Learning 3D Navigation Protocols on Touch Interfaces with Cooperative Multi-Agent Reinforcement Learning

Quentin Debard, Jilles Steeve Dibangoye, Stephane Canu, Christian Wolf

Learning 3D Navigation Protocols on Touch Interfaces with Cooperative Multi-Agent Reinforcement Learning

Quentin Debard[1], Jilles Steeve Dibangoye[2], Stéphane Canu[3], and Christian Wolf[4]

[1] Itekube, LIRIS, quentin.debard@itekube.com
[2] Inria, CITI-Lab, INSA-Lyon, jilles-steeve.dibangoye@insa-lyon.fr
[3] LITIS, INSA-Rouen, stephane.canu@insa-rouen.fr
[4] LIRIS, Inria, CITI-Lab, INSA-Lyon, christian.wolf@insa-lyon.fr

Abstract. Using touch devices to navigate in virtual 3D environments such as computer assisted design (CAD) models or geographical information systems (GIS) is inherently difficult for humans, as the 3D operations have to be performed by the user on a 2D touch surface. This ill-posed problem is classically solved with a fixed and handcrafted interaction protocol, which must be learned by the user. We propose to automatically learn a new interaction protocol allowing to map a 2D user input to 3D actions in virtual environments using reinforcement learning (RL). A fundamental problem of RL methods is the vast amount of interactions often required, which are difficult to come by when humans are involved. To overcome this limitation, we make use of two collaborative agents. The first agent models the human by learning to perform the 2D finger trajectories. The second agent acts as the interaction protocol, interpreting and translating to 3D operations the 2D finger trajectories from the first agent. We restrict the learned 2D trajectories to be similar to a training set of collected human gestures by first performing state representation learning, prior to reinforcement learning. This state representation learning is addressed by projecting the gestures into a latent space learned by a variational auto encoder (VAE).

Keywords: Multi-Agent, Deep Reinforcement Learning, H-C Interfaces

1 Introduction

The goal in user interface (UI) design is to propose a communication protocol between human users and a given machine that is intuitive, quick, precise, and which minimizes the amount of training required for new users not yet familiar with it. Designing such an interface is not trivial, as some of these desired properties are contradictory. Furthermore, some of these objectives are difficult to quantify, such as intuitivity of the interface or, more generally, user satisfaction.

Our work focuses on a specific component of touch user interfaces, which we call the interaction protocol. This protocol defines the rules that allow the computer to interpret 2D user gestures performed on touch tables into actions in the virtual environment. In the literature, this interaction protocol refers to the software side of an interaction technique [14][21]. In this paper, we address the problem of automatically learning a suitable interaction protocol for graphical user interfaces on touch surfaces, which requires users to manipulate 3D objects, for instance in computer assisted design (CAD)

software or in geographic information systems (GIS). In these situations, the problem is particularly ill-posed, as the trajectories produced by a user on the flat touch screen are restricted to a 2D surface, whereas the applications require the user to perform manipulations in a virtual 3D environment. To give a concrete example, inspecting a virtual mechanical product or navigating in a virtual building or city requires the possibility to change the camera viewpoint through rotations, translations, zooming, i.e. to manipulate 6 degrees of freedom (3 for the camera position and 3 for the camera direction) through trajectories of eventually multiple fingers in the 2D plane of the touch table. There is no universally accepted canonical solution for this kind of problem.

Advanced methods approach this mapping from gestures to actions in a 3D environment using several parameters: not only the 2D gesture themselves, but also the position of the camera (view of the user), the state of the 3D environment, etc. [5,7]. Theoretically, these methods offer more complex manipulation strategies and higher efficiency. However, the challenge here lies in the combination of precision and efficiency on one hand, and ease of use and learnability (by humans) on the other hand.

In this work, we propose to learn these interaction protocols automatically from interactions with humans. The mapping from 2D gestures to actions in the 3D environment is performed by a trainable agent whose policy is learned using reinforcement learning (RL). Such an agent observes user gestures, translates them into actions in the 3D environment and receives a reward, which should be related to user satisfaction in the optimal case. The motivations behind this choice are two-fold:

- to automatically learn complex interactions protocols instead of handcrafting them;
- to create *adaptive* user interfaces, where not only the human users (classically) adapts to the interface, but the computer also adapts to the way the interaction protocol is imagined by the user through online learning during usage.

The main challenge lies in the requirement of massive amounts of interactions, necessary for current RL algorithms, but which are difficult or impossible to come by when humans are involved. Requesting users to provide gestures and feedback on satisfaction at each iteration would be overly complicated and not realistic for any complex application.

In this work, we propose to circumvent this problem by firstly pre-training the RL agent from interaction with a learned user model which is jointly learned with the target agent. The interactions of the user model are statistically constrained to natural interactions collected in a static dataset. Secondly, creating loss / reward signals during this pre-training phase from success measures in standard interaction tasks, e.g. "*go to place X*".

The paper is organized as follows: section 2 discusses related work in reinforcement learning and HCI. Section 3 presents the exact formulation of the HCI problem as a reinforcement learning problem. Section 4 introduces the main contribution of this work: the formulation as a cooperative multi-agent problem, where the user model is jointly learned with the interaction protocol, restricting simulated interactions to natural ones. Two different experimental sections report evaluations of the approach on two scenarii of increasing complexity. Section 5 describes experiments on a simple 2D environment where users manipulate a 2D object on a 2D surface in a similar fashion to the widely known "*Pinch-To-Zoom*" interface developed for smartphones and tablets.

The common solution of this type of environment being known, the objective here is to automatically learn this interaction protocol from interactions instead of handcrafting it. Finally, section 6 describes experiments on the targeted application, namely learning a 2D to 3D interface protocol involving navigation in 3D environments. This application features additional complexities, such as the non-unicity of the solution (discussed in Sec.4) and the impossibility to analytically define an optimal user.

2 Related Work

Our work stands between two active fields of computer science: human-computer interface (HCI) and machine learning (ML). We will shortly describe relevant work in both areas to paint the background.

Adaptive user interface — The goal of Adaptive User Interfaces (AUI) is to adapt its visualization and its interactions to fit individual users' intent better. Machine learning, in this case, is traditionally used for user intent modelization. For an overview of state-of-the-art AUI with adaptive visualization, see [1]. To our knowledge, there is no prior work on learning the interaction protocol of an interface with continous action space.

Reinforcement learning and UI design — Reinforcement learning (RL) is a machine learning framework in which a software agent learns to solve an environment by taking actions that maximizes some cumulative reward. RL has seen some specific uses for AUI design, more precisely for user profiling and representation tasks. In [23], an agent learns to detect user preferences implicitly from observing user behavior instead of direct feedback. In [11], an agent uses user feedback to display personalized web pages.

Machine learning and 3D interaction design — 3D user interface (UI) design has been studied for about 20 years [6]. In the 3D UI context, a good overview of current state-of-the-art 3D UI methods is given in [21]. While ML has been used in user interface and user experience design for about two decades [16][25], using machine learning for interaction design is to the best of our knowledge an application yet to be explored. In the UI context, ML is classically used to improve the accuracy of an existing interaction protocol: in [24], a gaussian process regression is used to improve touch accuracy. In [18][10][8], supervised deep learning is used to improve the recognition rate of some multi-touch gesture classes.

Reinforcement learning and generative models — Combining the latent representations of generative models with policy learning was explored in some recent work. In [13], the encoding part of a modified VAE is used to build disentangled representations for a RL policy to use in domain adaptation tasks. In [20], a VAE together with an agent are trained for different purposes: synthesize training data from real observations of the policy, embed the observations to provide latent representations to the policy and measure reward signals in the latent space.

Learning from demonstration — In our paper, we are trying to learn a user model from a small part of all the possible interactions a user can perform. In [22], a model-based agent is built from examples given by a demonstrator that can either be a generative model, open loop excitation or an expert.

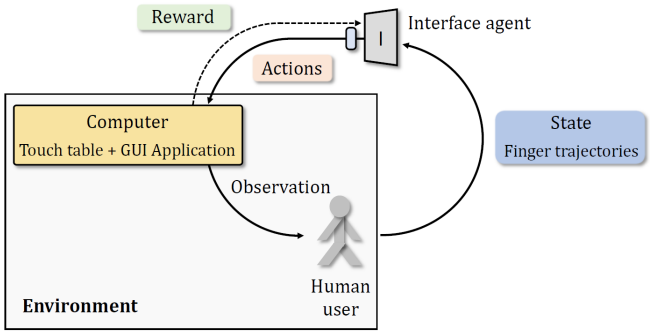

Fig. 1. Learning user interface protocols from interactions with users as an MDP/RL problem.

3 Learning an interaction protocol as a reinforcement problem

Our goal is to learn an interaction protocol coupling user trajectories with actions in a 3D application like CAD or GIS while maximizing the user's satisfaction. We cast this as an RL problem, where the agent gets observations in the form of finger trajectories and outputs actions, which correspond to viewpoint changes in a 3D environment. Because user intentions and gestures can have long term dependencies and depend on multiple latent factors, this problem could be modeled as a partially observable Markov decision process (POMDP). Assessing the complexity of such a modelization for a novel application, we prefer in this paper to consider the problem as a fully observed Markov decision process (MDP) by stacking two-time instants. This implies considering that the information given at two following time instants is enough to predict user intent. The problem is then treated as an MDP with continuous observation and action spaces. The agent A observes the state in the form of finger trajectories s and receives a reward r after performing an action a in the application. An agent A learns a policy π such as $a = \pi(s)$ to maximize its expected return, i.e., the expectation of cumulated reward. Fig. 1 illustrates this situation.

In the RL nomenclature, the agent interacts with an environment, which, in our case, corresponds to, both, the human user and the application, e.g. a CAD or GIS software (see Fig. 1). The agent observes a state, i.e. the user's finger trajectories, and then performs actions that change the viewpoint in the 3D software and which lead to a new state (new user gestures). The agent also receives feedback in the form of a reward. It is important to note here that user satisfaction is difficult to measure directly if we do not want the resort to solutions which estimate emotions from facial expressions. In the next sections, we will propose proxy metrics which approximate satisfaction.

4 Jointly learning the interface protocol and human behavior

Deep networks require large-scale training datasets, and Deep RL is not an exception. More so, RL requires dynamic data in the form of interactions, typically millions or billions when observations are of high dimensions and/or when the regularities are

complex. In robotics, where interactions with physical robots are slow (not faster than physical time) and expensive, this leads to the tendency of training from simulations, for instance [2][3] for robot navigation and [26] for grasping, and to the sim-to-real transfer problems [26].

Similar to robotics, learning from human interactions is limited. It is restricted to physical real-time, and the effort required from humans during training is to be taken into account. More so, human time is expensive. For these reasons, we address this by simulating the environment, which in our case also involves simulating the human user. However, while simulating robots through handcrafted solutions is feasible, at least approximately, human behavior is inherently difficult to model. For this reason, we propose a formulation where human behavior is learned jointly with the interface task itself. The next two sub sections describe the two main challenges for this task: (i) restricting the learned user behavior to realistic human gestures (sub section 4.1), and (ii) solving the joint learning problem (sub section 4.2).

4.1 Learning the manifold of natural human gestures

Let $x \in \mathcal{X}$ be an observation in the form of natural two-finger trajectories performed by a human user. We define an observation as a N-length sequence of 4-tuples, each 4-tuple consists of a pair of coordinates (x, y), one for each of the two fingers. $\mathcal{X}=\mathbb{R}^{4N}$ is the space of observations of length N. The gesture space $\mathcal{X}$ thus covers all possible pairs of 2D trajectories, including trajectories which are anatomically impossible to perform by human fingers. Our objective is to learn a subspace which corresponds to gestures naturally performed by humans. To this end, we suppose the existence of a training dataset of natural gestures $X = \{x_i\}$, which have been collected from user interactions. This training data can be collected without any manual annotation as simple interaction traces.

We want to build a model capable of producing any natural gesture x from a latent representation z. Sampling values from z should provide us samples of the manifold of natural human gestures. This involves learning a distribution $p(x|z)$ and to be able to evaluate it from a given z. Restricting our simulated user to produce samples of the latent representation z should therefore restrict it to produce natural gestures.

Several approaches exist for learning generative models of probability distributions from training data, among which are Generative Adversarial Networks (GANs) [12] and Variational Auto-Encoders (VAEs) [15]. Our definition of $p(x|z)$ can be related to the generative part of a GAN or the decoder part of a VAE. In this work, we chose VAEs for two reasons: they are simpler to train and less sensitive to hyperparameters; and the latent space is smoother, due to its soft constraint to be close to a multivariate Gaussian.

The VAE is trained on the dataset X, approximating the distribution $p_\theta(x)$ by measuring the reconstruction error on a sample x_i coded by an encoder E into a code z_i, then reconstructed into $\hat{x_i}$ using a decoder D. To describe the problem from a probabilistic point of view, the probability $p_\theta(x)$ of a sample x can be decomposed into a prior and a likelihood as:

$$p_\theta(x) = \int p_\theta(x|z)p_\theta(z)dz \tag{1}$$

where the prior on z is defined as a standard Gaussian distribution $p_\theta(z) = \mathcal{N}(\mathbf{0}, \mathbf{I})$. In our case (continuous values), we can assume that the likelihood is Gaussian distributed:

$$p_\theta(x|z) = \mathcal{N}(x|D(z, \phi_d), \sigma^2\mathbf{I}) \tag{2}$$

where $D(z, \phi_d)$ is the decoder of the VAE. The integral is difficult to evaluate, but can be approximated by a point estimate $z{=}q_\phi(x)$ from the variational distribution q:

$$p_\theta(x) \approx \mathcal{N}(x|D(q_\phi(x), \phi_d), \sigma^2\mathbf{I}) \tag{3}$$

$q_\phi(z|x)$ can be seen as an encoder, noted $E(x, \phi_e)$. In this case, we need to ensure that $q_\phi(z|x)$ is a good estimate of the true posterior $p_\theta(z|x)$. This is done using the Kullback-Leibler divergence, noted D_{KL}. Considering the approximation error for only a sample x_i, the KL divergence becomes $D_{KL}(E(x_i, \phi_e)||p(z))$. As stated earlier, $p_\theta(z) = \mathcal{N}(\mathbf{0}, \mathbf{I})$. If we use the L_2-norm to measure the reconstruction error, the total error can be written as a variation of the evidence lower bound (ELBO):

$$ELBO_i = ||x_i - D(E(x_i, \phi_e), \phi_d)||_2 - \beta\, D_{KL}(E(x_i, \phi_e)||\mathcal{N}(\mathbf{0}, \mathbf{I})) \tag{4}$$

where β is a parameter allowing us to adjust the tradeoff between the reconstruction precision and the latent space regularity [19]. We can then update ϕ_e and ϕ_d by minimizing this error using back-propagation.

4.2 Cooperative Multi-Agent RL

We formulate the task of jointly learning the user interface protocol and human behavior as a cooperative multi-agent reinforcement (MARL) problem, as shown in Fig. 2. Two agents are learned jointly, each with its own policy:

- agent A_i corresponds to the user interface. Learning its policy is the original goal of this work, as this agent is responsible for translating 2D finger gestures into actions in the 3D environment (CAD or GIS software).
- agent A_u corresponds to the simulated user, with which agent A_i interacts. The only purpose of A_u is to replace human users during the costly pre-training phase. In contrast to A_i, A_u is discarded after training.

These two agents are trained to maximize the same objective function, sharing the same reward (detailed in section 4.3), which makes this problem a *cooperative* multi-agent problem. Only A_i directly takes action in the virtual 3D environment, whereas A_u acts indirectly by producing the input of A_i.

Learning both agents by maximizing the joint reward without additional constraints could naturally lead to degenerate solutions, which are efficient (allow to navigate quickly), but where the gestures exchanged between the two agents are artificial and not easily and naturally doable by humans. For this reason, we restrict the exchange between A_i and A_u to a representation z learned by the VAE described in section 4.1. More precisely, after training the VAE, we discard its encoder. The agent A_u learns a policy on an action space which corresponds to the latent representation z. Each action z is then decoded to a natural gesture x through the decoder of the learned VAE, as

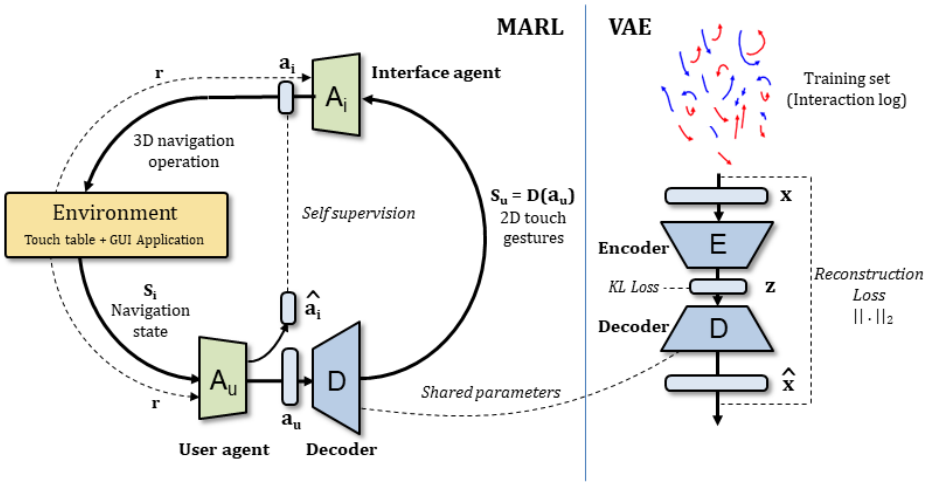

Fig. 2. Cooperative multi-agent RL problem for jointly learning user interface protocol and user behavior. Generative models are blue and RL policies are green (best viewed in color).

illustrated in Fig. 2. In other words, *the policy of the user agent learns to produce gestures by navigating the latent space of the VAE*. For readability, for the rest of the paper, we will refer to the interaction protocol agent as the interface agent A_i, and to the RL agent sampling in the VAE latent space as the user agent A_u. The combination of the user agent and the decoder will be called user model U.

The method can be more formally described as follows. The task is a sequential cooperative setup where A_u produces the state of A_i and A_i does not get any observation of the virtual environment. In what follows, we denote $s_\cdot^t$ as the state of an RL agent at time t, $a_\cdot^t$ as the action at time t and $r_\cdot^t$ as the resulting reward from action $a_\cdot^t$. $\pi_\cdot$ will denote an agent policy. All symbols are indexed by subscripts u or i, which stand, respectively, for the agent A_u and A_i. Let θ^t be the state of the software environment at time t, for instance the viewpoint in a building, or the 6D pose of a mechanical object in a CAD problem. Then, a given time step t in our sequential cooperative MARL setup will unroll as follows:

$$\begin{aligned} s_u^t &= \theta^t \\ a_u^t &= D(\pi_u(s_u^t)) \\ s_i^t &= a_u^t \\ a_i^t &= \pi_i(s_i^t) = \Delta\theta^t \\ s_u^{t+1} &= \theta^{t+1} \\ r_u^t &= r_i^t = r(a_i^t, s_u^{t+1}) \end{aligned} \tag{5}$$

where $\Delta\theta^t$ is a variation of the parametrization of the object, and $r_u^t{=}r_i^t$ is the joint reward at time t. We can note that the constraint on A_u actions defined by D is mandatory for our setup to converge to a cooperative setup. Indeed, supposing that we directly have

$s_i^t = \pi_u(s_u^t)$), the policy of one of the agents will degenerate to an identity, effectively lowering the complexity of the problem by getting rid of the intermediate representation between the two agents. One agent will end up observing and taking actions directly in the virtual environment, breaking the paradigm of this setup.

4.3 Defining the reward function

The goal of this work is to optimize user satisfaction during interactions, which is not easily measurable. We can attempt to empirically break it down to a set of less subjective parts: precision of the interactions, expressiveness, intuitivity and ease of use. The latter two are difficult to measure directly but can be added as learned soft constraints to the agent. In this work, we make the assumption that the latent representation learned by the VAE from interaction logs encodes intuitivity and ease of use, leveraged by restricting the user agent A_u to an action space defined as the latent representation of the VAE.

The former two (precision and expressiveness) are performance metrics related to the environment the agents are trying to solve. If we restrict ourselves to training from situations where the objective of the HCI experiment is known, these measures can be optimized directly by defining an appropriate reward function. As examples we could imagine asking users questions like "*Find Waldo in this building by navigating there"* or "*view the carburator of this V6 engine from above allowing to see inside it*". The downside to this approach is restricting training to situations with known outcomes and objectives. This still allows learning interfaces in a co-adaptive fashion, in two consecutive stages: a first off-line training stage on a set of "training users", followed by an enrollment training phase, where each user is asked to solve custom scenarii to adapt the system to its own interface behavior. It does *not*, however, allow continuous adaptation during usage with unknown objectives.

We will detail our chosen reward functions in the experimental section. It suffices to say at this point, that they measure a distance to the goal in the given user defined task. However, it is important to remember that our true goal is not for the agents to maximize their rewards, which only partially relate to user satisfaction. Convergence of cumulated reward is a necessary condition for a good solution, but not sufficient. Only humans can assess the true quality of these interaction protocols.

4.4 Stabilizing learning with self-supervision

In the standard formulation as described above, during training, the interface agent A_i learns to interpret actions produced by the user agent A_u, while A_u does not get any (unfiltered) information from the interface and as such cannot infer how the interface will interpret a gesture it produces. Furthermore, since the interface policy π_i evolves during training, the target for the user agent A_u is unstable, which makes training difficult.

We stabilize training by forcing A_u to approximate decisions taken by the interface A_i in the form of self-supervision. We add a second predictor head to A_u, which predicts the output of A_i. This predictor shares common layers with the classical predictor of the policy π_u, which ensures that the learned feature representation benefits both predictors. The new predictor is supervised with the real output of the interface agent

A_i using the L_2 loss $\mathcal{L}_e = ||a_i - \hat{a}_i||_2$. $\hat{a}_i$ is the interface action predicted by A_u, and a_i is the interface action predicted by A_i. In practice, this loss only affects the user actor π_u, the critic taking no part in this estimation. This is an external training signal added to the RL signal coming from the critic estimation of the state-action value function Q. As a note, adding a coefficient to the new loss to control its impact did not yield any meaningful improvements.

5 Experiments 1: a simple problem — solving "Pinch-to-Zoom"

As a first proof of concept, we will attempt to solve a well-known continuous HCI, for which a handcrafted solution does exist, the goal being to verify whether learning can discover the existing solution. Our choice here is the well known "Pinch-to-Zoom" interface widely used for smartphones and tablets. The name is a misnomer, since the interface not only allows to zoom, but also to translate and rotate the content of surface through 2D gestures made by two fingers. We suppose that a user performs gestures with exactly two fingers on a touch screen and we investigate the motion between two different time instants. We denote by $\boldsymbol{l} = [l_x\ l_y]^T$ the screen coordinates of a single finger at the first instant and by $\boldsymbol{l}' = [l'_x\ l'_y]^T$ the coordinates at the second instant. If we need to explicitly identify a finger, we will index finger i with a superscript as in $\boldsymbol{l}^i$ or $\boldsymbol{l}'^i$. The coordinates are normalized between $[0\ 0]$ (top-left) and $[1\ 1]$ (bottom-right).

The known solution — We will first derive the analytical form of the known solution before describing the experiments learning it. The gestures performed by the user are a combination of translation, rotation and scaling. We suppose that the 2D finger motion on the screen induces the same 2D motion of the manipulated surface, which can be seen as a special case of affine transformation where the shear component is zero. It transforms coordinates $\boldsymbol{l}$ into $\boldsymbol{l}'$ as $\boldsymbol{l}' = \boldsymbol{A}\boldsymbol{l} + \boldsymbol{t}$ where $\boldsymbol{t} = [\boldsymbol{t}_x\ \boldsymbol{t}_y]^T$ is the translation component and the rotation+scaling matrix can be calculated from the rotation angle α and the scaling factor σ as follows:

$$\boldsymbol{A} = \begin{bmatrix} \cos\alpha & -\sin\alpha \\ \sin\alpha & \cos\alpha \end{bmatrix} \begin{bmatrix} \sigma & 0 \\ 0 & \sigma \end{bmatrix} = \begin{bmatrix} \sigma\cos\alpha & -\sigma\sin\alpha \\ \sigma\sin\alpha & \sigma\cos\alpha \end{bmatrix} \tag{6}$$

The 4 parameters of the motion are thus $\alpha, \sigma, \boldsymbol{t}_x, \boldsymbol{t}_y$, which we will combine into a parameter vector $\theta = [\sigma\cos\alpha\ \ \sigma\sin\alpha\ \ \boldsymbol{t}_x\ \ \boldsymbol{t}_y]^T$. If we have motion of two different fingers $(\boldsymbol{l}^1, \boldsymbol{l}'^1)$ and $(\boldsymbol{l}^2, \boldsymbol{l}'^2)$, then the following linear relationship between the coordinates and the parameter vector θ holds: $\boldsymbol{d} = \boldsymbol{D}\theta$, where $\boldsymbol{d}$ is a vector containing the target coordinates and $\boldsymbol{D}$ is a matrix containing the source coordinates in a suitable form:

$$\begin{bmatrix} l'^1_x \\ l'^1_y \\ l'^2_x \\ l'^2_y \end{bmatrix} = \begin{bmatrix} l^1_x & -l^1_y & 1 & 0 \\ l^1_y & l^1_x & 0 & 1 \\ l^2_x & -l^2_y & 1 & 0 \\ l^2_y & l^2_x & 0 & 1 \end{bmatrix} \begin{bmatrix} \sigma\cos\alpha \\ \sigma\sin\alpha \\ t_x \\ t_y \end{bmatrix} \tag{7}$$

Because $\boldsymbol{D}$ is always invertible (except for the degenerated case where both fingers are at the origin), this linear equation can be solved easily as $\hat{\theta} = \boldsymbol{D}^{-1}\boldsymbol{d}$.

Learning a solution — We now let an RL agent learn this protocol. Let the state of the agent be a two-finger motion $\boldsymbol{s}_i = [\boldsymbol{l}^1_x\ \ \boldsymbol{l}^1_y\ \ \boldsymbol{l}^2_x\ \ \boldsymbol{l}^2_y\ \ \boldsymbol{l}'^1_x\ \ \boldsymbol{l}'^1_y\ \ \boldsymbol{l}'^2_x\ \ \boldsymbol{l}'^2_y]$ performed

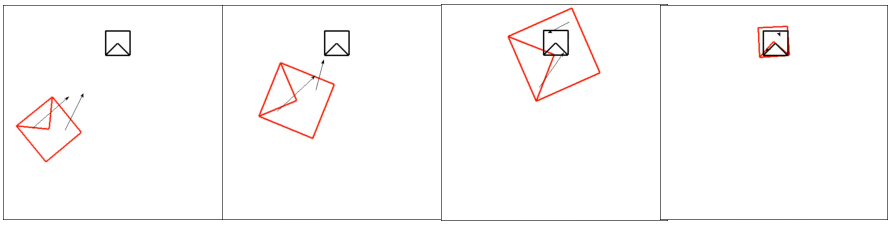

Fig. 3. An example rollout of the interface policy learned for the "Pinch-To-Zoom" problem. The rectangle need to be superimposed, finger trajectories are indicated by arrows.

by the user, where superscripts index fingers and subscripts indicate x or y coordinates. Agent actions $\boldsymbol{a}_i = [\sigma\cos\alpha \;\; \sigma\sin\alpha \;\; \boldsymbol{t}_x \;\; \boldsymbol{t}_y]^T$ are continuous vectors of size 4, which correspond to the parameters of an affine motion transformation without shear component. Note that while this affine transformation has the same functional form as the one expressed in the analytical solution above, we here describe output motion only (motion the manipulated object will endure) and not input finger motion. In other words, in this RL scenario, we do *NOT* suppose that object motion equals finger motion.

The environment is a simple scenario, where a user is required to move a virtual surface containing an object (a red rectangle). The goal is to bring this object to a fixed position by superimposing it on an object which does *not* move with the manipulated surface, i.e. a black rectangle "painted" on the glass of the device. The reward function in this task is the sum of L_1- and L_2-distances the object vertices and the target vertices:

$$r = -\sum_i \left(||\boldsymbol{o}_i - \boldsymbol{t}_i||_1 + ||\boldsymbol{o}_i - \boldsymbol{t}_i||_2 \right) - 0.2 \tag{8}$$

where $\boldsymbol{o}_i$ and $\boldsymbol{t}_i$ are the coordinates of $i-th$ vertice of the respective rectangle. Let us recall that the interface agent does *not* have access to these positions, else it would learn to simply ignore user gestures. The constant -0.2 reward is set to continuously encourage fast solutions. A positive reward of $+25$ is given if the agent successfully finishes an episode. These arbitrary values are chosen so that the expectation of the sum of rewards per episode is close to 0 for an agent close to the optimal solution.

Handcrafted simulation of the user — For this toy problem, the cooperative multi-agent formulation proposed in section 4 is not necessary. We instead handcraft a solution simulating a user who is aware of the "Pinch-To-Zoom" protocol, i.e. of the known analytical solution. We would like to stress that the agent is of course *not* aware of the solution. The interface solution expressed in 7 allows us to compute simulated user trajectories: this can simply be done by considering two diagonally opposed object vertices v_1 and v_2 and their target position v'_1 and v'_2. For a sampling time, we can consider that the user will move the object toward the target while keeping the vertices v_i on the segments $[v_i,\ v'_i]$ (which is the optimal way to solve the task). It means we can find intermediate positions of v_i on these segments using the linear combination:

$$v_i^{inter} = (1-\mu)v_i + \mu v'_i \ , \ \mu = \max(1, \frac{0.5}{||v'_i - v_i||_2}) \tag{9}$$

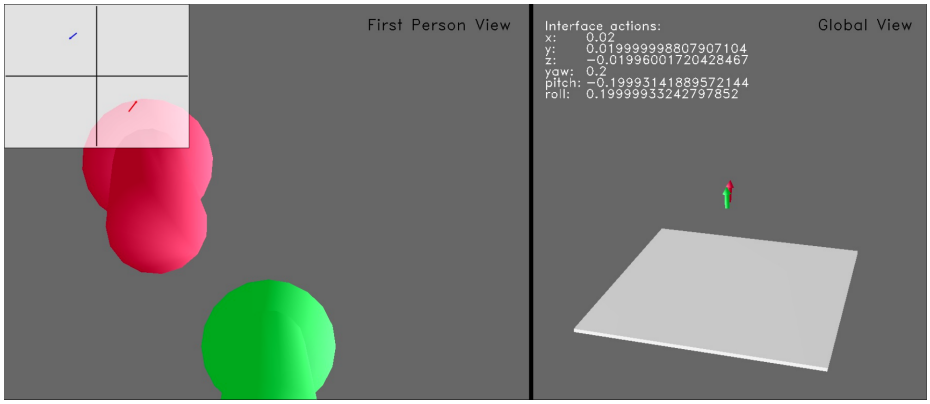

Fig. 4. The 3D navigation user interface. Left: the user/camera view; Right: static bird's eye view. Current user gestures are displayed on top left. The goal is to superimpose the green arrow attached to the camera with the red non moving target arrow.

μ is the user's gesture velocity. A small μ will mean small relative increments toward the target. This definition of μ goes in the sense that a human user will tend to do faster gestures while far from the target and slower, more precise gestures while close to it. Now that we have two points of the wanted intermediate object position, we can solve the equation 7 in order to get the transformation of every point of the object to the intermediate position. At last, we can choose two random points p_1 and p_2 on the object, transform them using the computed $\hat{\theta}$ parameters and build the two trajectories $[p_1,\ p_1^{inter}]$ and $[p_2,\ p_2^{inter}]$. The state bs_i of the agent will be the concatenation of these two trajectories, resulting in a vector of size 8.

Results — We compare our trained agent to an optimal solution. This optimal solution is easily modelled as we defined both the analytic solution of a user and of the interface. On an average of 100 episodes, the optimal solution finishes an episode in 40 steps and obtains a reward of +0.5 per episode.

After a training of about 300k steps, the interface agent obtains very similar results: on an average of 100 episodes, it finishes an episode in 41 steps and obtains a reward of +0.4. It is visually impossible to separate the optimal solution from the learned agent. An illustration of a rollout in the environment is given in Fig. 3.

6 Experiments 2: 3D navigation from 2D gestures

We now discuss the experiments on the real 3D navigation user interface, for which no optimal solution is known to exist. As described in the introduction, we want to learn an agent to map 2D finger gestures to motion in a 3D environment, an ill-posed problem. To this end, we extended the 2D toy problem to 3D, maintaining the user's goal of moving a (now 3D) content to superimpose an object over a non-moving object, as shown in Fig. 4. As there is no simple handcrafted way to simulate a human user, we use the multi-agent RL setup described in Sec. 4. The 3D affine transformation to

Fig. 5. Navigating the latent space learned by the VAE. Middle column: zero code z. Lines correspond to different modified latent variables (dim=8). Columns correspond to different values.

be learned by the interface agent can be expressed using homogeneous coordinates in 4 dimensions as a 4×4 matrix ϕ:

$$\phi = \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \\ 0 & 0 & 0 & 1 \end{bmatrix}, R = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix}, t = \begin{bmatrix} t_1 \\ t_2 \\ t_3 \end{bmatrix} \tag{10}$$

As we do not consider scaling (redundant with forward motion), the matrix is totally parametrized by 6 coefficients: $[\tau_x, \tau_y, \tau_z, \rho_x, \rho_y, \rho_z]$, where $\tau_.$ and $\rho_.$ are, respectively, translation coefficients and Euler angles on the 3 axis. We limit the Euler angles to $]-\pi, \pi]$ to ensure their unicity given an axis. Such a vector can define transformations as well as object positions when using a fixed referential in the environment. With this formalization, The user agent A_u gets as observations the camera position vector and learns a policy over actions which are trajectory vectors $[l_x^1\ l_y^1\ l_x'^1\ l_y'^1\ l_x^2\ l_y^2\ l_x'^2\ l_y'^2]$. The interface agent A_i observes the output of A_u and learns a policy over residual transformation vectors of the camera, i.e. $s_u^{t+1} = s_u^t + a_i$. We define the reward equivalent to the 2D problem in section 5, given in equation (8), the difference being that each object has 2 vertices instead of 3, and that vertices are in 3D space.

VAE training and architectures — We train the latent 2D gesture representation with a VAE on a dataset of multi-touch interaction gestures, in particular the Itekube-

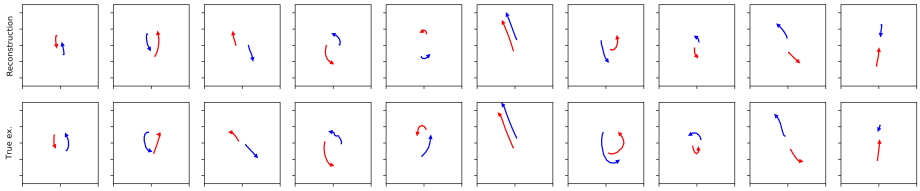

Fig. 6. Reconstruction examples made by the VAE. The first line displays reconstructed gestures while the second displays the corresponding original gestures.

7 Dataset [10]. We used all the two-finger gestures from the translation, pinch and rotation classes. Each gesture of the dataset was sampled using the dynamic sampling described in [10] to fix the length of the gestures to 10 timesteps. The VAE was trained for 50 epochs with β=0.07, a batch size of 128, and a learning rate 0.002. The latent code is of size 8. The encoder and the decoder are both recurrent. Encoder: a one-layer GRU [9] with a hidden state of size 256 and ReLU activation. It is recurrent in time and reads inputs of size 4 (two 2D finger positions). Second, two FC layer predict, respectively, the mean and the stddev of the latent code from the last hidden state. The decoder is a two-layer GRU: The first layer has a hidden state of size 128 with ReLU activation, the second layer has a hidden state of size 4 (to reconstruct the position of both fingers at each timestep), no activation. The entire code is fed to it at every timestep, i.e. the number of unrollings of the GRU will determine the number of timesteps of the reconstruction. At training time, this number is set to the number of timesteps of the original data (10), but at inference time, this number can be arbitrarily, which allows to produce shorter or longer trajectories from the same latent code.

Fig. 5 visualizes the latent representation, and reconstruction examples are given in Fig. 6. The representations from the latent space are satisfying as we can observe some high level features and disentanglement: for instance, dimension 0 expresses pinch, while dimension 2 expresses clockwise rotation.

Multi-agent RL training — We chose the model-free off-policy actor-critic method Deep Deterministic Policy Gradient (DDPG) [17]. In our setup, an epoch cycle consists of two phases: (i) a rollout phase on an episode, where all quadruplets [state, action, reward, new state] are stored in the replay memory of the agents; (ii) a training phase where quadruplets are randomly sampled from the memory, batched (in sizes of 4096) and used to train the actor and the critic of the agents. We define an epoch as 100 epoch cycles. The training is arbitrarily stopped when no improvement on the metrics is observed. A training session takes about 2 days on a Titan-X Pascal GPU.

Join training of both, A_u and A_i, was not successfull. We suspect the added variance and the moving value of state-action pairs for both agents as a source of the problem. Similar to [4], we chose to train them in an alternating manner: during an epoch, only one agent will be trained, while the weights of the other agents are kept fixed.

Stacking timesteps — We consider two ways for the two agents to communicate. The simplest one is a two-instant communication: the decoder produces a gesture of only two time instants, and the interface produces the corresponding action. It is simple, but leads to non-smooth user gestures difficult to appreciate from a human viewpoint

	Mean reward/ep.	Mean #steps/ep.	Nb. training steps
No Stacking	3.6±1.0	53±1	17.6M±0.4M
+ Stacking	0.5±1.5	56±4	24.6M±6.3M
Theoretical Opt.	5.0	40	N/A

Table 1. Results on the 3D environment. The last line gives theoretical optimal results based on the interface action amplitude, without considering a naturalness constraint. The Mean reward/ep. is the mean cumulated reward per episode obtained in the best performing epoch. The Mean nb. steps/ep. is the mean number of timesteps needed to successfully finish an episode in the best performing epoch. The Nb. training steps is the number of environment steps that was needed to attain the best performing epoch. Stacking improves usability but *NOT* efficiency.

is hard. We also consider stacking time instants: A_u produces a complete gesture of 10 timesteps, and A_i must produce the corresponding sequence of actions (9 if there are 10 timesteps). In this case, a step from the RL perspective will contain 10 update steps of the environment. This decouples the update speed of the agents from the sampling speed of the finger gestures, referred to as "*Stacking*" in Table 1.

Architectures — In what follows, FCX refers to an FC layer with X hidden units, with layer normalization and ReLU activation. The actor of A_u is an MLP with two hidden FC100 layers. The output layer is FC and activated with tanh, predicting a vector of size 8 (the latent code z expanded by the VAE decoder D). Another FC100 layer is plugged to the first hidden layer, with an output layer producing the estimate $\hat{a}_i$. The critic of A_u is an MLP with two hidden FC100 layers. The policy action is concatenated to the first hidden layer. A linear FC layer predicts the value Q.

The actor of A_i is an MLP with two hidden FC64, and an output layer with tanh activation. The output size is either 6 for the standard solution or 6x9=54 for the stacked solution. The critic of A_i has the same architecture as the critic of A_u, except hidden layers are FC64.

Results — Quantitative results are given in Table 1. Each setup was reproduced with 3 different random seeds. We consider that a run has converged whenever all 100 episodes of an epoch are successful. Once it has converged, it can still improve by solving episodes faster. This is measured as the mean number of steps needed to solve an episode. The mean reward per episode is also correlated to the quality of the interaction protocol, but should be interpreted differently. Indeed, a run with a lower mean step per episode but a higher mean reward per episode is most likely less satisfactory than a run with higher steps but lower reward. This is because the first type of solutions tend to be less continuous with harsher action changes, while the second is technically slower but goes in the direction of the objective more smoothly.

"Stacking" and usability — the interaction protocols must also be observed visually in order to assess their global quality: good interfaces should display distinctive characteristics, such as a similar curvature between 2d trajectories and 3D movements of the camera, or well defined classes for similar actions. While stacking does *NOT* improve efficiency (as shown in table 1), it makes the protocol usable. The continuous

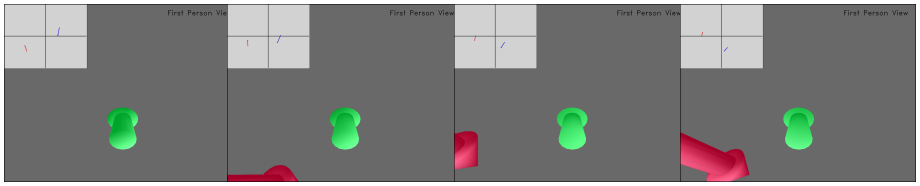

Fig. 7. An example of 4 back to back frames from the MARL setup learning the 3D navigation problem with instant stacking. We want to emphasize on the continuous aspect of user gestures (top-left) and the semantic. We can see that the user agent is currently performing a rotation-like gesture.

aspect of gestures using instant stacking is illustrated in 7, videos are provided in the supplementary material.

7 Conclusion

We presented a novel method for automatically learning interaction protocols from natural interactions. While our application for this paper is limited to touch interfaces, this setup can virtually be applied to any technology and any software, as long as a large enough interaction dataset can be collected. We want this work to be a step toward a better co-adaptive relation between the human and a computer, allowing for individually suited interfaces and higher levels of interaction. Future work will model the interaction protocol as a POMDP, which should allow to better represent long term dependencies and tackle more complex regularities. The main remaining problem is to fine-tune the learned models on real human users, which requires large-scale efforts with a large amount of human partners.

References

1. Alvarez-Cortes, V., Zayas, B., Silva, V., Ramirez Uresti, J.: Current trends in adaptive user interfaces: Challenges and applications. pp. 312–317 (2007)
2. Beattie, C., Leibo, J.Z., Teplyashin, D., Ward, T., Wainwright, M., Küttler, H., Lefrancq, A., Green, S., Valdés, V., Sadik, A., Schrittwieser, J., Anderson, K., York, S., Cant, M., Cain, A., Bolton, A., Gaffney, S., King, H., Hassabis, D., Legg, S., Petersen, S.: DeepMind Lab. arxiv 1612.03801 (2016)
3. Beeching, E., Wolf, C., Dibangoye, J., Simonin, O.: Deep Reinforcement Learning on a Budget: 3D Control and Reasoning Without a Supercomputer. arxiv 1904.01806 (2019)
4. Boutilier, C.: Planning, learning and coordination in multiagent decision processes. In: Theoretical Aspects of Rationality and Knowledge. pp. 195–210 (1996)
5. Bowman, D., Chen, J., Wingrave, C., Lucas, J., Ray, A., Polys, N., Li, Q., Haciahmetoglu, Y., Kim, J.S., Kim, S., Boehringer, R., Ni, T.: New directions in 3d user interfaces. IJVR 5, 3–14 (2006)
6. Bowman, D.A., Kruijff, E., LaViola, J.J., Poupyrev, I.: An introduction to 3-d user interface design. Presence: Teleoper. Virtual Environ. pp. 96–108 (2001)

7. Cashion, J., Wingrave, C., Joseph J., Jr, L.: Dense and dynamic 3d selection for game-based virtual environments. IEEE transactions on visualization and computer graphics (2012)
8. Chen, Z., Anquetil, É., Mouchère, H., Viard-Gaudin, C.: A graph modeling strategy for multi-touch gesture recognition. I.C. on Frontiers in Handwriting Recognition (2014)
9. Cho, K., van Merriënboer, B., Gülçehre, Ç., Bahdanau, D., Bougares, F., Schwenk, H., Bengio, Y.: Learning phrase representations using rnn encoder–decoder for statistical machine translation. In: Conference on Empirical Methods in Natural Language Processing (2014)
10. Debard, Q., Wolf, C., Canu, S., Arné, J.: Learning to recognize touch gestures: Recurrent vs. convolutional features and dynamic sampling. I.C. on Automatic Face and Gesture Recognition (2018)
11. Ferretti, S., Mirri, S., Prandi, C., Salomoni, P.: Exploiting reinforcement learning to profile users and personalize web pages. In: International Computer Software and Applications Conference Workshops (2014)
12. Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., Bengio, Y.: Generative adversarial nets. In: NIPS, pp. 2672–2680 (2014)
13. Higgins, I., Pal, A., Rusu, A.A., Matthey, L., Burgess, C., Pritzel, A., Botvinick, M., Blundell, C., Lerchner, A.: Darla: Improving zero-shot transfer in rl. In: ICML (2017)
14. Hinckley, K., Wigdor, D.: The human-computer interaction handbook: Fundamentals, evolving technologies and emerging applications (2012)
15. Kingma, D.P., Welling, M.: Auto-encoding variational bayes. CoRR abs/1312.6114 (2014)
16. Langley, P.: Machine learning for adaptive user interfaces. In: Brewka, G., Habel, C., Nebel, B. (eds.) KI-97: Advances in Artificial Intelligence. pp. 53–62 (1997)
17. Lillicrap, T.P., Hunt, J.J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., Wierstra, D.: Continuous control with deep reinforcement learning. CoRR abs/1509.02971 (2015)
18. Lü, H., Li, Y.: Gesture coder: A tool for programming multi-touch gestures by demonstration. In: SIGCHI Conference on Human Factors in Computing Systems (2012)
19. Matthey, L., Pal, A., Burgess, C., Glorot, X., Botvinick, M., Mohamed, S., Lerchner, A.: β-vae: Learning basic visual concepts with a constrained variational framework. In: ICLR (2017)
20. Nair, A., Pong, V., Dalal, M., Bahl, S., Lin, S., Levine, S.: Visual reinforcement learning with imagined goals. In: NeurIPS. pp. 9209–9220 (2018)
21. Ortega, F.R., Abyarjoo, F., Barreto, A., Rishe, N., Adjouadi, M.: Interaction Design for 3D User Interfaces: The World of Modern Input Devices for Research, Applications, and Game Development. A. K. Peters, Ltd. (2016)
22. Ross, S., Bagnell, D.: Agnostic system identification for model-based reinforcement learning. In: ICML (2012)
23. Seo, Y.W., Zhang, B.T.: A reinforcement learning agent for personalized information filtering. In: International Conference on Intelligent User Interfaces (2000)
24. Weir, D., Rogers, S., Murray-Smith, R., Löchtefeld, M.: A user-specific machine learning approach for improving touch accuracy on mobile devices. In: ACM Symposium on User Interface Software and Technology. pp. 465–476 (2012)
25. Weld, D.S., Anderson, C., Domingos, P., Etzioni, O., Gajos, K., Lau, T., Wolfman, S.: Automatically personalizing user interfaces. pp. 1613–1619. IJCAI (2003)
26. X.B. Peng, M.A., Zaremba, W., Abbeel, P.: Sim-to-real transfer of robotic control with dynamics randomization. In: ICRA (2018)