either rope or box action. We evaluated all the models on 5 different random seeds. For both stacking versions, all the models performed with accuracy 99% or higher, while rope-box models achieved ≈ 96%. This is because the box stacking task results in an 18-class classification problem for action prediction which is simple enough to be learned from any of the VAEs, while the classification task in the rope-box is slightly more challenging due to the required extra prediction whether to move a rope or a box.

X. FOLDING EXPERIMENTS

In this section, we validate the proposed approach on a real world experiment involving manipulation of deformable objects, namely folding a T-shirt. As opposed to the simulated tasks, the true underlying states were in this case unknown and it was therefore not possible to define an automatic verification of the correctness of a given visual action plan.

The folding task setup is depicted in Fig. 12 (middle). We used a Baxter robot equipped with a Primense RGB-D camera mounted on its torso to fold a T-shirt in different ways. The execution videos of all the performed experiments and respective visual action plans can be found on the project website. A summary of the experiments can also be found in the accompanying video. For this task, we collected a dataset \( \mathcal{T}_f \) containing 1283 training tuples. Each tuple consists of two images of size 256 \( \times \) 256 \( \times \) 3, and action specific information \( u \) defined as \( u = (p, r, h) \) where \( p = (p_r, p_c) \) are the picking coordinates, \( r = (r_r, r_c) \) the releasing coordinates and \( h \) picking height. An example of an action and a no-action pair is shown in Fig. 4. The values \( p_r, p_c, r_r, r_c \in \{0, \ldots, 255\} \) correspond to image coordinates, while \( h \in \{0, 1\} \) is either the height of the table or a value measured from the RGB-D camera to pick up only the top layer of the shirt. Note that the separation of stacked clothing layers is a challenging task and active research area on its own \cite{56} and leads to decreased performance when it is necessary to perform it, as shown in Sec. X-E2. The dataset \( \mathcal{T}_f \) was collected by manually selecting pick and release points on images showing a given T-shirt configuration, and recording the corresponding action and following configuration. No-action pairs, representing \( \approx 37\% \) of training tuples in \( \mathcal{T}_f \), were generated by slightly perturbing the cloth appearance.

| ns [%]         | hs [%]         | rb [%]         | 3D Sh. [%]         |
|---------------|---------------|---------------|------------------|
| 99.47 ± 0.27  | 0.0 ± 0.0     | 0.0 ± 0.0     | 0.0 ± 0.0        |
| 0.0 ± 0.0     | 95.71 ± 0.61  | 0.0 ± 0.0     | 0.0 ± 0.0        |
| 9.48 ± 7.45   | 13.5 ± 8.57   | 99.6 ± 0.1    | 9.72 ± 8.38      |

Table VI: Classification of covered states for the normal (first row) and hard (second row) box stacking models and rope-box models (third row) when using as inputs novel images from the tasks (ns, hs and rb columns) and the 3D Shapes (3D Sh. column) datasets.

1) **MM** Does the action loss term \( \mathcal{L}_d \) improve the structure of the latent space for the folding task?
2) **LSR** How good is the approximation of the covered regions provided by the LSR for a real world dataset?
3) **APM** How does the APN perform in comparison to alternative implementations of the APM?
4) **System** How does the real system perform and how does it compare to our earlier work \cite{7}? What is the performance on a folding that involves picking the top layer of the shirt?

Following the notations introduced in Sec. IX-A we denote by VAEn_d f-d a VAE with ld-dimensional latent space, where \( f \) stands for the folding task and \( d \) indicates whether or not the model was trained with the action loss \( \mathcal{L}_d \). We use \( d = b \) for the baseline VAEs which were trained with the original training objective \( \mathcal{L}_d \). We use \( d = L_p \) for the action VAEs trained with the objective \( \mathcal{L}_d \) containing the action term \( \mathcal{L}_p \) using metric \( L_p \) for \( p \in \{1, 2, \infty\} \). We modelled VAEs with the same ResNet architecture and same hyperparameters \( \beta, \gamma \) and \( d_{ap} \) as in the box stacking task introduced in Sec. IX but increased the latent space dimension to \( ld = 16 \). We refer the reader to the code repository \cite{4} for implementation details.

For the LSR, we denote by LSR-L_p a graph obtained by using metric \( L_p \) in Algorithm 1. We set the upper bound \( c_{\text{max}} \) in \( \mathcal{L}_d \) to 5, and the search interval boundaries \( \tau_{\text{min}} \) and \( \tau_{\text{max}} \) in Algorithm 2 to 0 and 3.5, respectively.

The performance of the APMs and the evaluation of the system was based on the VAEn_d f-L_1 realization of the MM. We therefore performed the experiments using APN_{16}f-L_1 which was trained on latent action pairs \( \mathcal{T}_f \) extracted by the latent mapping \( \xi \) of VAEn_d f-L_1. We trained 5 models for 500 epochs using different random seeds as in case of VAEs, and used 15% of the training dataset as a validation split to extract the best performing model for the evaluation.

We compared the performance of our system S-OUR consisting of VAEn_d f-L_1, LSR-L_1 and APN_{16}f-L_1 with the systems S-L_1, S-L_2 and S-L_\infty introduced in \cite{7} on the same 5 folding tasks. The start configuration was the fully unfolded shirt shown in Fig. 10 on the left, while the goal configurations are shown on the right. The latter are of increasing complexity requiring a minimum of 2, 3, 3, and 4 folding steps for folds 1-5, respectively.

![Fig. 10: Start state (right) followed by 5 different goal configurations for the folding task.](image)

The lower right configuration requires to pick a layer on top of the T-shirt.

Each fold was repeated 5 times and scored in the same way as in \cite{7}. In particular, we scored the system performance...
where a folding was considered successful if the system was able to fold the T-shirt into the desired goal configuration. As the state space of the T-shirt is high-dimensional, there exists no objective measure that would evaluate the success of the fold automatically. Therefore, the evaluation of the full folding procedure was manually done by a human (one of the authors) but all execution videos of all folds and repetitions can be found on the project website. We additionally evaluated the percentage of successful transitions of the system. A transition was considered successful if the respective folding step was executed correctly. Lastly, we evaluated the quality of the generated visual plans $P_1$ and the generated action plans $P_0$. We considered a visual (action) plan successful if all the intermediate states (actions) were correct. Even for a correctly generated visual action plan, the open loop execution is not robust enough for a real robot system. We therefore added a re-planning step after each action completion as shown in Fig. 12. This accounts, for instance, for potential execution uncertainties, inaccuracies in grasping or in the positioning phases of pick-and-place operations which led to observations different from the ones planned in $P_1$. Note that after each action execution, the current observation of the cloth was considered as a new start observation, and a new visual action plan was produced until the goal observation is reached or the task is terminated. Such re-planning setup was used for all folding experiments. As the goal configuration does not allude to how the sleeves should be folded, the LSR suggests multiple latent plans. A subset of the corresponding visual action plans is shown on the left of Fig. 12. If multiple plans were generated, a human operator selected one to execute. After the first execution, the ambiguity arising from the sleeve folding was removed and the re-planning generated a single plan, shown in the right.

To deal with the sparse nature of the collected dataset, if no path was found from the start to the goal node, the planning was repeated using the closest nodes to the current start and/or goal nodes in the latent space. This procedure was repeated until a path was found.

B. MM Analysis

Analogously to the box stacking task in Sec. X-B, we answered question 1) by analyzing the relative contrast (reported in Appendix A) as well as by evaluating the separation of action and no-action pairs during the training.

1) Influence of dynamic $d_m$: We investigated the influence of the dynamic increase of $d_m$ in the action term $f_a$ on the structure of the latent space. Figure 11 shows the histogram of action (in blue) and no-action (in green) pair distances calculated at different epochs during training using $\text{VAE}_{16}^{-f-b}$ (top row) and $\text{VAE}_{16}^{-f-L_1}$ (bottom row). We observed that the separation was complete in case of action VAEs but was not achieved with the baseline VAEs. To precisely quantify the amount of overlap between action and no-action pairs, we calculated the difference between the minimum action-pair distance and maximum no-action pair distance on the training dataset. Therefore, a positive value implies that action pairs were successfully separated from the no-action pairs. For $\text{VAE}_{16}^{-f-b}$ (top row), the difference evaluated to $-31.8$, $-19.2$, and $-19.4$ for epoch 1, 100, and 500, respectively, while it was improved to $-6.3$, $-1.6$, and $1.5$ in case of the action $\text{VAE}_{16}^{-f-L_1}$ (bottom row). This shows that the dynamic selection of $d_m$ successfully separated the actions and no-action pairs also for the folding task.

C. LSR Analysis

Similarly to the simulated tasks, we exploited the LSR to investigate the covered regions of the latent space $\mathcal{Z}$, thus answering question 2) listed in Sec. X-A. Note that in Sec. X-E the LSR was also employed to perform the folding task with the real robotic system.

1) Covered regions using LSR: We used $\text{VAE}_{16}^{-f-L_1}$ model and reproduced the experiment from Sec. X-C where we measured the accuracy of various novel observations being recognized as covered. We inputted 224 novel observations that correspond to possible states of the system not used during training, as well as 5000 out-of-distribution samples from each of the three datasets of the simulated tasks and the standard 3D Shapes dataset. We observed that the LSR achieved good recognition performance even in the folding task. More precisely, on average 213/224 samples representing true states of the system were correctly recognized as covered, resulting in $95 \pm 2.4\%$ accuracy averaged over the 5 different random seeds. For the four out-of-distribution datasets, all samples were correctly recognized as not covered.

This analysis illustrates the effectiveness of the LSR in capturing the covered regions of the latent space.

D. APM Comparison

In this section we validate the choice of the APM by comparing it to several possible alternatives.

The Action Proposal Network, described in Sec. VII, was built upon the one introduced in [7] to which we added dropout regularization layers. The APN receives as inputs latent action pairs contained in a latent plan found by the LSR, and outputs the predicted action specifics. We refer to the earlier version in [7] as e-APN and to the current version $\text{APN}_{16}^{-f-L_1}$ as APN. We compared the performance of APN to e-APN as well as several alternatives introduced below.
Table VII: Comparison of MSE achieved with different realizations of the Action Proposal Modules. Best results in bold.

| Method     | X Pick | Y Pick | X Release | Y Release | Height | Total   |
|------------|--------|--------|-----------|-----------|--------|---------|
| e-APN [7]  | 144.1 ± 52.2 | 52.8 ± 18.3 | 317.2 ± 143.3 | 159.9 ± 17.4 | 0.0 ± 0.0 | 674.0 ± 147.6 |
| C-APN      | 498.0 ± 63.8 | 47.4 ± 7.7 | 818.8 ± 121.9 | 226.5 ± 92.5 | 0.0 ± 0.0 | 1590.8 ± 155.0 |
| R-APN      | 697.2 ± 345.1 | 246.2 ± 174.9 | 792.4 ± 388.8 | 268.9 ± 157.0 | 0.0 ± 0.0 | 2004.6 ± 908.2 |
| AAB        | 113.0 | 22.4 | 201.4 | 194.7 | 0.0 | 531.5 |
| APN (Ours) | 82.6 ± 22.9 | 29.3 ± 2.2 | 270.6 ± 158.2 | 71.8 ± 15.0 | 0.0 ± 0.0 | 454.3 ± 153.8 |

Fig. 12: Execution of the folding task with re-planning. On the left, a set of initial visual action plans reaching the goal state is proposed. After the first execution, only one viable visual action plan remains.

**Action Averaging Baseline (AAB)**

Firstly, we investigated whether the action predictions can be retrieved directly from the LSR instead of a separate module. The basic idea is to use the latent action pairs in the training dataset to calculate the average action specifics associated with each edge in the LSR. Let \( \mathcal{E}_{\text{sys}} = \{ (z_1, z_2) \in \mathcal{E} | z_1 \in \mathcal{Z}_{\text{sys}}^i, z_2 \in \mathcal{Z}_{\text{sys}}^j \} \) be the set of edges from the reference graph \( \mathcal{E} \) connecting covered regions \( \mathcal{Z}_{\text{sys}}^i \) and \( \mathcal{Z}_{\text{sys}}^j \) (Algorithm 1). We parameterized each edge \( e_{\text{LSR}}^{ij} = (z_{1, \text{LSR}}^i, z_{1, \text{LSR}}^j) \in \mathcal{E}_{\text{LSR}} \) with the action \( u_{\text{LSR}}^{ij} \) obtained by averaging actions corresponding to the edges in \( \mathcal{E}_{\text{sys}}^{ij} \)

\[
u_{\text{LSR}}^{ij} = \frac{1}{|\mathcal{E}_{\text{sys}}^{ij}|} \sum_{(z_1,z_2) \in \mathcal{E}_{\text{sys}}^{ij}} u^{z_1,z_2} (10)
\]

where \( u^{z_1,z_2} \) is the action specification associated with the action pair \( (z_1, z_2) \) in the training dataset \( \mathcal{T}_s \). The parametrization \( \{10\} \) yields the action plan associated with a path \( P_s \).

Secondly, we investigated how the use of the latent encodings as inputs to the APM influences the LSR performance. We compared APN-d with two distinct versions of APMs that use images as inputs.

**C-APN** is a neural network that uses a convolutional encoder followed by the APN. The encoder in C-APN was trained using only MSE loss. During the inference, the observations given to C-APN as input are obtained by decoding the latent plan found by the LSR with the observation generator \( \omega \).

**R-APN** is an extension of C-APN that uses a ResNet encoder identical to the VAE encoder.

Detailed architectures of all the models can be found in our code repository. The training details for APN and APN-d are described in Sec. X. For C-APN-d and R-APN-d, we similarly trained 5 models using different random seeds but on a training dataset \( T_f \) obtained by decoding \( T_s \) with the observation generator \( \omega \) of VAE_{16} \( \cdot f \)-L_1. This is because the visual plans, given to C-APN-d and R-APN-d, are produced by decoding the latent plans with \( \omega \). Moreover, C-APN-d and R-APN-d were trained for 1000 epochs to ensure the convergence of the initialized encoders. Note that we can only obtain one AAB model for a chosen VAE as AAB is defined by the LSR.

We evaluated the performance of all the models on a holdout dataset consisting of 41 action pairs. Given a holdout action pair, we calculated the mean squared error (MSE) between the predicted and the ground truth action specifics. We report the mean and standard deviation of the obtained MSE calculated across the 5 random seeds (except for AAB). The results are shown in Table VII where we separately report the error obtained on picking and releasing as well as the total model error. Firstly, we observed that the added regularization layer positively affected the result as APN achieved lower error than our earlier version e-APN [7]. Secondly, APN significantly outperformed both C-APN and R-APN. Using the latent encodings as inputs also significantly decreased the size of the models and reduces the computational power needed for their training. Lastly, our APN also on average outperformed AAB with respect to the total model error. Although the enhancement compared to the AAB was not as significant as for the other models, APN is beneficial since it is less prone to averaging errors obtained from the LSR and can be easily adapted to any realization of action specifics. Moreover, a neural network realization of the APN potentially allows more accurate modeling of more complex action specifics. In summary, using a separate neural network to predict action specifics from latent representations led to a lower prediction error and can be easily adapted to different types of actions.

**E. System Analysis**

We benchmarked our method against our earlier method in [7] on the same T-shirt folding task, and additionally measured the performance on a more challenging fold involving picking a layer of the cloth on top of another layer.

1) **Folding performance and comparison with [7]**: We performed each fold 5 times per configuration using the goal configuration shown in Fig. 10 and framework S-OUR, consisting of VAE_{16} \( \cdot f \)-L_1, LSR-L_1 and APN_{16} \( \cdot f \)-L_1, and compared the performance with the results from our earlier work [7] obtained using S-L_1, S-L_2 and S-L_∞.

The results are shown in Table VIII while, as previously mentioned, all execution videos, including the respective visual action plans, are available on the website [8]. We report
the total system success rate with re-planning, the percentage of correct single transitions, and the percentage of successful visual plans and action plans from start to goal. We observed that S-OUR outperformed the systems from [7] with a notable 96% system performance, only missing a single folding step which results in a transition performance of 99%. As for S-L₁, S-OUR also achieved optimal performance when scoring the initial visual plans $P_I$ and the initial action plans $P_u$. We thus conclude that the improved MM, LSR and APM modules together contribute to a significant better system than in [7].

2) Folding with multiple layers: As the previous folds resulted in nearly perfect performance of our system, we challenged it with an additional much harder fold that requires picking the top layer of the garment (repeated 10 times).

Table VIII: Results (best in bold) for executing visual action plans on 5 folding tasks (each repeated 5 times) shown in the top. The bottom row shows the fold requiring to pick the top layer of the garment (repeated 10 times).

| Method | Syst. | Trans. | $P_I$ | $P_u$ |
|--------|-------|--------|-------|-------|
| S-OUR  | 96%   | 99%    | 100%  | 100%  |
| S-L₁   | 80%   | 90%    | 100%  | 100%  |
| S-L₂   | 40%   | 77%    | 60%   | 60%   |
| S-L₁ₐ | 24%   | 44%    | 56%   | 36%   |

XI. CONCLUSIONS

In this work, we presented an extended version of the Latent Space Roadmap first introduced in [7] which allows visual action planning of manipulation tasks. Firstly, we improved the building procedure of the LSR in the latent space by introducing an outer optimization loop that eliminates the need for a hard-to-tune clustering parameter. Secondly, we improved the training procedure of the VAE, used to represent the Mapping Module, by dynamically increasing the desired distance between action pairs. We thoroughly investigated the structure of the latent space, and presented a deep insight into the effects that each of the improvements have for the system. In addition, we compared different realizations of the Action Proposal Module and showcased the benefits of using latent representations for generating action plans. Lastly, we evaluated the LSR on three simulated tasks as well as real-world folding task. We introduced a harder version of the box stacking task and a rope-box manipulation task involving a rigid and deformable object, which enabled a more informative ablation study. We showed that the improved LSR significantly outperforms the one presented in [7] on the same folding task.

We are convinced that in order to advance state-of-the-art manipulation techniques for rigid and deformable objects, improvements on two fronts are necessary: learning a structured latent space as well as its exploration. We believe that our proposed method is a step toward achieving this goal which also opens many interesting future directions. For example, we wish to expand our method to encode full trajectories that could be leveraged to further structure the latent space, or to apply it to reinforcement learning settings with active exploration.

REFERENCES

[1] J. Oh, X. Guo, H. Lee, R. L. Lewis, and S. Singh, “Action-conditional video prediction using deep networks in atari games,” in Advances in neural information processing systems, pp. 2863–2871, 2015.
[2] I. Garcia-Camacho, M. Lippi, M. C. Welle, H. Yin, R. Antonova, A. Varava, J. Borras, C. Torras, A. Marino, G. Alenyà, and D. Kräig, “Benchmarking bimanual cloth manipulation,” IEEE Robot. Autom. Lett., vol. 5, no. 2, pp. 1111–1118, 2020.
[3] H. Yin, A. Varava, and D. Kräig, “Modeling, learning, perception, and control methods for deformable object manipulation,” Science Robot., vol. 6, no. 54, 2021.
[4] D. H. Ballard, “Modular learning in neural networks...,” in AAAI, pp. 279–284, 1987.
[5] D. P. Kingma and M. Welling, “Auto-encoding variational bayes,” Int. Conf. Learn. Represent., 2015.
[6] D. Müllner, “Modern hierarchical, agglomerative clustering algorithms,” arXiv preprint arXiv:1109.2378, 2011.
[7] M. Lippi, P. Polukar, M. C. Welle, A. Varava, H. Yin, A. Marino, and D. Kräig, “Latent space roadmap for visual action planning of deformable and rigid object manipulation,” in IEEE/RSJ Int. Conf. Intelligent Robots and Systems, pp. 5619–5626, 2020.
[8] N. Savinov, A. Dosovitskiy, and V. Koltun, “Semi-parametric topological memory for navigation,” in Int. Conf. Learn. Represent., 2018.
[9] T. Lozano-Pérez and L. P. Kaelbling, “A constraint-based method for solving sequential manipulation planning problems,” in IEEE/RSJ Int. Conf. Intelligent Robots and Systems, pp. 3684–3691, IEEE, 2014.
[10] C. Galindo, J.-A. Fernández-Madrigal, J. González, and A. Saffiotti, “Robot task planning using semantic maps,” Robotics and Autonomous Systems, vol. 56, no. 11, pp. 955 – 966, 2008. Semantic Knowledge in Robotics.
[11] D. Hafner, T. Lillicrap, I. Fischer, R. Villegas, D. Ha, H. Lee, and J. Davidson, “Learning latent dynamics for planning from pixels,” in Int. Conf. Mach. Learn., pp. 2555–2565, 2019.
[12] T. Lozano-Pérez and L. P. Kaelbling, “A constraint-based method for solving sequential manipulation planning problems,” in IEEE/RSJ Int. Conf. Intelligent Robots and Systems, pp. 3684–3691, IEEE, 2014.
[13] S. M. LaValle, Planning Algorithms. Cambridge U.K.: Cambridge University Press, 2006.
[44] Unity Technologies, “Unity.”
[45] G. Maeda, M. Ewerton, T. Osa, B. Busch, and J. Peters, “Active incremental learning of robot movement primitives,” in Conf. Robot. Learn., pp. 37–46, 2017.
[46] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” in IEEE Conf. Computer Vision and Pattern Recognition, pp. 770–778, 2016.
[47] C. C. Aggarwal, A. Hinneburg, and D. A. Keim, “On the surprising behavior of distance metrics in high dimensional space,” in Int. Conf. Database Theory, pp. 420–434, Springer, 2001.
[48] K. Beyer, J. Goldstein, R. Ramakrishnan, and U. Shaft, “When is ‘nearest neighbor’ meaningful?”, in Int. Conf. Database Theory, pp. 217–235, Springer, 1999.
[49] Z. Xie, I. Sato, and M. Sugiyama, “Stable weight decay regularization,” arXiv preprint arXiv:2011.11152, 2020.
[50] R. Berk, “A primer on robust regression,” Modern methods of data analysis, p. 292–324, 1990.
[51] Y. Cheng, “Mean shift, mode seeking, and clustering,” IEEE Trans. Pattern Anal. Mach. Intell., vol. 17, no. 8, pp. 790–799, 1995.
[52] M. Ankerst, M. M. Breunig, H.-P. Kriegel, and J. Sander, “Optics: ordering points to identify the clustering structure,” ACM Sigmod record, vol. 28, no. 2, pp. 49–60, 1999.
[53] W. H. Day and H. Edelsbrunner, “Efficient algorithms for agglomerative hierarchical clustering methods,” J. Classif., vol. 1, no. 1, pp. 7–24, 1984.
[54] L. McInnes, J. Healy, and S. Astels, “HDBSCAN: Hierarchical density based clustering,” J. of Open Source Software, vol. 2, no. 11, p. 205, 2017.
[55] C. Burgess and H. Kim, “3d shapes dataset.” https://github.com/deepmind/3dshapes-dataset/, 2018.
[56] J. Qian, T. Weng, L. Zhang, H. Okorn, and D. Held, “Cloth region segmentation for robust grasp selection,” in IEEE/RSJ Int. Conf. Intelligent Robots and Systems, pp. 9553–9560, IEEE, 2020.
[57] 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.
[58] R. J. Campello, D. Moulavi, and J. Sander, “Density-based clustering based on hierarchical density estimates,” in Pacific-Asia Conf. Knowledge Discovery and Data Mining, pp. 160–172, Springer, 2013.

APPENDIX

This section provides complementary results on the separability of data samples for the simulation and the folding tasks, an ablation study considering different latent dimensions as well as a short overview of the different clustering methods.

A. Separability of data samples

To further analyze how the action loss influences the structure of the latent space, we resorted to the concept of relative contrast, introduced in [47] and based on [48]. Given a dataset $\mathcal{T}$ containing $ld$-dimensional data points, the relative contrast $rc$ of $\mathcal{T}$ with respect to the metric $L_p$ is defined as

$$rc(\mathcal{T}, ld, p) = \frac{D_{\text{max}} - D_{\text{min}}}{D_{\text{min}}}$$

where $D_{\text{max}}$ and $D_{\text{min}}$ denote the distances of the furthest and the closest points in the dataset $\mathcal{T}$ from a chosen point [47], respectively. We calculated $rc$ of the latent training dataset $\mathcal{T}_z$ using VAEs with different latent dimensions $ld$. The $rc$ values were used to evaluate significance of distances among points in the latent space $Z$, where a lower $rc$ implies diminishing contrast in distances. Authors of [48] show that, under broad assumptions, $rc$ converges to 0 when $ld$ approaches infinity.
as in this case the distance from a point to its nearest neighbor is identical to the one to its furthest neighbor.

In our case, measuring $rc$ on the full dataset can be misleading since the latent no-action pairs are encoded closely because of the action term. This would lead to a deceptively small $D_{\text{min}}$ and therefore to a higher $rc$ than warranted. To get a reliable estimate of $rc$ given $T_z$, we therefore chose to measure it in the following way. We split the latent training dataset $T_z$ into datasets $T_z^1$ and $T_z^2$ containing the first and the consecutive states of all the training tuples in $T_z$, respectively. We then measured $D_{\text{min}}$ and $D_{\text{max}}$ with respect to each state in $T_z^1$ and $T_z^2$ separately. More precisely, we considered each latent state $z_i \in T_z^1$ as the origin, and calculated $D_{\text{min}}$ and $D_{\text{max}}$ within $T_z^1 \setminus \{z_i\}$. We repeated the same process for all $z_i \in T_z^2$. Finally, we used the average of the obtained $D_{\text{min}}$ and $D_{\text{max}}$ values in the computation of $rc$ as in Eq. (1).

**Simulation tasks:** Table IX reports the relative contrast of all the training tuples in $T_z$, respectively. We then measured $D_{\text{min}}$ and $D_{\text{max}}$ with respect to each state in $T_z^1$ and $T_z^2$ separately. More precisely, we considered each latent state $z_i \in T_z^1$ as the origin, and calculated $D_{\text{min}}$ and $D_{\text{max}}$ within $T_z^1 \setminus \{z_i\}$. We repeated the same process for all $z_i \in T_z^2$. Finally, we used the average of the obtained $D_{\text{min}}$ and $D_{\text{max}}$ values in the computation of $rc$ as in Eq. (1).

**Simulation tasks:** Table IX reports the relative contrast of the baseline and action VAEs. Results for all simulated tasks are reported.

| ld | $ns$ (%) | $hs$ (%) | $rb$ (%) |
|----|---------|---------|---------|
| 4  | 7.9 ± 2.2 | 8.8 ± 7.9 | 62.7 ± 13.9 |
| 6  | 99.96 ± 0.08 | 56.2 ± 23.1 | 74.9 ± 5.0 |
| 8  | 99.96 ± 0.08 | 62.7 ± 18.7 | 80.6 ± 5.3 |
| 10 | 100.0 ± 0.0 | 92.1 ± 2.9 | 90.4 ± 2.9 |
| 16 | 100.0 ± 0.0 | 95.9 ± 1.4 | 92.2 ± 1.1 |
| 32 | 97.5 ± 4.33 | 96.4 ± 0.4 | 92.6 ± 2.0 |

Table XI: Comparison of the LSR performance when using VAEs with different latent dimensions for all the simulated tasks.

C. **Overview of clustering algorithms**

In this section, we provide a brief overview of the ablated clustering methods considered in Sec. IX-C4.

**Epsilon clustering:** used in our earlier work [17] and functionally coincident with DBSCAN [57]. Its performance is affected by the parameter $\varepsilon$, i.e., radius of the $\varepsilon$-neighborhood of every point, and deteriorates when clusters have different densities.

**Mean-shift:** centroid-based algorithm [51] with moving window approach to identify high density regions. At each iteration, the centroid candidates associated to the windows are updated to the mean of the points in the considered region. The window size has a significant influence on the performance.

**OPTICS:** improved version of DBSCAN introduced by [52] in which a hierarchical reachability-plot dendrogram is built, whose slope identifies clusters with different densities. The parameter $\delta \in [0, 1]$ is used to tune the slope and heavily affects the outcome of the algorithm. However, its influence is not easy to understand intuitively, as discussed in [53].

**Linkage:** hierarchical, agglomerative clustering algorithm discussed in Sec. VI-A. Possible dissimilarity functions to merge points are *single*, based on the minimum distance between any pair of points belonging to two distinct clusters, *complete*, based on the maximum distance, and *average*, based on the unweighted average of the distances of all points belonging to two distinct clusters. As discussed in Sec. VI-A the clustering threshold $\tau$ determines the vertical cut through the dendrogram and consequently influences the performance of the algorithm.

**HDBSCAN:** agglomerative clustering algorithm in which the branches of the dendrogram are optimized for non-overlapping clusters using a notion of “cluster stability” based on their longevity. HDBSCAN automatically identifies clusters with different densities and requires specifying only the minimum cluster size prior to the training.