Learning Enriched Latent Spaces for Flexible Model-based Control

Pier Luigi Dovesi **Eysteinn Gunnlaugsson** **Carlo Rapisarda** **Dominik Fay**
KTH Royal Institute of Technology
{dovesi, eysgun, carlora, dominikf} @kth.se

Abstract

When training a robotic agent using a simulator, one often finds that the learned policy performs poorly when transferred to the real world. Existing approaches do not consider explicit knowledge that we might have about the real environment that was not available at training time. This work aims to find an approach to link observable environment features to relevant non-observable features through compression and disentanglement of the observed input. We use recent advances in Deep Learning and Reinforcement Learning to implement a control pipeline for OpenAI continuous-space environments starting from images. In particular, we employ a dynamic-β-VAE for the image compression and disentanglement, a MLP for the model dynamics and finally a MPC for the control. We show that the proposed pipeline performs well on the inverted pendulum swing up task, and is able to take advantage of the learned latent space enriched with information provided only at test time. Furthermore, we investigate the performance of the pipeline on the more complex robotics environments.

1 Introduction

In robotics, obtaining training data on real hardware is costly, both in terms of time and money. At the same time, robotics tasks often require a large number of interactions with the environment until a reasonably high performance can be achieved. This is especially true for vision tasks, e.g. when the robot is equipped with cameras. For this reason, robot agents are often trained in a simulator instead. When transferring the trained agent onto real hardware, one often finds that it performs poorly due to significant differences between simulation and reality. To bridge this so-called *reality gap*, many approaches have been explored in the past (see section 2.1). In addition, extensive manual tuning is often required.

A common theme among the aforementioned approaches is that they do not explicitly learn which properties of the environment differ between simulation and reality. We do, however, often have knowledge about certain aspects of objects that are not directly observable, e.g. physical properties such as mass and friction, at the time we interact with them in reality. The core idea of this project is to find a way to incorporate this knowledge into the agent's decisions at prediction time, i.e. after training has completed.

The path we explore in this project is in its core motivated by two recent advances in Deep Learning and Reinforcement Learning:

1. High interpretability of the latent spaces learned by disentangled VAEs [3].
2. High data efficiency of latent state space models at solving complex tasks.

These allow us to use a VAE to embed the sensor observations into a small interpretable latent space in which we learn the transition dynamics. After training, through visual inspection we identify the latent variable that corresponds to the physical property we want to influence later on. Since

these properties are not directly observable, we encode them into the image with visual aids such as a changed color or size. At prediction time, we then "hijack" the latent variable that encodes this visual aid and assign it the value that represents the true value of the physical property. We validate experimentally that this visual encoding is indeed learned in the disentangled latent space.

Our contributions are the following:

- An approach to explicitly encode knowledge about the environment into a disentangled state space model that was not available at training time
- We motivate our approach by showing in a simple environment that rewards increase when additional knowledge about physical properties is available to the agent
- We conduct qualitative experiments for a visuo-motor task that confirm the benefit of disentanglement to learn interpretable state representations
- We show in a simplified environment that using the disentangled representations to encode physical properties leads to increased rewards, compared to robust policies obtained through domain randomization

2 Related Work

2.1 From simulation to the real world

Many methods aiming to make simulator-trained agents work in the real world have been explored in the past. A common approach is to first train a policy in the simulator and then fine-tune it with a small amount of labeled samples from the real environment. Several problems can arise from this strategy. Firstly, the cost of obtaining the labeled samples may be too high in order to collect an amount that is sufficient for adapting to the new domain. Secondly, fine-tuning may result in "catastrophic forgetting", that is, previous learning progress is undone due to the large gradients that arise from changing the problem domain. In order to overcome these limitations, new transfer learning methods such as adversarial domain adaptation [22, 21] and progressive networks [14] have recently been proposed for robotics tasks.

Another approach to overcome the gap between simulation and reality is domain randomization, which aims to directly learn a robust policy in the simulator. This can be achieved by introducing noise to the learning process, e.g. by adding randomness to the transition dynamics or by perturbing the (simulated) sensor data with Gaussian noise. Although it has been shown that this can be a viable strategy [16, 11], robustness does not explicitly take into account in which way the real environment will be different from the simulation. Whenever we know that a certain physical property will be different in the real environment, it may therefore be preferable to encode this difference explicitly into the agent in a way that it can be changed after training. Neither transfer learning nor domain randomization provide a way to do this. This is the approach we will explore in this work.

2.2 Learning for visuo-motor tasks

Vision tasks in robotics are commonly addressed by first encoding the images in a smaller latent space (e.g. using VAEs) and then solving the problem in the latent domain [18, 19]. By modelling the transition dynamics in the latent space, the sample complexity of the learning task can be heavily decreased, thus making the training process more efficient [20, 6]. [7] use generative networks to model the environment, which enables them to train a policy based solely on latent representations that are generated by the networks, without the need for the agent to interact with the environment directly. They evaluated their approach on video games. We draw inspiration from their idea to use a VAE in conjunction with a LSTM and apply it to a robotics environment. However, we follow the suggestion of [13] to use a model-predictive controller (MPC) instead of a feed-forward neural network to select the best action. Other recent literature suggests that latent dynamics models should contain both stochastic and deterministic elements [2, 5, 8].

3 Background

3.1 Reinforcement Learning

Robotics tasks are usually framed within the context of Reinforcement Learning (RL). In RL, we aim to design an agent that behaves optimally when interacting with an environment. The behavior of the agent is described by a *policy* $\pi_t(s)$: a mapping from the state space (e.g. the joints of a robot arm or positions of objects on a table) to the action space (e.g. velocities of the end effector). For a given state s and action a, the environment transitions into a new state s'. For every action the agent takes, it receives a *reward* $r(s, a)$. Our goal is to find the policy that will, on average, receive the highest rewards, summed over all time steps.

Learning algorithms for RL can be broadly categorized into:

- **Model-based:** First learn to predict the transition dynamics of the environment (called a state space model) and then search for an action that will lead to the highest cumulative reward.
- **Model-free:** Skip the modelling and directly learn the mapping from current state to optimal action.

Although model-free algorithms such as the one used for AlphaGo (Zero) [15] have caught widespread attention, there is a strong incentive for model-based RL: it is usually much more data-efficient, i.e. requires fewer training samples to learn a good policy. For the reasons mentioned in the previous sections, this is a very attractive property to have for robotics tasks.

3.2 Model Predictive Controller

In order to exploit the learned model and close the loop, it is common to employ a Model Predictive Controller (MPC) to select the best action to take, given the current state. In particular, at each time step, the controller generates N possible trajectories over a finite horizon of T time steps, by querying the model ($\mathcal{M}$) recursively with a given sequence of actions.

$$\hat{\boldsymbol{U}} = [u_{t+1},\ u_{t+2},\ \ldots,\ u_{t+T}] \xrightarrow{\mathcal{M}} \left[s'_{t+1},\ s'_{t+2},\ \ldots,\ s'_{t+T}\right] \tag{1}$$

$$\hat{\boldsymbol{U}} = \underset{\boldsymbol{U}}{\operatorname{argmax}} \sum_{k=t+1}^{t+T} r_{\mathcal{M}}(s'_k, u_k) \tag{2}$$

The MPC then selects the best action sequence $\hat{\boldsymbol{U}}$ by computing the cumulative reward of each trajectory, and returning its first action as the control.

The action sequences can be generated in a number of different ways, for instance, through a cross-entropy method [4]. Following Nagabandi *et al.*, we instead generate the action sequences with a uniform random shooting method, which represents a compromise between quality of the resulting policy and efficiency of the system. It can be shown that with an exact model and a large enough number of sequences, the best action chosen with this method actually converges to the optimal action for the given state, i.e. $\hat{\boldsymbol{U}} \to \boldsymbol{U}^*$ as $N \to +\infty$.

3.3 Disentangled VAE model

3.3.1 Variational Autoencoders (VAE)

Variatonal Autoencoders have been introduced by Kingma *et al.* in 2013. Given a dataset **x** of samples from a distribution parameterized by generative ground truth factors **z**, a Variational Autoeconder will learn the data marginal likelihood:

$$\max_{\phi,\theta} \mathbb{E}_{q_\phi(\mathbf{z}|x)}[logp_\theta(x|\mathbf{z})] \tag{3}$$

where ϕ is the parametrization of the encoder and θ is the parametrization of the decoder. This can be decomposed in:

$$logp_\theta(x|\mathbf{z}) \geq \mathcal{L}(\theta, \phi; \mathbf{x}, \mathbf{z}) = \mathbb{E}_{q_\phi(\mathbf{z}|x)}[logp_\theta(x|\mathbf{z})] - D_{KL}(q(\mathbf{z}|x)||p(\mathbf{z})) \tag{4}$$

where D_{KL} is the KL divergence between the target posterior and the predicted one. Therefore, maximizing $\mathcal{L}(\theta, \phi; \mathbf{x}, \mathbf{z})$, the reconstruction loss, corresponds to the maximization of the lower bound of the true objective.

Other important assumptions concern the prior and posterior distribution that will be parameterized as Gaussians with diagonal covariance matrix. This allows the implementation of the "reparametrization trick" through the adding of normalized noise, which makes training the network with backpropagation possible.

3.3.2 β-VAE

Introduced by Higgins *et al.* in 2016, β-VAE is an important improvement of the original the Variational Autoencoder. The difference consists in the introduction of an adjustable hyperparameter β (generally greater than 1) that is multiplied with the KL divergence term in the objective function:

$$\mathcal{L}(\theta, \phi; \mathbf{x}, \mathbf{z}) = \mathbb{E}_{q_\phi(\mathbf{z}|x)}[logp_\theta(x|\mathbf{z})] - \beta D_{KL}(q(\mathbf{z}|x)||p(\mathbf{z})) \tag{5}$$

Adjusting β to appropriate values can lead to a well disentangled latent representation $\mathbf{z}$. Higher values of β will provide stronger disentanglement, but will cause a worse reconstruction quality. This happens since the disentanglement is caused by a higher pressure for the posterior $q_\phi(\mathbf{z}|x)$ to correspond to the Gaussian prior $p(\mathbf{z})$. This higher pressure results in information loss through the VAE bottleneck.

3.3.3 Increased Capacity β-VAE methods

An effective way to overcome this trade-off is to dynamically change the disentanglement pressure during training, an approach that has been introduced by Burgess *et al.* in 2018:

$$\mathcal{L}(\theta, \phi; \mathbf{x}, \mathbf{z}) = \mathbb{E}_{q_\phi(\mathbf{z}|x)}[logp_\theta(x|\mathbf{z})] - \gamma|D_{KL}(q(\mathbf{z}|x)||p(\mathbf{z})) - C| \tag{6}$$

where C is an hyperparameter progressively increased from zero (high disentanglement pressure) to a value large enough to guarantee better quality reconstruction (low disentanglement pressure).

Starting from this approach, we implemented a similar and even easier solution consisting in annealing β during training, from an initial high value (adjustable) down to 1.

3.4 Long short-term memory

Given a latent embedding from the VAE, we experimented with a long short-term memory (LSTM) [10] network to predict the transition to the embedding of the next state. A LSTM is a specific kind of recurrent network that was designed to solve the problem of vanishing/exploding gradients that often occur in regular RNNs. It is composed of units that are not limited by how long they can retain information from previous iterations. As in other networks, the units are arranged into layers, followed by a fully-connected output layer.

4 Methodology

We evaluate our approach on two different types of environments from OpenAI Gym [1]: the Pendulum environment, and the Push robotics environment. The two have very different characteristics, such as complexity of visual features, complexity of the simulated physics, and complexity of the task to solve, therefore they are well suited to test whether or not our architecture generalizes. In the Pendulum environment, the task is to swing a pendulum to a vertical position and keep it in this position for as long as possible, controlling only the torque applied from the pivot point. For the Push environment, as the name suggests, the task is to operate a robotic arm and push a cube on a

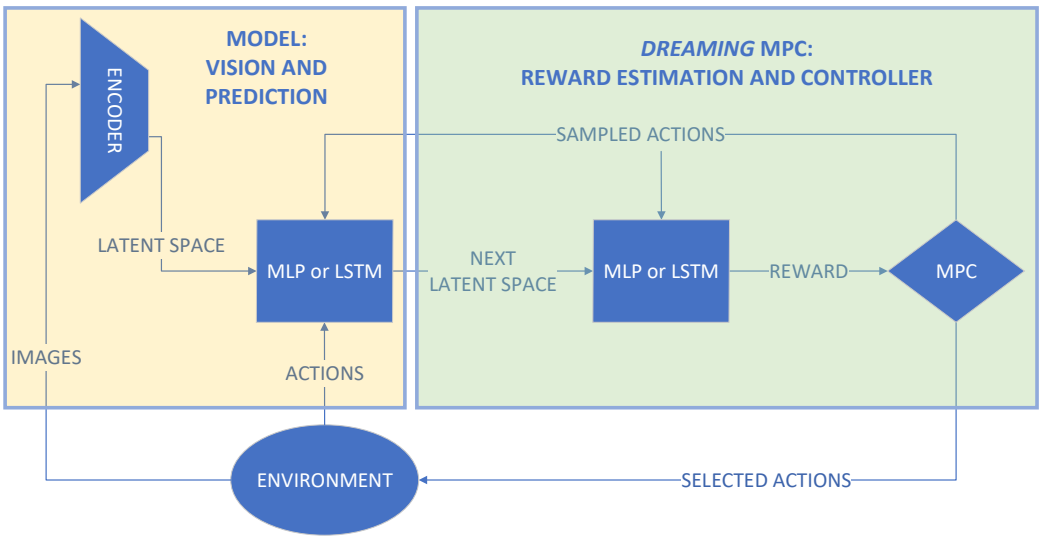

Figure 1: Overall architecture configuration. Left box: VAE encoder and prediction network, Right box: Reward estimator network and Control

table towards a goal position, controlling the velocities of the end effector in Cartesian space. While the Pendulum environment has very simple deterministic physics, the Push environment is much less predictable, as it is simulated with MuJoCo [17], an advanced physics engine. Additionally, we introduce another layer of difficulty by replacing the cube with a sphere with irregularities, meaning that when pushed, this will roll with a pattern that is difficult to predict.

Since the goal of this work is to parameterize the agent's policy with varying values of physical properties of the environment, we are interested in ways to influence the latent space to encode such parameters. We do this by modifying visual features of the environment in relationship with the physical property that changes, in ways that are plausible in the real world. In particular, in the case of the Pendulum environment, we set a piece-wise linear relationship between the mass and the length, so that a visually longer pendulum is simulated as heavier, and vice versa. Instead, in the Push environment, we show the sphere as blue when this is simulated as heavy, and as yellow when simulated as light.

4.1 Overall Architecture

The overall architecture, illustrated in Figure 1, can be divided into two parts with different roles: a model and a "dreaming" MPC. The model is composed by the encoder part of a VAE and by a predictor network that aims to estimate the next state of the system. In particular, the encoder is fed with RGB images representing the current state of the environment, and compresses them to a small, disentangled latent space. The predictor network, either a MLP or LSTM, has the role of predicting the next state (in latent space) given an action and a short history of previously seen latent spaces. The output of the model box represents an prediction of the next environment state in latent space. Following, we have the second part of our architecture, the "dreaming" MPC, which can be further divided into two blocks: a reward predictor and a Model Predictive Controller (MPC). The former predicts the reward at the next timestep given a history of past states, while the role of the MPC is to select the best action by exploring possible future states, picking the control that would lead to the highest reward.

5 Implementation

5.1 VAE implementation

In order to guarantee a fair comparison with the VAE in World Models by Ha and Schmidhuber, we maintained a similar network architecture in terms of layers and filter sizes. On the other hand, regarding the loss function used, we experimented with several alternatives and improvements.

Reconstruction Loss: We implemented three different options for the reconstruction loss:

- $\boldsymbol{L_2}$ **norm**: Good for overall picture understanding, but poor resolution. The image might remain blurry and details are easily lost (original loss implemented in World Models).
- **Binary Cross-Entropy**: Focuses on the information preservation through the network; provides reliable results both in terms of accuracy and precision.
- $\boldsymbol{L_1}$ **norm**: Returns very sharp results, but provides low accuracy.

Kullback–Leibler Divergence loss: Regarding the KL loss, three options were considered:

- **Clipped KL**: Original KL clipping implemented in World Models. The KL is kept low in an attempt to preserve the reconstruction quality.
- **Increased Capacity** β**-VAE**: Loss proposed by Burgess *et al.* with linearly increasing C, offering a trade-off between disentanglement and reconstruction quality.
- **Our experiment**: Based on the Increased Capacity loss, rather than increasing C, we use an annealing β. This provides similar benefits as the Increased Capacity loss.

5.2 MPC implementation

When using model-based Reinforcement Learning methods, we typically assume that the reward function is known and can be computed given any state action couple, however, this presents at least two problems. First, for some particular tasks we might not have direct access to the real reward function of the system, for instance when such function is stochastic, or too complex to be computed analytically. Second, and perhaps more importantly, the reward function could be very sensitive to noise, meaning that if the predicted future states are not accurate enough, we might obtain rewards with very large errors, hence preventing the Model Predictive Controller from selecting good actions.

To solve both of these issues, we designed our architecture to predict the reward directly as opposed to predicting future states, since the former is the only value required for the MPC to work. In this case, the procedure remains almost unchanged, replacing the reward in eq. 2 with an estimate r' that depends on the action and on the predicted state in the latent space.

5.3 Predicting future states

Since the controller relies on the model to predict future states, it is crucial that this component is accurate enough, because when the MPC explores possible future trajectories, the errors of the model predictions accumulate for each timestep towards the horizon. Following [7], we initially implemented both the latent-to-latent and the latent-to-reward networks as LSTMs, as these are particularly suited for predicting future states given a history. As an alternative to LSTMs, we explore the possibility of employing two simple Multi-Layer Perceptron (MLP) networks, concatenating a number of past states together with the current state (all encoded in latent space), in a way emulating the window of a Long short-term memory network. The latent-to-latent MLP also takes the selected action as input, as this is required to predict future states of the environment.

5.3.1 Training the overall architecture

In order to train our architecture, we generate a large dataset by doing random exploration on the environments. Both for the Push and the Pendulum environments, the dataset is composed by 1000 episodes of length 10 each, i.e. a total of 10000 images (64x64), associated with actions and rewards at each timestep. Since the latent-to-latent and latent-to-reward networks must be trained on states encoded into latent space, the VAE needs to be trained first. In particular, this is done

in an unsupervised manner. Once the VAE is trained, it is used to encode the existing dataset into compressed latent space, which as discussed, is then used to train the other two predictor networks.

6 Results

6.1 Motivating result

A core assumption of our approach is that knowledge about physical properties of the environment that are not directly observable will improve the performance of the agent. In a first experiment, we tested this assumption in a very simplified setting. We trained several model-based policies in the Pendulum environment, while varying the pendulum's mass between each episode. The settings differ by (1) which masses the agent is exposed to during training, and (2) whether it can observe the mass (i.e. receive it as an additional input). Figure 2 shows the rewards each of these policies collect during an episode when run in environments of varying masses.

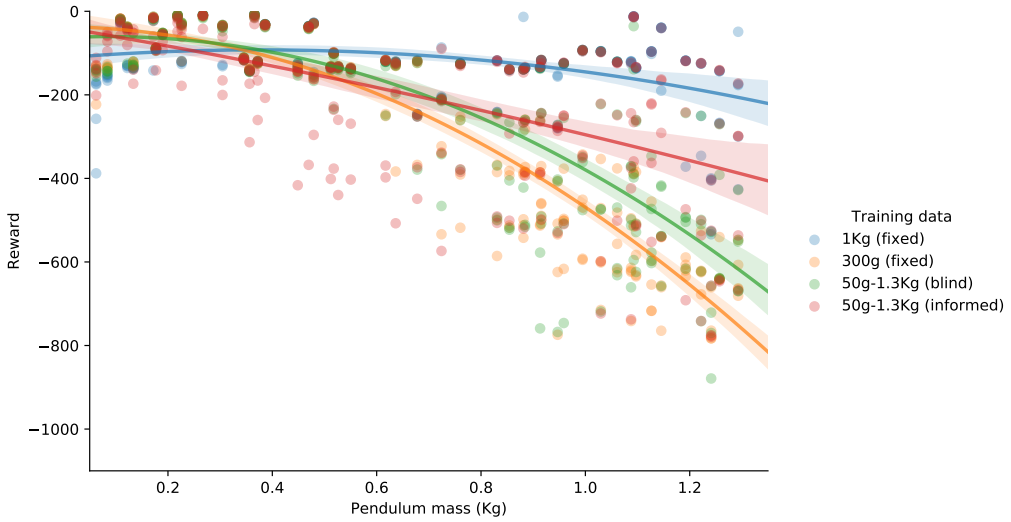

Figure 2: Cumulative rewards of policies trained in various settings. Each point represents the cumulative reward (y-axis) the policy collects during an episode when the pendulum is initialized with a certain mass (x-axis). Blue and yellow are trained with constant masses of 1000g and 300g, respectively. Green is a robust policy trained on varying mass values drawn uniformly between 50g and 1300g. Red is also trained with varying mass values but, in contrast to the other three settings, it additionally receives the exact value as an input. The regression curves are obtained from quadratic regression.

Firstly, we notice that all policies consistently perform worse the heavier the pendulum is. This could indicate that heavy pendulums are inherently more difficult to control in this environment. A reason for this might be that heavier pendulums require higher torques but the environment places an upper limit on the torque the agent is allowed to apply. This means that it will take a longer time to swing the pendulum up, hence it takes longer to reach the high-reward region. At the same time, policies trained on heavy pendulums seem to still work decently on low masses.

Secondly, by comparing the red and the green line, we see that being informed about the mass provides a meaningful benefit over being robust. Although the performance of the informed policy is still lower than the policy trained with a large constant mass, we attribute this to the peculiarities of the Pendulum environment. Due to the reasons stated in the previous paragraph, the learning benefits from having more data about interaction with heavy pendulums because this is the most difficult region of the state space.

6.2 VAE results

6.2.1 Push environment

In order to choose the best combination of loss functions, we performed a grid search keeping a constant latent space size of 32 and the value of β fixed to 100. The test was performed over 20 reconstructed test images and the reported quality is associated to the number of correct reconstructions. In particular, we count as "correct" only the output images where all necessary information (sphere position, colors, gripper position) is correctly reconstructed.

	Clipped KL	**Incr. Capacity**	**Annealing β**
MSE	20%	0	10%
Binary CE	65%	70%	80%
$\boldsymbol{L_1}$	0	0	0

Table 1: VAE reconstruction quality on the Push env., with latent space of size 32, and $\beta = 100$. Note that the columns correspond to different types of the KL loss, while the rows correspond to different reconstruction losses.

As shown in Table 1, the VAE model with Annealing β and Binary Cross-Entropy reconstruction loss outperforms all other loss combinations. At this point, using precisely this combination of losses, we proceed with the study of the disentanglement quality and the number of latent variables used.

In order to assess the quality of the disentanglement, due to lack of appropriate metrics, we are forced to perform a qualitative evaluation. In particular, we report our results on a scale of three levels: no disentanglement visible, partial disentanglement, complete disentanglement (None, Fine, Great). Figure 3 shows an example of disentanglement in the Push environment.

The activity of a latent variable instead is evaluated according the variance of the decoded images over the linear change of that variable, keeping all the other latent variables fixed. The process is then repeated for several latent spaces. If the variance of the decoded images is lower than a fixed threshold, then the variable is considered inactive.

β	**Reconstruction Quality**	**Disentanglement Quality (None, Fine, Great)**	**Active Units (out of 32)**
10	80%	None	15
50	80%	None	14
100	80%	Fine	14
150	80%	Fine	14
200	80%	Great	14
250	70%	None	13
300	75%	Fine	14
400	70%	None	13
500	70%	None	13

Table 2: VAE results on the Push env. with different values of β. Latent space size: 32; reconstruction loss: Binary CE; KL loss: Annealing β from initial value down to 1 in 100 epochs.

From Table 2, we can identify the two classical trends observed in β-VAE. In particular, we notice how the increment of the initial β is both compressing the used latent space (from 15 active variables with $\beta = 10$ to 13 with $\beta = 400$) and reducing the reconstruction quality (from 80% to 70%). The disentanglement instead is limited to only some models. Indeed, we noticed that clear disentangled representations are visible only when the initial β is at least 100 and no higher than 200.

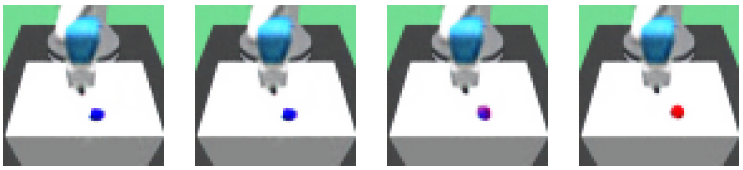

Figure 3: Latent walk over a single latent variable, i.e. changing the value of a specific latent variable while keeping all others fixed. In this example on the Push environment, it is clear that the latent walk changes the color of the sphere, from blue to red, with minimal visual changes to other elements of the image.

6.2.2 Pendulum environment

The same analysis has been performed on the Pendulum environment. In this case, the simplicity of the environment allows an even clearer analysis of the effect of the losses and β.

	Clipped KL	**Incr. Capacity**	**Annealing β**
MSE	0	0	0
Binary CE	100%	95%	100%
L_1	0	0	0

Table 3: VAE reconstruction quality on the Pendulum env., with latent space of size 8, and $\beta = 100$. Note that the columns correspond to different types of the KL loss, while the rows correspond to different reconstruction losses.

Similarly to what we saw for the Push environment, Table 3 shows that, in this case too, the Binary Cross-Entropy loss offers the best reconstruction quality by far. As for the KL losses, we can observe that the differences in terms of output quality are negligible, although we choose to proceed with the Annealing β variant to experiment more with it.

β	**Reconstruction Quality**	**Disentanglement Quality (None, Fine, Great)**	**Active Units (out of 8)**
1	100%	None	3
10	100%	Great	3
50	100%*	None	4
100	100%*	Great	3
150	100%*	None	3
200	100%*	None	4
250	0%	None	4
300	0%	None	4

Table 4: VAE results on the Pendulum env. with different values of β. Latent space size: 8; reconstruction loss: Binary CE; KL loss: Annealing β from initial value down to 1 in 100 epochs. Results marked with * correspond to configurations yielding blurry reconstructed images.

When it comes to choosing the best value for β, in this case the answer is not as straightforward, and the trends that we had observed before on the Push environment are now much harder to spot. This is likely due to the simplicity of the images of the pendulum, and the smaller size of the latent space. Table 4 shows that for different values of β, the disentanglement is either good or not present

at all. Similarly, the reconstruction quality is either very good or very low, with the most extreme cases being a few models outputting blank images. In all cases, the number of active units varies from 3 to 4 out of 8, with 3 being effectively the smallest possible representation size for the state of this environment. Interestingly, when considering only models with 3 active units, we noticed the emergence of two possible representations of the images, which can be seen in Figure 4 and Figure 5.

Figure 4: Disentangled pendulum representation with polar coordinates. Note that the rows correspond to the latent variables responsible for the torque circle, the angle of the pendulum, and the length, from top to bottom.

In order to illustrate this, we perform latent walks for the active units in the two different cases. In practice, this corresponds to decoding latent space into images while changing one latent variable and keeping the others fixed, enabling us to observe the effect of each component visually. In one case (shown in Figure 4), the representation found by the VAE corresponds to the position of the end of the pendulum in polar coordinates: the first variable represents the size of black circle (the torque applied to the pendulum), the second variable corresponds to the angle, while the third variable is proportional to the length of the pendulum (both positive and negative).

Figure 5: Disentangled pendulum representation with Cartesian coordinates. Note that the rows correspond to the latent variables responsible for the y position, the torque circle, and the x position, from top to bottom.

In other cases (shown in Figure 5), the representation found by the VAE corresponds to the position of the end of the pendulum in Cartesian coordinates. It is a different way to disentangle the variables describing the input image, not dividing the pendulum position into radius and angle, but rather into x and y coordinates. Interestingly enough, this is precisely the representation of the observations that is provided by the Pendulum environment when we don't use vision. Motivated by this, we investigate this learned representation even further in the following paragraph.

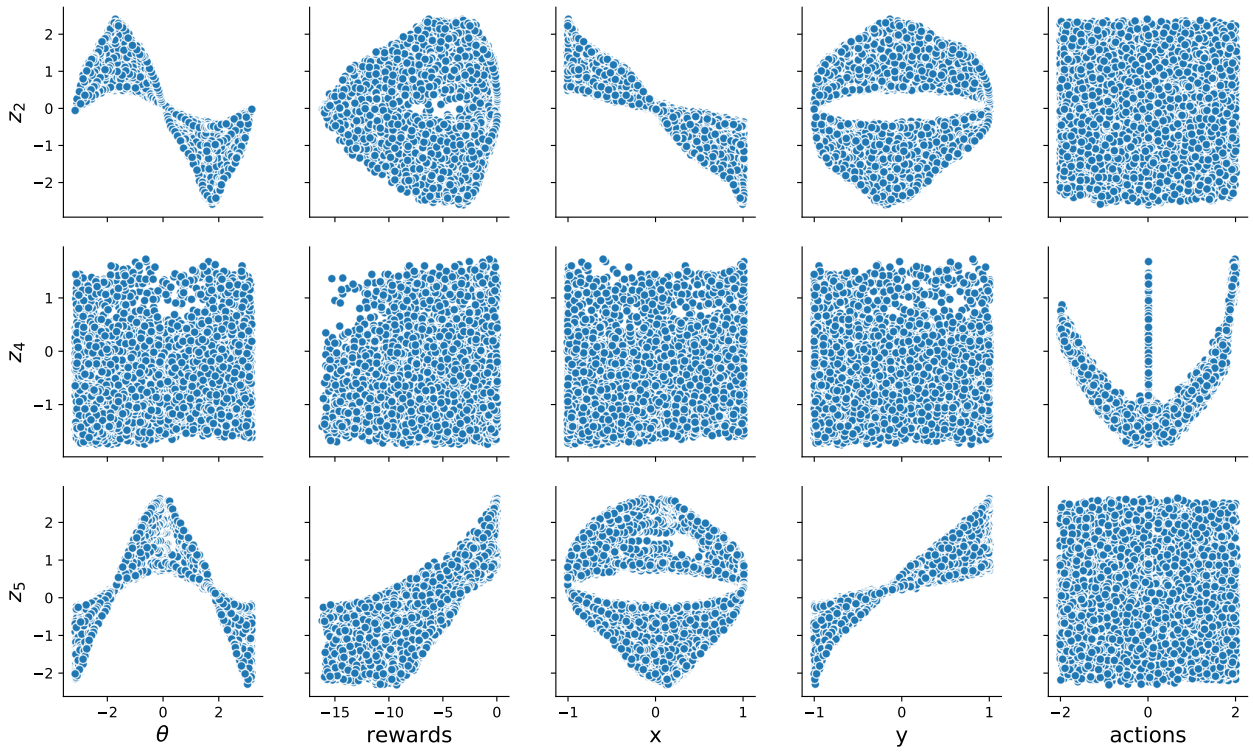

Figure 6: Latent variables (rows) against actual environment features (columns)

To get a better overview of this result, for each image in the Pendulum dataset we plot the x and y coordinates of the head of the pendulum against three different latent variables (Figure 6). It is easy to see the x and y disentanglement by observing the simple linear relationship between the actual x value against z_2, and actual y value against z_5. Note that the correlation is not represented by a single line, but a bundle of lines, since the pendulums can be of different lengths.

The figure also shows that z_2 and z_5 have a sinusoidal relationship with the pendulum angle θ, which is precisely what we would expect. We observe that z_5 has a positive linear correlation with the reward (which is a function of y). Furthermore, when plotting x against z_5 and y against z_2, all the combinations are present except instances where the data is too close or too far from the pendulum pivot point (as the radius is bounded to a minimum and maximum value). Here it can also be noticed that having many possible lengths in the dataset is causing a bundle of circumferences. Finally, it can be observed that z_3 reacts only to the action with a quadratic relationship, which is precisely what we would expect; indeed, the size of the torque circle displayed around the pendulum is directly proportional to the torque applied (i.e. the action) squared, hence we see a parabola.

6.3 Impact of the additional information on the full pipeline

After having evaluated the VAE in particular, we now performed experiments to see if the additional information encoded in the latent space indeed led to higher rewards when putting all pieces together. For this purpose, we try to "cheat" the flexible policy by visualizing the pendulum with lengths that do not correspond to its true mass. We selected five true masses and five fake masses each spread evenly between 0.05 and 1.3. Then, we ran the policy on all combinations of the two and recorded the cumulative reward. Figure 7 shows the corresponding plot.

We can observe that rewards tend to peak whenever visual mass and real mass are identical. This is especially evident for the blue and green lines in the plot. This pattern indicates that the policy is indeed making use of the visual aid we provided to it and it is doing so in a way that increases the obtained rewards.

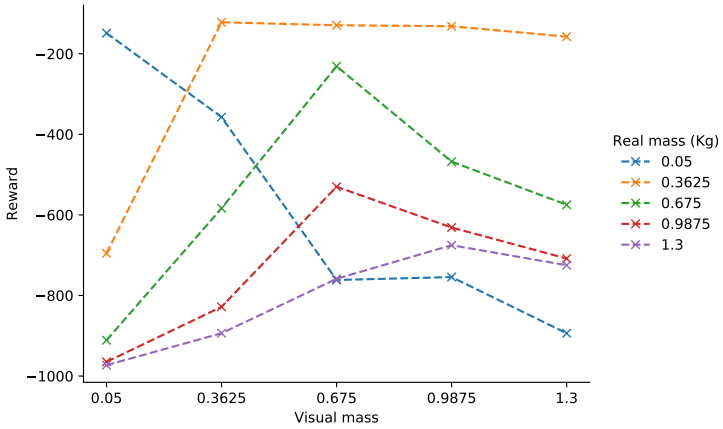

Figure 7: Rewards (y-axis) obtained by the flexible policy, depending on the mass used to visualize the pendulum length (x-axis) and the actual mass of the pendulum (legend)

.

7 Conclusion

We have demonstrated the possibility of learning policies from latent space enrichment and we showed that the transfer of knowledge from visual to physical proprieties is possible and generally helps the agent to perform better. We have identified some crucial factors that highly impacted on our results:

- **Quality of the disentanglement**: We noticed that a clear disentanglement leads to a much better understanding and estimation of the next state and reward. Because of this, we were able to reach significantly better results in the Pendulum environment, as the simplicity of its visual features allowed us to use just a few active latent variables for the reward evaluation. With a high enough disentanglement quality, it is possible to operate effectively on just a very small set of variables; moreover, this also opens the possibility to manipulate those variables in order obtain specific results in the predictions.
- **Accuracy of the predictors**: Since a Model Predictive Controller is highly dependant on the performance of the predictors (both reward and next state), the low accuracy of our models for the Push environment impacted on our ability to test the pipeline on more complex tasks.
- **Dataset generation**: The dataset size, variance, and the behavioral policy employed to generate training data influenced significantly the performance of the full architecture.

8 Future Work

Since our results are quite promising, we would like to take our research further and use the proposed pipeline to solve more complex tasks such as the one in the Push environment. In order to do so, we would need to improve each part of the pipeline individually. It is possible that the disentanglement of the latent space in the Push environment may have been insufficient, which might be solved simply by using a larger training dataset and more hyperparameter tuning. We would also need to do a more thorough search to find a sufficiently good MLP/LSTM architecture for the predictors. For instance, recent literature suggests that state space models are most effective when they contain both deterministic and stochastic elements. We have not explored such approaches due to time constraints but they may improve the performance of our pipeline.

References

[1] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym, 2016.

[2] Lars Buesing, Theophane Weber, Sébastien Racanière, S. M. Ali Eslami, Danilo Jimenez Rezende, David P. Reichert, Fabio Viola, Frederic Besse, Karol Gregor, Demis Hassabis, and Daan Wierstra. Learning and querying fast generative models for reinforcement learning. *CoRR*, abs/1802.03006, 2018.

[3] Christopher P. Burgess, Irina Higgins, Arka Pal, Loic Matthey, Nick Watters, Guillaume Desjardins, and Alexander Lerchner. Understanding disentangling in β-vae. *arXiv e-prints*, page arXiv:1804.03599, April 2018.

[4] Kurtland Chua, Roberto Calandra, Rowan McAllister, and Sergey Levine. Deep reinforcement learning in a handful of trials using probabilistic dynamics models. *CoRR*, abs/1805.12114, 2018.

[5] Andreas Doerr, Christian Daniel, Martin Schiegg, Duy Nguyen-Tuong, Stefan Schaal, Marc Toussaint, and Sebastian Trimpe. Probabilistic recurrent state-space models. In *ICML*, 2018.

[6] Chelsea Finn, Xin Yu Tan, Yan Duan, Trevor Darrell, Sergey Levine, and Pieter Abbeel. Deep spatial autoencoders for visuomotor learning. *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 512–519, 2016.

[7] David Ha and Jürgen Schmidhuber. Recurrent world models facilitate policy evolution. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems 31*, pages 2451–2463. Curran Associates, Inc., 2018. `https://worldmodels.github.io`.

[8] Danijar Hafner, Timothy Lillicrap, Ian Fischer, Ruben Villegas, David Ha, Honglak Lee, and James Davidson. Learning latent dynamics for planning from pixels, 2018.

[9] Irina Higgins, Loic Matthey, Arka Pal, Christopher Burgess, Xavier Glorot, Matthew Botvinick, Shakir Mohamed, and Alexander Lerchner. beta-vae: Learning basic visual concepts with a constrained variational framework, 2016.

[10] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9:1735–1780, 1997.

[11] Stephen James, Andrew J. Davison, and Edward Johns. Transferring end-to-end visuomotor control from simulation to real world for a multi-stage task. In *CoRL*, 2017.

[12] Diederik P. Kingma and Max Welling. Auto-encoding variational bayes. *CoRR*, abs/1312.6114, 2013.

[13] A. Nagabandi, G. Kahn, R. S. Fearing, and S. Levine. Neural Network Dynamics for Model-Based Deep Reinforcement Learning with Model-Free Fine-Tuning. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 7559–7566, May 2018.

[14] Andrei A. Rusu, Matej Vecerik, Thomas Rothörl, Nicolas Heess, Razvan Pascanu, and Raia Hadsell. Sim-to-real robot learning from pixels with progressive nets. In *CoRL*, 2017.

[15] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, Yutian Chen, Timothy Lillicrap, Fan Hui, Laurent Sifre, George van den Driessche, Thore Graepel, and Demis Hassabis. Mastering the game of go without human knowledge. *Nature*, 550:354 EP –, 10 2017.

[16] Joshua Tobin, Rachel Fong, Alex Ray, Jonas Schneider, Wojciech Zaremba, and Pieter Abbeel. Domain randomization for transferring deep neural networks from simulation to the real world. *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 23–30, 2017.

[17] Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, pages 5026–5033. IEEE, 2012.

[18] Niklas Wahlstrom, Thomas B. Schön, and Marc Peter Deisenroth. Learning deep dynamical models from image pixels. *CoRR*, abs/1410.7550, 2014.

[19] Niklas Wahlstrom, Thomas B. Schön, and Marc Peter Deisenroth. From pixels to torques: Policy learning with deep dynamical models. *CoRR*, abs/1502.02251, 2015.

[20] Manuel Watter, Jost Tobias Springenberg, Joschka Boedecker, and Martin A. Riedmiller. Embed to control: A locally linear latent dynamics model for control from raw images. In *NIPS*, 2015.

[21] Fangyi Zhang, Jürgen Leitner, ZongYuan Ge, Michael Milford, and Peter I. Corke Peter. I. Corke. Adversarial discriminative sim-to-real transfer of visuo-motor policies, 2018.

[22] Fangyi Zhang, Jürgen Leitner, Michael Milford, and Peter I. Corke. Sim-to-real transfer of visuo-motor policies for reaching in clutter: Domain randomization and adaptation with modular networks. *CoRR*, abs/1709.05746, 2017.