Learning Task Constraints in Visual-Action Planning from Demonstrations

Francesco Esposito, Christian Pek, Michael C. Welle and Danica Kragic

***Abstract*— Visual planning approaches have shown great success for decision making tasks with no explicit model of the state space. Learning a suitable representation and constructing a latent space where planning can be performed allows non-experts to setup and plan motions by just providing images. However, learned latent spaces are usually not semantically-interpretable, and thus it is difficult to integrate task constraints. We propose a novel framework to determine whether plans satisfy constraints given demonstrations of policies that satisfy or violate the constraints. The demonstrations are realizations of Linear Temporal Logic formulas which are employed to train Long Short-Term Memory (LSTM) networks directly in the latent space representation. We demonstrate that our architecture enables designers to easily specify, compose and integrate task constraints and achieves high performance in terms of accuracy. Furthermore, this visual planning framework enables human interaction, coping the environment changes that a human worker may involve. We show the flexibility of the method on a box pushing task in a simulated warehouse setting with different task constraints.**

I. Introduction

The recent advances in representation learning have enabled novel planning approaches, such as visual planning [1], which require significantly less training effort. These approaches allow one to automatically create low dimensional latent space representations that can be used for planning without explicitly modelling the state space specifically for the task. For instance, planning algorithms for visual plans can learn the state representations directly from sets of image pairs of the scene and the executed robot action between the images of each pair instead of using a separate perception module, see example in Fig. 1. Subsequently, classical planning techniques, e.g., graph search, can be used to compute plans from an initial image to a desired goal image of the scene in the learned latent space.

Visual planning approaches with learned state representations have the advantage that non-experts can easily setup the system and plan motions by just providing a goal image of the scene, e.g., a desired box configuration in a box-pushing task (see Fig. 1). However, besides goal configurations, many tasks require additional constraints to be satisfied for safety or desired task properties. For instance, certain box configurations need to be avoided or specific temporal orders to be fulfilled. The complexity further increases when human workers are involved in the task, e.g., a human wants to periodically pickup boxes at a certain location in the workspace of the robot. Yet, the integration of those constraints, for instance using temporal logic, usually requires an explicit model of the system, which is not available in visual planning. In addition, specifying constraints should be as simple and intuitive as demonstrating desired system behaviours through image sequences.

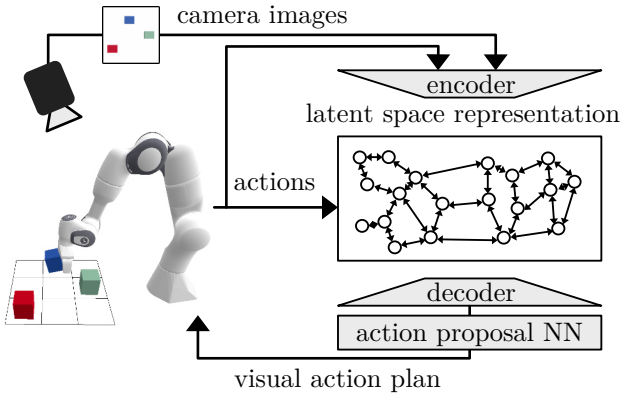

Fig. 1. Overview of visual-action planning. The framework uses images of the scene and corresponding robot actions to create a lower dimensional graph representation in the latent space. Each path in the graph represents a possible path from an initial to a goal image. An action proposal neural network (NN) assigns actions to each possible path to control the robot.

To enable designers to easily specify, compose and integrate task constraints, we propose a visual planning architecture which learns to evaluate the satisfaction of the constraints for a given action plan from demonstrations of policies that satisfy or violate a set of properties, expressed as Linear Temporal Logic (LTL) [2]. The outputs of the learned classifiers are then composed according to the Boolean operators of LTL to allow one to assess more sophisticated user-defined specifications. To demonstrate the flexibility of our approach, we consider the visual planning task illustrated in Fig. 1, in which a robot is pushing boxes and a human can demonstrate desired task properties through sets of images. We can enable designers to specify constraints in the image space while constraint evaluation is performed in a low dimensional latent space representation of the system.

A. Related work

In order to make systems increasingly more autonomous, it is essential that they can not only autonomously learn from their own experiences, but furthermore achieve this by requiring less and less prior knowledge for their setup. Representation learning methods play an important role

All of the authors are with the Division of Robotics, Perception and Learning, School of Electrical Engineering and Computer Science, KTH Royal Institute of Technology, 114 28 Stockholm, Sweden. Mail addresses: {fesp, pek2, mwelle, dani}@kth.se

This work was supported by the European Research Council, Swedish Research Council and Knut and Alice Wallenberg Foundation.

in this context especially when the dimensionality of the system and the complexity of the dynamics make standard approaches intractable. Indeed in this perspective, it is crucial to enable the systems to determine a policy to reach the goal just according to the information extracted out of a stream of raw data, e.g. raw images, because in this way the system would autonomously extrapolate the essential information to plan the sequence of actions to run.

Many recent works focused on using learning to find a more suitable representation by constructing a latent space where planning can be performed [3] [4]. An application of representation learning to decision making and planning is [5], which introduces a Latent Sampling-based Motion Planning (L-SBMP) architecture that leverages a latent representation with techniques from sampling-based motion planning to return a sequence of actions both within visual space and for high-dimensional, complex systems. Instead an important work in this regard but in the field of control is Embed to Control [6], where the authors build on insights from the optimal control formulation to leverage a variational autoencoder (VAE), that learns to generate image trajectories from a latent space in which the dynamics is constrained to be locally linear. The work in [1] also uses VAEs but with augmented loss function to generate Visual Action Plans making use of a Latent Space Roadmap that produces a sequence of images as well as connecting actions given only a start and goal image. However, since these approaches do not operate in a semantically-interpretable latent space, it is difficult to determine whether plans or sequences of actions are satisfying a given set of specifications. For instance, the original version of VAEs [7] solely optimizes for data reconstruction fidelity and does not consider the interpretability of the representation. A possible solution is to train the VAE optimizing a function that combines both accuracy and interpretability, but this implies a lower accuracy because optimizing solely for reconstruction fidelity is typically better at fitting the data than one optimized for both data reconstruction fidelity and interpretability [8].

On the other hand, a completely different approach is taken when a mathematical model of the system is available, because in these cases it is possible to perform an extensive analysis of the correctness of the system with respect to the requirements by using formal verification techniques [9] [10]. Formal verification describes the process of checking whether a model of a system, e.g., a finite transition system, satisfies given specifications, such as formulas of temporal logics [11]. In this regard, LTL and Computation Tree Logic (CTL) are the most commonly encountered temporal logics in computer science [12]. There also exist logics, such as Signal Temporal Logic (STL) [13], and Metric Temporal Logic (MTL) [14], in which the temporal operators have specific time bounds. They have quantitative semantics, which allow one to quantify how far a system execution is from satisfying a given formula. Recent works, such as in [15], showed that quantitative semantics can be used to formulate machine learning and control problems as optimization problems with costs induced by quantitative semantics.

Inspired by formal verification techniques, this paper investigates how learning from demonstration can be applied in the context of planning with LTL. We propose an architecture that learns to distinguish between system executions that satisfy the user-defined requirements and undesired executions from provided demonstrations. The architecture extends the visual action planning framework in [1] by integrating binary classifiers to evaluate the satisfaction of the constraints without sacrificing the benefits of data-driven low-dimensional latent space representations.

B. Contributions and structure of the paper

To enable designers to easily specify task constraints in visual planning frameworks, this work

1) learns task constraints from demonstrations using LSTMs;
2) demonstrates the accuracy of the constraint evaluation in the latent space representation;
3) uses the structure of linear temporal logic to compose complex constraints; and
4) extends the visual-action planning framework in [1] to allow jumps between different latent states so that humans can interact with the system, e.g., by removing boxes;

The remainder of the paper is organized as follows: Sec. II provides necessary preliminaries. Subsequently, Sec. III presents how to learn constraints and their integration in the visual-action planning framework presented in [1]. In Sec. IV, we perform a detailed analysis of our approach and demonstrate its benefits on a box pushing task with human interaction. We finish with conclusions in Sec. V.

II. Preliminaries

In this section, we introduce the visual planning framework in [1] that is used to demonstrate our approach and the formalism of LTL.

A. Visual Action Planning

Given start and goal images, a visual action planning framework generates a path in the image space, that is a sequence of images representing intermediate states, and computes actions to control the robot between them. Directly operating in the image space $\mathcal{I} \subset \mathbb{R}^w \times \mathbb{R}^h \times \mathbb{R}^3, w, h \in \mathbb{N}_+$, where w and h are the width and height of the images respectively, to solve planning problems is computationally intractable due to the size of the space. To reduce the complexity, the problem can be solved in a lower dimensional latent space $\mathcal{Z} \subset \mathbb{R}^d, d \in \mathbb{N}_+$ instead, which is usually much smaller than $\mathcal{I}$, i.e., $d \ll wh$. This latent space $\mathcal{Z}$ encodes a compact representation of the image space $\mathcal{I}$. To learn the latent space and plan in it, we consider the visual-action framework in [1]. This framework consists of two components, the visual foresight module (VFM) and the action proposal network (APN).

The VFM consists of a trained variational autoencoder with augmented loss function (VAE) and a latent space roadmap (LSR). The latent space of the VAE corresponds

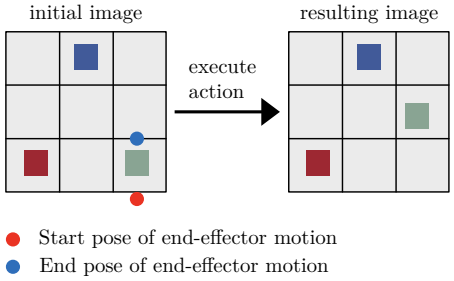

Fig. 2. Example of image transition for a given action. The robot executes a given action, which changes the scene from the initial image to the resulting image.

to $\mathcal{Z}$ and is trained using a loss term that captures the reconstruction error of the images and an additional action term to form valid regions and consider noisy state representations. An example of an image pair and corresponding action is illustrated in Fig. 2. Valid regions in the latent space are clusters of states $z_1, z_2 \in \mathcal{Z}$ that are at most at $\epsilon \in \mathbb{R}_+$ distance, i.e., $|z_1 - z_2|_1 < \epsilon$, where $|\cdot|_1$ is the L1-norm and ϵ is a task-dependent parameter. Building these valid regions allows one to consider the uncertainty induced by imprecision in action execution and generating a valid visual plan. The LSR is a graph-based structure that globally captures the latent system dynamics. It builds on the idea that each node in the roadmap is associated with a valid region, and two nodes are connected by an edge if there exists an action pair in the training dataset to transit from one region to another. For planning, given start and goal image, the VFM produces a visual plan $p_{\mathcal{I}} = (I_0, I_1, \ldots, I_G), I_i \in \mathcal{I}$, as a sequence of images. The sequence $p_{\mathcal{I}}$ is a decoded latent plan $p_{\mathcal{Z}} = (z_0, z_1, ..., z_N), z_i \in \mathcal{Z}$, obtained through graph search in the LSR. The APN takes a pair $(z_i; z_{i+1})$ of consecutive latent states from the latent plan P_z produced by the VFM and proposes an action u_i to achieve the desired transition.

The two components combined produce a visual action plan that can be executed by any suitable framework. If open loop execution is not sufficient for the task, a re-planning step can be added after every action by substituting the start state with the current state and generating a new visual plan with corresponding action plan. More details can be found in [1].

B. Linear Temporal Logic

We use LTL and its structure to synthesize and compose constraints with temporal properties. LTL is expressive enough to capture a rich spectrum of properties, including safety, liveness, and more complex combinations of Boolean and temporal statements [11]. Since we operate on paths with finite length, we consider the finite version of LTL, called LTL$_f$ [16], [17]. For simplicity, we use the terms LTL and LTL$_f$ synonymous.

LTL formulas are constructed from a set of observations O (also called outputs), Boolean operators, and temporal operators. The standard notation for the Boolean operators denotes True by $\top$, False by $\bot$, the negation by $\neg$ and the conjunction by $\wedge$; while for the graphical notation of the temporal operators we denote *Next* operator by $\bigcirc$ and *Until* operator by U. Formally, the syntax of LTL$_f$ coincides with LTL syntax and it is defined as follows:

Definition 1 (LTL$_f$ Syntax): A (propositional) Linear Temporal Logic (LTL$_f$) formula φ over a given set of observations O is recursively defined as

$$\varphi ::= \top \mid o \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \vee \varphi_2 \mid \neg\varphi \mid \bigcirc\varphi \mid \varphi_1 U \varphi_2 \quad (1)$$

where $o \in O$ is an observation and φ, φ_1 and φ_2 are LTL$_f$ formulas, and $\bigcirc$ and U are the temporal operators *next* and *until*, respectively. In addition, the temporal operators *Eventually* $\Diamond$ and *Always* $\Box$ are defined as follows:

$$\begin{aligned} \Diamond\varphi &:= \top U \varphi \\ \Box\varphi &:= \neg\Diamond\neg\varphi \end{aligned} \quad (2)$$

The semantics of LTL$_f$ is interpreted over *finite traces*, that is finite sequence of consecutive instants of time. Let us denote a finite sequence as π over the alphabet 2^O (set of all the subsets of O) and the *length* of the trace π as $length(\pi)$. We denote the positions on the trace as $\pi(i)$ with $0 \leq i \leq length(\pi) - 1$, and the segment of π starting from position i and terminating in position j as $\pi(i, j)$.

Definition 2 (LTL$_f$ Semantics): The satisfaction of LTL$_f$ formula φ over the finite trace π at position $i \in \mathbb{N}_+$ is denoted as $\pi, i \models \varphi$ and recursively defined as follows:

- $\pi, i \models \varphi$ iff $\pi(i) \models \varphi$,
- $\pi, i \models \neg\varphi$ iff $\pi, i \not\models \varphi$,
- $\pi, i \models \varphi_1 \wedge \varphi_2$ iff $\pi, i \models \varphi_1$ and $\pi, i \models \varphi_2$,
- $\pi, i \models \bigcirc\varphi$ iff $i < lenght(\pi) - 1$ and $\pi, i + 1 \models \varphi$,
- $\pi, i \models \varphi_1 U \varphi_1$ iff there exist j such that $i \leq j \leq length(\pi) - 1$, we have $\pi, j \models \varphi_2$ and, for all k such that $i \leq k < j$, we have $\pi, k \models \varphi_1$,

A formula φ is True in π, or in other words π is valid, if $\pi, 0 \models \varphi$. The LTL$_f$ semantics are slightly different than the classical LTL formulas. In particular, the *safety* formula $\Box\varphi$ means that always till the end of the trace φ holds, while the *liveness* formula $\Diamond\varphi$ that eventually before the end of the trace φ holds.

III. Constraints in Visual Planning

We are interested in evaluating whether a given visual-action plan (see Sec. II-A) adheres to a given set of desired properties, e.g., certain undesired box configurations are avoided. In this regard, we want to add constraints expressed as LTL to the planning problem so that computed paths adhere to the properties. In this context, the observations o in LTL are Boolean variables that indicate whether the sequence of images at position $i \in \mathbb{N}_+$ respects a particular condition.

Problem 1 (Problem statement): Given a finite sequence of observations π, an LTL formula φ and the set of all sequences $\mathcal{L}_\varphi = \{\omega \in 2^O \mid \omega \models \varphi\}$ that comply with φ, we are interested in evaluating if π belongs to the set $\mathcal{L}_\varphi$.

A. Learning constraints from demonstrations

Since we do not have an explicit model in visual planning, we cannot evaluate observations analytically (see Prob. 1).

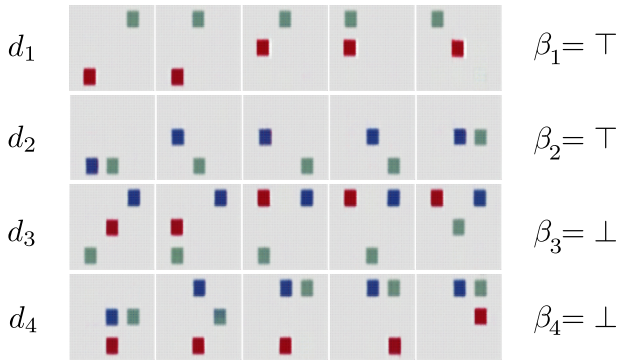

Fig. 3. Example demonstrations d_i for the property "*no green box in the center*" with labels β_i.

Instead, we propose to learn binary classifiers that evaluate whether a given plan satisfies simple properties from demonstrations. These demonstrations $\mathcal{D}$ are sequences of images I and labels β, i.e., $\mathcal{D} := \{(I_0, \ldots, I_n, \beta), n \in \mathbb{N}_+, \beta \in \{\top, \bot\}\}$, that can be recorded by observing humans performing the task. The human provides positive and negative demonstrations that respect or violate a certain property (indicated by the label β). Our approach enables human workers to compose more complex constraints out of simple properties, e.g., a certain box configuration is avoided (see Sec. III-B). Examples of demonstrations are illustrated in Fig. 3.

However, learning the classifiers in image space is computationally intractable due to the sheer size of images and the length of demonstrations. For computational efficiency, we exploit the fact that when we encode positive or negative demonstrations in the latent space $\mathcal{Z}$, they will still be positive or negative examples, i.e., the label β remains unchanged. This property allows to enable non-experts to specify task constraints in image space while performing the classification in the much smaller latent space.

To achieve high accuracy for path classification, we make use of deep learning architectures as these are successful in solving binary classification problems. The class of neural networks which has proved to be well suited to sequential data processing is the family of Recurrent Neural Networks (RNNs) [18]. In general, the key concept in the design of RNNs is the use of cycles, which allows information to persist and to be passed from one step of the network to the next one. This property is crucial because a specific piece of information can occur at multiple positions within the sequence. In contrast, fully connected feed-forward neural networks are not designed to guarantee the persistence of information along a sequence. Long Short-Term Memory (LSTM) [19] is a particular recurrent neural network architecture that has been designed to address the vanishing and exploding gradient problems which occur when training conventional RNNs with the gradient-based back-propagation through time technique [20]. LSTMs have been successfully applied to sequence prediction and sequence labelling tasks and furthermore they have shown to perform better than traditional RNNs on learning context-free and context-sensitive languages [19].

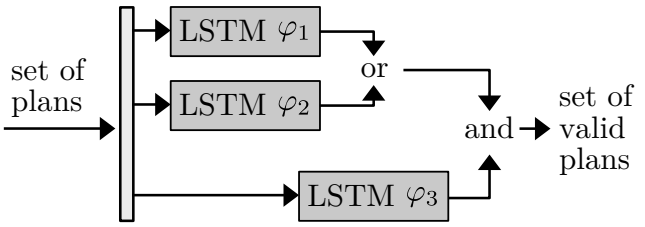

Fig. 4. Learned LSTMs for properties φ_i can be combined with Boolean operators to compose more complex constraints for planning.

The design of LSTMs is based on internal mechanisms called *gates* that regulate the flow of information to be kept or discarded at each time step. Furthermore, LSTMs store indefinite temporal contextual information by using short-term memory variables (hidden state) and long-term memory variables (cell state).

B. Composition of constraints

More sophisticated constraints are usually obtained by composing simple observations in combination with the Boolean operators of LTL. Instead of deploying a new network for each possible combination, a more convenient approach is to re-employ the networks that have been already trained to assess the simple observations individually. Therefore, the evaluation of the composed constraint is performed by elaborating the evaluations of single LSTMs (see Sec. IV-D).

Formally, by applying the Boolean operators in (1) to a set of formulas φ_i, we can compose new constraints φ_{new}. These constraints are then implemented by converting the formula to a tree structure, in which each leaf corresponds to the evaluation of an observation. These observations can be replaced by the corresponding LSTM. The constraint is evaluated by providing the plan to each LSTM, which assess the corresponding φ_i separately. Fig. 4 illustrates this procedure for the formula $((\varphi_1 \vee \varphi_2) \wedge \varphi_3)$ that is composed out of three simpler observations φ_1, φ_2 and φ_3. For instance, we are able to specify an *always* constraint by negating a given *eventually* constraint. In this way, we leverage the structure of LTL to compose complex constraints while keeping the effort for recording demonstrations low for human workers.

C. Visual planning with task constraints

Algorithm 1 summarizes the steps to compute visual-action plans that fulfil a set of formulas φ_i. We first encode the current image of the scene I_s and the goal image I_g in the latent space $\mathcal{Z}$. Afterwards, we determine all possible simple plans from z_s to z_g in the LSR. For each plan p, we assess whether it fulfils all properties φ_i and select one of the paths that are compliant to every φ_i. Using the APN, we determine the robot actions for the chosen valid plan $p^\star$.

IV. Evaluation

To demonstrate our approach, we choose a task similar to the one in Fig. 1 that is inspired by warehouse tasks and sum-

Algorithm 1 Visual-Action Planning with Constraints

Require: I_s, I_g, start and goal images of the scene, and max sequence length M

Encode I_s, I_g into the latent space: z_s, z_g
Detect to which nodes of the Roadmap z_s, z_g belong to
Generate all simple paths from source to goal $\{p_{\mathcal{Z}}\}_{s \to g}$
$\mathcal{P}_{\text{valid}} \leftarrow \emptyset$
for each $p_{\mathcal{Z}} \in \{p_{\mathcal{Z}}\}_{s \to g}$ **do**
    for each LTL$_f$ φ_i **do**
        Assess whether $p_{\mathcal{Z}}$ satisfies φ_i
        if $p_{\mathcal{Z}}$ valid **then**
            $\mathcal{P}_{\text{valid}} \leftarrow \mathcal{P}_{\text{valid}} \cup \{p_{\mathcal{Z}}\}$
        end if
    end for
end for
Select one of the valid latent plans $p^*_{\mathcal{Z}}$
for each (z_i, z_{i+1}) in $p^*_{\mathcal{Z}}$ **do**
    Propose an action u_i to achieve the desired transition from z_i to z_{i+1}
end for

marized in Sec. IV-A. Subsequently, we show how to learn two temporal properties for the task from demonstrations and that LSTMs perform the best compared to other classification techniques in Sec. IV-B. In Sec. IV-C, we evaluate how the size of the demonstration dataset influences the accuracy of the classification. Finally, we demonstrate the composition of complex constraints out of simpler observations in Sec. IV-D. Our simulations are performed in the physics simulator pybullet [21], and the source code is publicly available in the git repository[1].

A. Considered task and visual planning setup

Our task is similar to the one illustrated in Fig. 1 in which a robot is pushing boxes in a warehouse setting. The task of the robot is to push boxes to a desired goal position so that a human worker can pick up the box. Moreover, the robot needs to fulfil different temporal constraints during planning. For the sake of simplicity, the workspace is modelled as a 3x3 grid of dimensions $0.45 \times 0.45(m)$ to fit three boxes of size $0.08 \times 0.08 \times 0.09(m)$ and to consider the reachable arm space of the simulated robot.

For visual planning, we make use of the framework presented in [1] and described in Sec. II-A. To create the training dataset, we let the robot push one box at a time in a horizontally or vertically direction (only the cardinal directions) towards cells that are not yet occupied. The actions $u = (p_1, p_2)$ are a pair of coordinates p_1 and p_2, each of which is a tuple of row and column indices. The recorded images and actions are used to train the VAE and generate the LSR.

Since our task involves a human that pickups boxes, we extend the visual-action planning framework to consider arbitrary (but pre-defined) box configurations. In our task, we consider that the grid may contain three or two boxes. The main difficulty of considering multiple box configurations occurs during the training of the VAE. Configurations with three boxes that share similar positions to configurations with two boxes may be incorrectly encoded close to each other in the latent space, and consequently be incorporated in the same cluster when building the LSR; an example is provided in Fig. 5. Note that transitions from 2-boxes nodes to 3-boxes nodes on the roadmap are forbidden, but the human operator could change the configuration causing a jump in the latent state space from one node component to another one. To address this problem, we propose to add contrastive pairs to the training dataset of the VAE. The pairs are randomly selected from the dataset that contains both kind of configurations. As a result, we create an LSR with multiple connected components that correspond to the different box configurations (see Fig. 5). Thus, when the human pickups a box, we can transit the system automatically from one component to another.

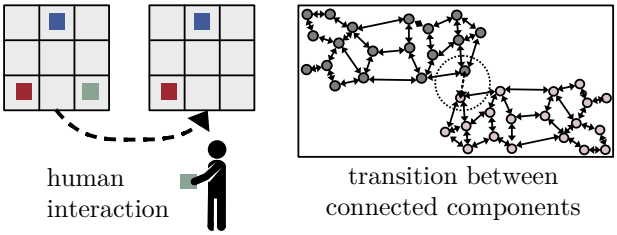

Fig. 5. Example of human interaction in visual planning. The human worker removes the green box which causes a transition from one connected component (representing 3 boxes configurations) to another component (representing 2 boxes configurations).

Our dataset consists of 5000 tuples of two images of dimensions $256 \times 256 \times 3$ and action u. Roughly 72% are 3-boxes and 28% 2-boxes configurations. The position of each box in the grid cell is generated by adding noise (uniformly drawn from $[-0.012.5, +0.0125](m)$) to the center of the cell along each axis. The dataset is augmented by adding 2000 contrastive pairs to address different box configurations. The VAE is trained for 125 epochs on the training dataset of 5500 randomly chosen tuples and tested on the remaining data. Similarly, we train the APN for 150 epochs on the latent training dataset of 3815 tuples, derived from the image dataset pre-processing it with the encoder of the trained VFM, and test it on a dataset of 1008 tuples. The LSR is built with the parameter $w_\epsilon = 1.0$ and contains 831 nodes belonging to 6 connected components, composed by 605, 77, 73, 72, 2, 2 nodes respectively, and 1971 edges.

B. Learning temporal constraints from demonstrations

To demonstrate our approach of learning constraints from demonstrations, we consider our warehouse task and two example constraints that the robot needs to satisfy in this setting. To allow the human worker to pickup boxes, we want the robot to periodically push boxes to the lower row of the grid. Moreover, boxes should not be close to each other for multiple consecutive time steps so that the robot

[1]https://github.com/Francescoes/visual_planning

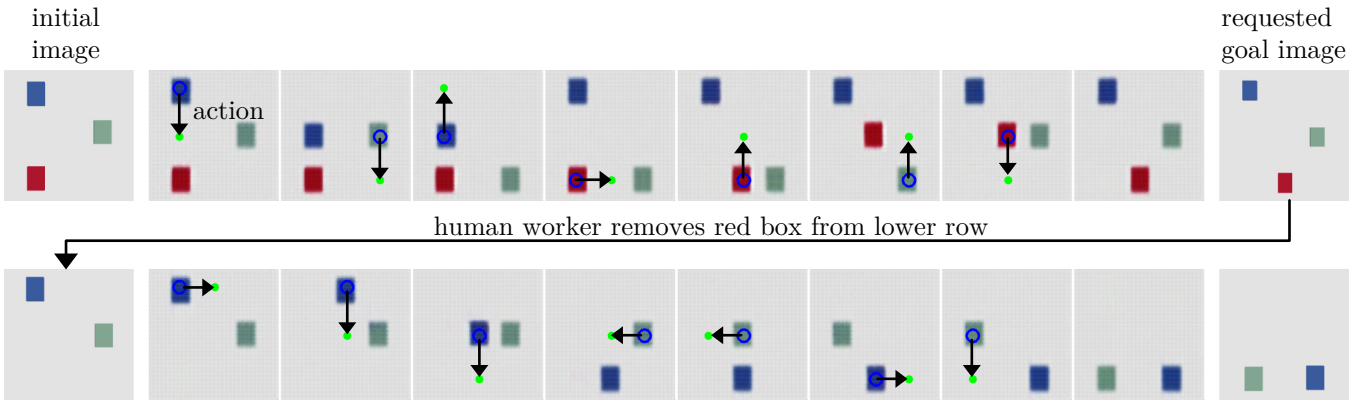

Fig. 6. Planning results when considering both constraints φ_{row} and φ_{adj}. Moreover, the human worker is removing a box from the lower row and our extension automatically switches to the corresponding new configuration of the system.

is able to push boxes without interfering with other boxes. Both task constraints can be summarized as:

1) Always place a box in the lower row at least every three time steps;
2) Always ensure that boxes are in adjacent cells for at most three time steps.

Both requirements can be formulated in LTL as:

$$\varphi_{\text{row}} ::= \Box(o_1 \ \vee \ (\bigcirc o_1 \ \vee \ (\bigcirc(\bigcirc o_1)))), \tag{3}$$

$$\varphi_{\text{adj}} ::= \Box\neg(o_2 \ \wedge \ (\bigcirc o_2 \ \wedge \ (\bigcirc(\bigcirc o_2)))), \tag{4}$$

where the observations o_1 and o_2 denote that at least one box is in the lower row and two boxes are adjacent, respectively.

We generate the datasets of demonstrations $D_{P,\varphi_{\text{row}}}$ and $D_{P,\varphi_{\text{adj}}}$ by synthesizing images with desired and undesired box configurations. These images are encoded in the latent space and then normalized by removing the mean and scaling to unit variance. The synthesis is feasible, since we only need to account for box positions and not for any dynamics of the system. We consider 2000 tuples for $D_{P,\varphi_{\text{row}}}$ and 4000 tuples for $D_{P,\varphi_{\text{adj}}}$. Roughly the 33% of $D_{P,\varphi_{\text{row}}}$ are tuples of 2-boxes configurations sequences, and the remaining 66% are tuples of 3-boxes configurations sequences. For $D_{P,\varphi_{\text{row}}}$ for each start-goal nodes we selected a valid and a non-valid path, so that 50% of paths are valid and 50% are not, instead, for $D_{P,\varphi_{\text{adj}}}$ the percentage of valid paths is higher because 2-boxes paths always satisfy (4). Each dataset is always split in training (70%), validation (20%) and testing (10%) datasets.

To evaluate the performance of LSTMs, we compare them with linear SVM classifiers, non-linear SVM classifiers with a Radial basis function kernel, and a Multi Layer Perceptron (MLP). The LSTM model is composed in sequence by a stacked LSTM layer, a Dropout layer, a Linear layer and a Sigmoid output unit. The hidden state of the stacked LSTM layer contains 12 features, and 4 recurrent layers, so that 4 LSTMs together form the stacked LSTM, with each LSTM taking in the outputs of the previous one. The MLP is composed by a Flatten layer, an input Linear layer with 512 nodes, four Linear layers, with $256, 128, 32, 4$ nodes per layers respectively, followed by rectified linear unit functions, and finally a Sigmoid output unit. The networks are implemented using PyTorch [22].

Since plans may have different lengths, all the sequences are 0-padded to length 8. The input to the MLP and the SVMs are flattened sequences, while the inputs of the LSTM layer are tuples of dimensions (b, ℓ, d), with batch size $b = 200$, sequence length $\ell = 8$, and sequence dimension $d = 64$ to fit the latent space $\mathcal{Z}$.

We train the LSTM, MLP and SVMs classifiers in a supervised fashion on each dataset. Tables I and II report the classification accuracy a, the precision p, the recall r, the specificity s, and the balanced F-score F for each model and each property. The specificity and F-score are:

$$s = \frac{tn}{tn + fp} \qquad F = 2\frac{pr}{p + r}, \tag{5}$$

where tn are true negatives and fp the false positives. In our task, $p = 1.0$ means that $fp = 0$, i.e., every path labelled as valid by the model is in fact valid, whereas a $r = 1.0$ means that $fn = 0$, i.e., every valid path was labelled correctly, and similarly for the specificity.

For both constraints, our results indicate that the LSTM classifier achieves better performance than the other considered classifiers. This performance is most likely a result of the information persistence in LSTMs, which leads to a better generalization of what the network learns during training along several time steps. We added the learned LSTMs to our visual-planning framework according to Alg. 1, so that it is possible to obtain various plans for random initial and goal images that fulfil the two specifications. An example of such a plan where a human is picking up a box is shown in Fig. 6.

C. Requirements on the demonstration dataset

In this paragraph, we investigate how the LSTM classification performance changes when varying the size of the dataset in order to obtain the sufficient number of demonstrations to learn a constraint with a certain degree of accuracy. For this analysis, we consider the *adjacency* constraint (4). To this end, we trained the LSTM for 300 epochs on datasets of variable size randomly generated

TABLE I

CLASSIFICATION PERFORMANCE FOR "BOX IN LOWER ROW" CONSTRAINT (3).

Method	**Classification Metrics**				
	p	r	s	F	a
lin-SVM	0.871	0.816	0.876	0.842	0.847
nonlin-SVM	0.827	0.796	0.830	0.811	0.814
MLP	0.811	0.793	0.814	0.800	0.804
LSTM	**0.956**	**0.907**	**0.960**	**0.931**	**0.934**

TABLE II

CLASSIFICATION PERFORMANCE FOR "ADJACENCY" CONSTRAINT (4).

Method	**Classification Metrics**				
	p	r	s	F	a
lin-SVM	0.573	0.623	0.480	0.596	0.555
nonlin-SVM	0.637	0.645	0.588	0.640	0.618
MLP	0.591	0.691	0.430	0.628	0.571
LSTM	**0.946**	**0.934**	**0.938**	**0.940**	**0.936**

from the $D_{P,\varphi_{\text{adj}}}$. Table III summarizes the classification performance of the LSTM in our experiment. Halving the size of dataset from 2800 to 1400 examples reduces the accuracy by 0.094. In this case, training the LSTM with 1400 demonstrations will provide 84.2% accuracy.

Lastly, we investigate the effect of uneven demonstration datasets, i.e., the distribution of positive and negative examples is uneven. Therefore, we generate two additional datasets $D_{P,\varphi_{\text{row}},asym}$ and $D_{P,\varphi_{\text{adj}},asym}$ for each constraint. However, this time 75% of the demonstrations are positive and 25% not. Table IV illustrates the classification performance for two LSTMs trained on these two datasets with an uneven distribution over the classes. The classification accuracy for (3) and (4) with an uneven distribution over the classes are 0.933 and 0.888 respectively. Even though these values are comparable to the ones with even dataset, the specificity for (4) is much smaller, i.e. 0.674. Since this value indicates that the number of false positives is high, we encourage users to focus on creating evenly distributed demonstration datasets.

D. Composition of constraints

In our final experiment, we demonstrate the composition of complex constraints based on the syntax of LTL and learned simpler constraints. We show how the composition is performed and analyze whether the performance decreases compared to learning the complex constraint directly from demonstrations. Therefore, we consider the following three simple constraints:

1) Eventually place a box in the lower right corner;
2) Eventually place a box in the lower left corner;
3) Always ensure that the green box is not in the center of the grid.

We formalize those constraints in LTL as:

$$\varphi_{\text{rc}} ::= \Diamond o_R \quad \varphi_{\text{lc}} ::= \Diamond o_L \quad \varphi_{\text{cen}} ::= \Box\neg o_c, \tag{6}$$

TABLE III

CLASSIFICATION PERFORMANCE FOR (4) W.R.T. DATASET SIZE.

Size	**Classification Metrics**				
	p	r	s	F	a
2800	0.946	0.934	0.938	0.940	0.936
2450	0.934	0.925	0.920	0.928	0.923
2100	0.905	0.874	0.899	0.889	0.886
1750	0.889	0.877	0.872	0.883	0.874
1400	0.876	0.831	0.857	0.852	0.842

TABLE IV

CLASSIFICATION PERFORMANCE FOR (3) AND (4) TRAINING ON UNEVEN DATASETS

LTL$_f$	**Classification Metrics**				
	p	r	s	F	a
φ_{row}	0.958	0.952	0.881	0.955	0.933
φ_{adj}	0.902	0.956	0.674	0.928	0.888

where o_R signalizes that a box is in the lower right corner, o_L that a box is in the lower left corner, and o_c that the green box is in the center of the image.

Finally, we want the robot to place a box in either the lower left or right corner while never placing the green box in the center of the grid, which corresponds to the constraint:

$$(\varphi_{\text{rc}} \vee \varphi_{\text{lc}}) \wedge \varphi_{\text{cen}} \tag{7}$$

For each simple constraint in (6), we generate a training dataset of 700 examples, and test it on a dataset of 200 data points. To directly learn the composed constraint (7), we use the sizes 1400 and 400 for training and test datasets, respectively. Table V summarizes the classification performance for each simple constraint, the composed one, and directly learning the complex constraint. The accuracy of the composed constraint is comparable to directly learning the complex constraint. Figure 7 shows an example plan obtained for the complex constraint in (7) using the composition approach of three LSTMs.

Table V also reports the classification accuracy for the composition of $(\varphi_{\text{row}} \wedge \varphi_{\text{adj}})$. The results show that the approach based on the composition performs as well as the approach which directly considers the composed constraint, but the composition has several advantages. Simpler constraints can be created with fewer examples and the human does not need to consider multiple constraints at the same time for the demonstrations. Moreover, they allow one create a library of building blocks to compose complex constraints on-the-fly. For instance, by negating φ_{cen}, we can achieve that the robot eventually places the green box in the center.

V. CONCLUSIONS

This work presents how to integrate task constraints in visual action planning frameworks while allowing designers to easily specify, compose and integrate task specifications. We show that LSTMs are appropriate tools to assess the satisfaction of the constraints for a given action plan with

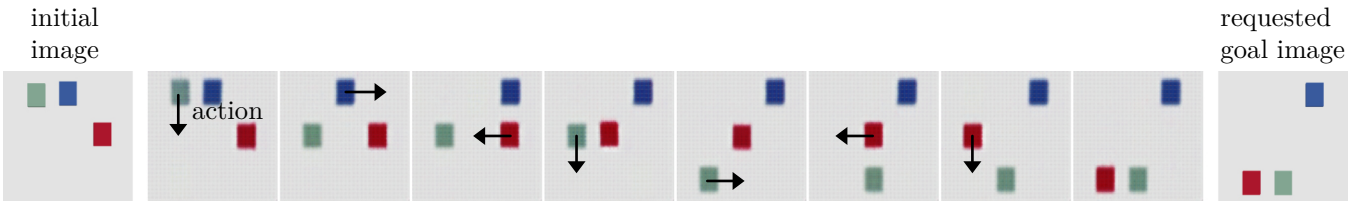

Fig. 7. Planning results for the composed constraint in (7).

TABLE V

ACCURACY FOR SIMPLE AND THE COMPOSED CONSTRAINTS.

LTL$_f$	**Classification Accuracy**
φ_{rc}	0.998
φ_{lc}	0.990
φ_{cen}	0.904
1-LSTM: $(\varphi_{\text{rc}} \vee \varphi_{\text{lc}}) \wedge \varphi_{\text{cen}}$	0.896
3-LSTM: $(\varphi_{\text{rc}} \vee \varphi_{\text{lc}}) \wedge \varphi_{\text{cen}}$	0.909
1-LSTM: $\varphi_{\text{row}} \wedge \varphi_{\text{adj}}$	0.836
2-LSTM: $\varphi_{\text{row}} \wedge \varphi_{\text{adj}}$	0.806

high accuracy directly in the latent space representation. We conducted an analysis of the requirements of the demonstration dataset when varying the size and for unevenly distributed data. Finally, we showed the benefits of our approach in a box pushing task in an interactive warehouse setting. We are able to obtain visual-action plans that satisfy various learned as well as composed constraints, which are created by applying the Boolean operators to the simpler constraints.

REFERENCES

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

[2] M. Huth and M. Ryan, *Logic in Computer Science: Modelling and reasoning about systems*. Cambridge university press, 2004.

[3] D. Hafner, T. Lillicrap, I. Fischer, R. Villegas, D. Ha, H. Lee, and J. Davidson, "Learning latent dynamics for planning from pixels," in *International Conference on Machine Learning*. PMLR, 2019, pp. 2555–2565.

[4] M. Asai and A. Fukunaga, "Classical planning in deep latent space: Bridging the subsymbolic-symbolic boundary," in *Proc. of the AAAI Conf. on Artificial Intelligence*, vol. 32, no. 1, 2018.

[5] B. Ichter and M. Pavone, "Robot motion planning in learned latent spaces," *IEEE Robotics and Automation Letters*, vol. 4, no. 3, pp. 2407–2414, 2019.

[6] M. Watter, J. T. Springenberg, J. Boedecker, and M. Riedmiller, "Embed to control: A locally linear latent dynamics model for control from raw images," *arXiv preprint arXiv:1506.07365*, 2015.

[7] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," *arXiv preprint arXiv:1312.6114*, 2013.

[8] T. Adel, Z. Ghahramani, and A. Weller, "Discovering interpretable representations for both deep generative and discriminative models," in *Int. Conf. on Machine Learning*. PMLR, 2018, pp. 50–59.

[9] E. M. Clarke, T. A. Henzinger, H. Veith, and R. Bloem, *Handbook of model checking*. Springer, 2018, vol. 10.

[10] C. Baier and J.-P. Katoen, *Principles of model checking*. MIT press, 2008.

[11] O. Kupferman and M. Y. Vardi, "Model checking of safety properties," *Formal Methods in System Design*, vol. 19, no. 3, pp. 291–314, 2001.

[12] C. Belta, B. Yordanov, and E. A. Gol, *Formal methods for discrete-time dynamical systems*. Springer, 2017, vol. 15.

[13] O. Maler and D. Nickovic, "Monitoring temporal properties of continuous signals," in *Formal Techniques, Modelling and Analysis of Timed and Fault-Tolerant Systems*. Springer, 2004, pp. 152–166.

[14] R. Koymans, "Specifying real-time properties with metric temporal logic," *Real-time systems*, vol. 2, no. 4, pp. 255–299, 1990.

[15] D. Aksaray, A. Jones, Z. Kong, M. Schwager, and C. Belta, "Q-learning for robust satisfaction of signal temporal logic specifications," in *Proc. of the IEEE Conf. on Decision and Control*. IEEE, 2016, pp. 6565–6570.

[16] G. De Giacomo and M. Y. Vardi, "Linear temporal logic and linear dynamic logic on finite traces," in *Proc. of the Int. joint Conf. on Artificial Intelligence*, 2013, pp. 854–860.

[17] ——, "LTLf and LDLf synthesis under partial observability." in *IJCAI*, vol. 2016, 2016, pp. 1044–1050.

[18] L. R. Medsker and L. Jain, "Recurrent neural networks," *Design and Applications*, vol. 5, 2001.

[19] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[20] Y. Bengio, A. Courville, and P. Vincent, "Representation learning: A review and new perspectives," *IEEE transactions on pattern analysis and machine intelligence*, vol. 35, no. 8, pp. 1798–1828, 2013.

[21] E. Coumans and Y. Bai, "Pybullet, a python module for physics simulation for games, robotics and machine learning," http://pybullet.org, 2016–2019.

[22] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems*, 2019, pp. 8024–8035.