PVEs: Position-Velocity Encoders for Unsupervised Learning of Structured State Representations

Rico Jonschkowski[1,2], Roland Hafner[1], Jonathan Scholz[1], and Martin Riedmiller[1]
[1]DeepMind, [2]Technische Universität Berlin

***Abstract*—We propose position-velocity encoders (PVEs) which learn—without supervision—to encode images to positions and velocities of task-relevant objects. PVEs encode a single image into a low-dimensional position state and compute the velocity state from finite differences in position. In contrast to autoencoders, position-velocity encoders are not trained by image reconstruction, but by making the position-velocity representation consistent with priors about interacting with the physical world. We applied PVEs to several simulated control tasks from pixels and achieved promising preliminary results.**

I. Introduction

While position and velocity are fundamental components of state representations in robotics, robots cannot directly sense these properties. Instead, they need to extract task-relevant position and velocity information from sensory input. For robots to be versatile, they must be able to learn such representations from experience. Deep learning allows to learn position-velocity representations in principle—but most existing approaches depend crucially on labelled data.

In this paper, we investigate how robots could learn position-velocity representations without supervision. We approach this problem by using robotics-specific prior knowledge about interacting with the physical world, also known as *robotic priors* [7], in order to learn an encoding from high-dimensional sensory observations to a low dimensional state representation. Our contribution is to split the state representation into a velocity state and a position state and to incorporate robotic priors about position and velocity in the form of model constraints and learning objectives.

Our method, the *position-velocity encoder* (PVE), implements a hard model constraint by estimating velocity states from finite differences in position states. This constraint fixes the relation between these two parts of the state representation. Additionally, PVEs include soft objectives that measure consistency with robotic priors. These objectives are optimized during learning and shape which information is encoded and how multiple state samples relate to each other. Both ingredients work together to learn an encoding into a structured state representation that includes *position states*, which describe information from a single observation, and *velocity states*, which describe how this information changes over time.

Figure 1 shows the position encoder that maps observations (blue rectangles) to position states (blue dots). The velocity state—the time derivative of the position state—is approximated from finite differences in position. This structured state space allows us to formulate new robotic priors, specifically for positions and velocities, in the form of learning objectives.

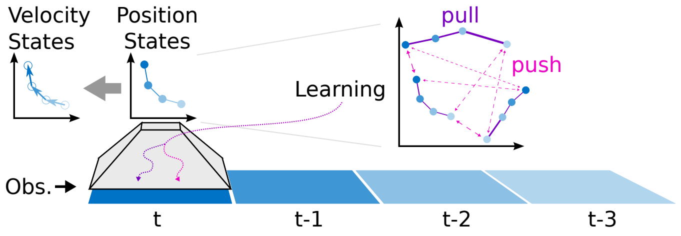


Fig. 1: PVEs encode an observation into a low-dimensional position state. From a sequence of such position states, they estimate velocities. PVEs learn the encoding by optimizing consistency of positions and velocities with robotic priors.

PVEs learn to encode observations into states without state labels and without learning a decoder. Instead, they learn the encoding by making position states and their derivatives consistent with different robotic priors. Inconsistency with each prior is measured in a loss function. PVEs learn by minimizing a weighted sum of these losses using gradient descent. The gradients can be imagined as forces in the state space that pull state samples together (when they should be similar) or push them apart (when they should be different). Backpropagation transforms these forces into parameter changes in the encoder (see pink and purple arrows in Fig. 1).

In our preliminary experiments, we apply position-velocity encoders to simulated control tasks from pixels. We show that PVEs are able to discover the topology and the dimensionality of the task, that they can learn equivalent representations from different camera perspectives, that they capture information about the true positions and velocities of physical objects in the scene, and that reinforcement learning based on the learned position-velocity state can produce precise control.

II. Related Work

This paper extends work on learning state representations with robotic priors by Jonschkowski and Brock [7], which introduced the idea of robotic priors and their implementation in the form of objectives for representation learning. Our extension to position-velocity states is inspired by work on physics-based priors in model-based reinforcement learning by Scholz et al. [16], which proposed to learn a physically plausible dynamics model given a position-velocity representation. Here, we turn their approach around and ask: How could we learn a position-velocity representation from sensory input without specifying which positions and velocities are relevant for the task?

The answer we are proposing, the position-velocity encoder, works by incorporating prior knowledge in two different ways, which fit Mitchell's categorization of *inductive biases* into restriction biases and preference biases [13, p. 64]. *Restriction biases* restrict the hypothesis space that is considered during learning, such as our constraint in the PVE model to estimate velocities from finite differences in position (rather than learning to extract velocity information from a sequence of observations). *Preference biases* express preferences for certain hypothesis, such as our loss functions for training PVEs, which measure inconsistency with robotic priors.

Other examples of restriction biases in the visual representation learning literature include convolutional networks [10], spatial transformer networks [5], and spatial softmax [11], which incorporate priors about visual input as architectural constraints of a neural network, as well as backprop Kalman Filters [3] and end-to-end learnable histogram filters [8], which incorporate the structure of the Bayes' filter algorithm for recursive state estimation. SE3-nets [2] implement assumptions about rigid body transformations. While these approaches can regularize learning, unsupervised learning (also) requires suitable preference biases.

Preference biases for internal representations are commonly expressed *indirectly* via other learnable functions based on the representation. For example, others train representations by using them to learn image reconstruction [21], prediction of future states [19, 21], or other auxiliary tasks [4, 12]. A powerful but underexplored alternative is to express preference biases *directly* on the learned representation, an approach which can enable to learn symbol grounding [6] or perform label-free supervised learning [18]. Direct preference biases are also the focus of metric learning, e.g. learning representations of faces using the fairly general triplet loss [17], but there is large potential for formulating more informative robotic priors in the form of direct preference biases, as we do in this work.

III. Position-Velocity Encoders

Position-Velocity Encoders (PVEs) learn to map raw observations into a structured state space that consists of a position part and a velocity part. PVEs are trained without state labels and they do not need to learn auxiliary functions such as reconstructing observations or predicting state transitions. PVEs achieve this by combining two key ideas:

1) PVEs encode the current observation into a *position state* and estimate a *velocity state* from finite differences in position (more details in Sec. III-A).
2) PVEs are trained by optimizing consistency with robotic priors about positions, velocities, and accelerations (more details in Secs. III-B & III-C).

A. Model

The PVE model consists of a convolutional network and a numerical velocity estimation. The convolutional network ϕ encodes a visual observation $\mathbf{o}_t$ into a low-dimensional position-state $\mathbf{s}_t^{(p)}$, where superscript (p) stands for *position*.

$$\mathbf{s}_t^{(p)} = \phi(\mathbf{o}_t).$$

From the difference of the last two position states $\mathbf{s}_t^{(p)}$ and $\mathbf{s}_{t-1}^{(p)}$, the model estimates the velocity state $\mathbf{s}_t^{(v)}$:

$$\mathbf{s}_t^{(v)} = \alpha(\mathbf{s}_t^{(p)} - \mathbf{s}_{t-1}^{(p)}),$$

where α is a hyperparameter that subsumes $\frac{1}{\text{timestep}}$ and scales velocity states. It is important that velocity states have the right scale relative to position states in order to create a sensible metric in the combined state $\mathbf{s}_t$, which we construct by stacking the position state and the velocity state.

$$\mathbf{s}_t = \begin{bmatrix} \mathbf{s}_t^{(p)} \\ \mathbf{s}_t^{(v)} \end{bmatrix}.$$

We can also use finite differences to estimate acceleration (or jerk, jounce, etc.). We do not include these derivatives in the state because we assume that the robot controls accelerations by its actions. But we do use the acceleration state in some loss functions. We compute the acceleration state $\mathbf{s}_t^{(a)}$ in the same way as the velocity state but we omit the scaling since we will not use accelerations in the combined state space:

$$\mathbf{s}_t^{(a)} = \mathbf{s}_t^{(v)} - \mathbf{s}_{t-1}^{(v)}.$$

B. Robotic Priors and Learning Objectives

The encoder ϕ is trained by making the combined state space consistent with a set of robotic priors, which we will describe in this section. These priors use the structured state space and are specific to positions, velocities, and accelerations. Consistency with these priors is defined in the form of loss functions that are minimized during learning. The following list of robotic priors should be understood as an exploration into this matter, not as a final answer.

1) ***Variation****: Positions of relevant things vary.* As the robot explores its task and manipulates its environment, the positions of task-relevant objects (including itself) will vary—otherwise there is not much that the robot could learn. If we assume that positions of relevant objects vary in the robot's experience, the internal representation of such positions must also vary; random pairs of position states should not be similar. Therefore, we optimize consistency with the variation prior by minimizing the expected similarity between random pairs of position states,

$$L_{\text{variation}} = \mathbf{E}\left[e^{-\|\mathbf{s}_a^{(p)} - \mathbf{s}_b^{(p)}\|}\right],$$

where we use $e^{-\text{distance}}$ as a similarity measure that is 1 if the distance is 0 and that goes to 0 with increasing distance between the position states, which is exactly what we want.

2) ***Slowness****: Positions change slowly* [22]. Physical objects do not teleport; they do not change their position arbitrarily from one second to the next. To make the internal position state consistent with the slowness prior, we minimize the expected squared distance between consecutive position states,

$$L_{\text{slowness}} = \mathbf{E}\left[\|\mathbf{s}_t^{(p)} - \mathbf{s}_{t-1}^{(p)}\|^2\right].$$

Since this change in position is directly connected to the rate of position change (or velocity), we can also write down the same loss using the velocity state.

$$L_{\text{slowness}} = \mathbf{E}\left[\left\|\frac{\mathbf{s}_t^{(v)}}{\alpha}\right\|^2\right],$$

where α is the scaling hyperparameter defined earlier. This reformulation hints at a different interpretation of slowness, which is simply: *velocities are low*[1].

3) ***Inertia****: Velocities change slowly.* Since physical objects have inertia, they resist changes to their velocity (both in direction or magnitude). If we assume limited forces to overcome this resistance, velocities should only change by small amounts. Note how the inertia prior corresponds to the slowness prior applied to velocities.

$$L_{\text{inertia}} = \mathbf{E}\left[\|\mathbf{s}_t^{(v)} - \mathbf{s}_{t-1}^{(v)}\|^2\right] = \mathbf{E}\left[\|\mathbf{s}_t^{(a)}\|^2\right].$$

This formulation of the inertia prior focuses on large velocity changes due to the square in the loss function. Alternatively, we can define the loss function based on *absolute* changes.

$$L_{\text{inertia (abs)}} = \mathbf{E}\left[\|\mathbf{s}_t^{(a)}\|\right].$$

Small changes in velocity have a higher weight in the second loss compared to the first loss. We found that combining both losses leads to better results than using either one of them.

4) ***Conservation****: Velocity magnitudes change slowly.* This prior derives from the *law of conservation of energy*, which states that the total energy in a closed system remains constant. As the robot applies forces to the environment, we do not have a closed system. Additionally, we cannot estimate, e.g. kinetic energy without knowing the masses of objects, let alone potential energy stored in springs etc. Still, we want to enforce the same idea of keeping the absolute amount of energy, or in our case "movement" similar in consecutive time steps.

$$L_{\text{conservation}} = \mathbf{E}\left[\left(\|\mathbf{s}_t^{(v)}\| - \|\mathbf{s}_{t-1}^{(v)}\|\right)^2\right].$$

5) ***Controlability****: Controllable things are relevant.* The objects that can be controlled by the robot are likely relevant for its task. If the robot acts by applying forces, controllable things could be those whose accelerations correlate with the actions of the robot. Accordingly, we can define a loss function per action dimension i to optimize covariance between action dimension i and accelerations in a state dimension i.

$$\begin{aligned} L_{\text{controlability (i)}} &= e^{-\text{Cov}(\mathbf{a}_{t,i}, \mathbf{s}_{t+1,i}^{(a)})} \\ &= e^{-\text{E}\left[\left(a_{t,i} - \text{E}[a_{t,i}]\right)\left(s_{t+1,i}^{(a)} - \text{E}[s_{t+1,i}^{(a)}]\right)\right]}. \end{aligned}$$

Note that we used this loss in only one of the tasks—ball in cup—because the above priors were insufficient. The results for this task are still preliminary. A complete solution of this task and a deeper investigation into other formulations of controlability are part of future work.

[1]Note that defining the slowness prior to mean *velocities are low* translates to the loss function $L_{\text{slowness}} = \mathbf{E}[(\mathbf{s}_t^{(v)})^2] = \mathbf{E}[(\alpha(\mathbf{s}_t^{(p)} - \mathbf{s}_{t-1}^{(p)}))^2]$, which depends on the scaling parameter α. We use the other formulation to make this loss independent of α because we want to change α during training without affecting this loss (see Sec III-C for more details).

C. Training Procedure

We train PVEs by minimizing a weighted sum of the loss functions described above using gradient descent. This section explains the training procedure in detail.

1) *Data Gathering:* First, the robot gathers data by exploring its environment. Since we are using a reinforcement learning setting, the data consist of sequences of observations, actions, and rewards. Most of the presented loss functions only use observations, the controlability loss also uses actions, but none of our current losses uses the reward signal.

2) *Loss Computation:* We iterate through the collected data in mini batches, which consist of a small set of short sequences. For each mini-batch, we compute the loss functions by replacing expectations with statistical averages.[2]

3) *Loss Combination:* We combine these losses in a weighted sum. Finding the right weights is important because they balance how much each prior is enforced during learning. We determined these weights empirically by adjusting them until the gradients in the encoder parameters had similar magnitudes for all priors. Future work should try to automate this process of weight tuning, potentially by applying the same heuristic in an automated fashion.

4) *Parameter Updates:* For each mini-batch, we compute the gradient[3] of the combined loss with respect to the enoder parameters using symbolic auto-differentiation [1] and perform an update using the Adam optimizer [9]. We iterate this process until convergence.

5) *Velocity Scaling Curriculum:* While training PVEs, we follow a curriculum that in the beginning focuses on positions and only later also takes velocities into account. This curriculum is implemented by changing the velocity scaling parameter α. In the first phase, we train with $\alpha = 0$ until convergence. In the second phase, we increase α linearly from 0 to its final value and train until convergence again. In phase one, only the first two priors, variation and slowness, are active. Surprisingly, these two are powerful antagonists that can unfold the topology of the position-state space. The second phase mainly smooths the state space such that velocities can be accurately estimated from finite differences.

6) *Hyperparameters:* We used the following hyperparemeters in our experiments. The convolutional network had three convolutional layers with 16, 32, and 64 channels, kernel size 5x5, and stride 2, followed by three fully connected layers of sizes 128, 128, and 5 (for a 5-dimensional position state). Every layer except the last one was followed by a ReLu nonlinearity [14]. The mini-batch size was 32 sequences of 10 steps each. The maximum velocity scaling α was 10. The weights for the different losses are shown in Table I.

IV. Experiments and Results

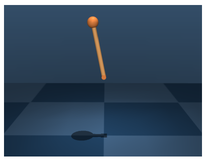
We applied PVEs to a series of simulated control tasks from raw pixel input (see Fig. 2). All tasks use the Mujoco

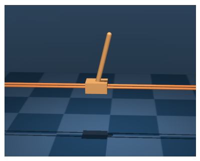
[2]For the variation loss, we sample all pairs of experiences with the same time step in different sequences of the mini batch. For all other losses we consider all samples in the mini batch.

[3]Some of the gradients can only be computed after adding small Gaussian noise to the encoded states.

TABLE I: Loss weights per task (same for a and b).

	Task a	Task b	Task c
Variation	1.0	1.0	1.0
Slowness	1.0	1.0	1.0
Inertia	0.1	0.1	0.001
Inertia (abs)	0.1	0.1	0.02
Conservation	0.2	0.2	0.005
Controlability (i=1,2)	0.0	0.0	0.5


(a) Inverted pendulum


(b) Cart-pole

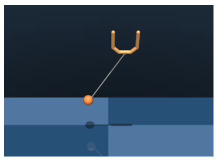
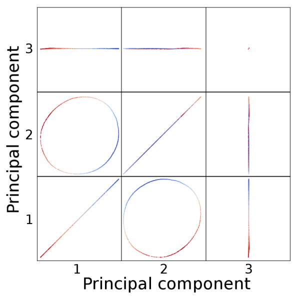
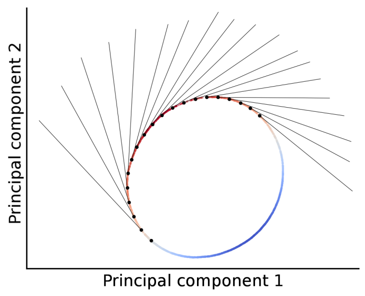

(c) Ball in cup

Fig. 2: Three control tasks from pixel input



(a) Encoded position states



(b) Overlayed state sequence

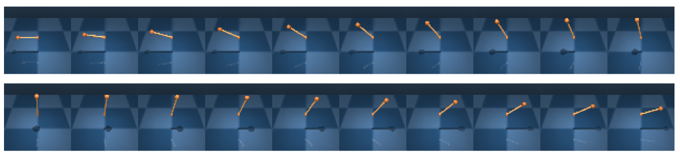

(c) Observation sequence

Fig. 3: For the inverted pendulum, PVEs learn a circular position representation that allows accurate velocity estimation. Each dot in (a) and (b) is the encoding of a single observation. The color denotes the reward received with the observation (red = high, blue = low). Black dots in (b) show the encoding of the observation sequence in (c). Black lines show the estimated velocities. **Supplementary videos**: http://youtu.be/ipGe7Lph0Lw shows the learning process, http://youtu.be/u0bQwz89h1I demonstrates the learned PVE.

simulator [20]. For each task, we collected a batch of training data that consists of 1000 short trajectories of 20 steps by randomly sampling start configurations with different positions and velocities and applying a random policy.

A. *Tasks*

1) Inverted Pendulum: The inverted pendulum is a stick that is fixed to a rotary joint at one end. The goal is to swing it up and balance it upright by applying forces at the joint. However, the motor is not strong enough to pull the pendulum up all at once. Instead the pendulum must be swung back and forth to generate enough (but not too much) momentum.

2) Cart-Pole: The cart-pole task is an extension of the inverted pendulum task. Since the pole is attached to a cart with a passive joint, it can only be swung up by accelerating the car correctly, which requires precise control.

3) Ball in Cup: This task includes a cup and a ball attached to the bottom of the cup with a string. The goal is to move the cup in such a way that the ball lands in the cup. In our version of the task, cup and ball can only move in the plane.

B. *Learned Position-Velocity Representations*

For each task, we will now look at the learned state representations. We visualize 5-dimensional position-states by projecting to their principal components.

1) Inverted Pendulum: The state representation learned by the PVE is shown in Figure 3a, where we can see the encoding of test observations into the position-state space. Each dot is the position encoding of a single image. The color denotes the amount of reward that was achieved in that instance.

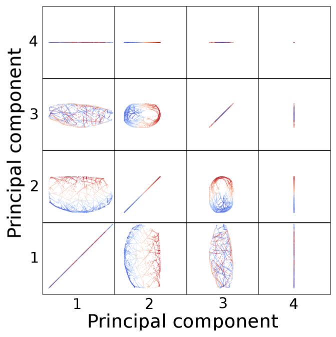
The plot shows a number of interesting results. First, observations that correspond to similar rewards are encoded close together in position space. Second, the position states form a circle, which makes sense because the inverted pendulum moves in a circle. Third, all principal components after the first two are close to zero. This means that the circular encoding lies on a plane in the five-dimensional space—the PVE discovered that the task is two dimensional[4].

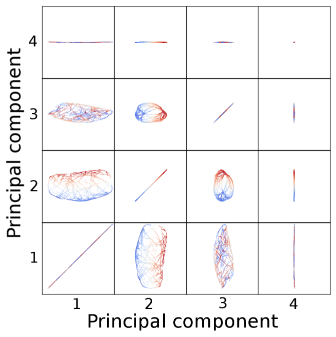
Next, we will look at the estimated velocities in the learned space. In Figure 3b, we overlayed encoded training data colored by reward with the encoding of a single sequence of observations shown in Figure 3c. The position states are marked with black dots and the velocity state vectors are drawn as lines. In the observation sequence, the pendulum swings from the left side to the top and then to the right side. Similarly, the encoded positions move from a medium reward region via the high-reward region (red color) to the medium reward region on the other side. During this motion, the velocity estimations are tangential to the circle in the position space with minimal noise, which should be useful for controlling the pendulum (see video links in Fig. 3).

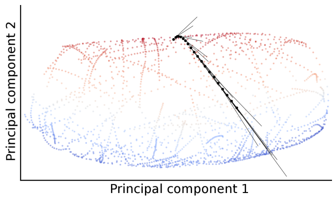
2) Cart-Pole: Here, we compare PVEs on two different observations: 1) using a moving camera that follows the cart by rotating sideways, 2) using a static camera that covers the entire region in which the cart moves. Figure 4 shows the learned position representations for both perspectives.

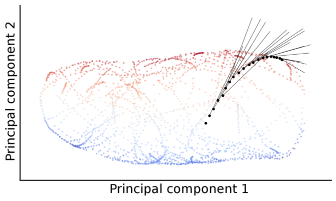
This experiment demonstrates how PVEs can learn equivalent internal representations (compare Figs. 4a and 4b) from observations that look very different (Figs. 4e, 4f). For both kinds of observations, the state samples form a tube, the length

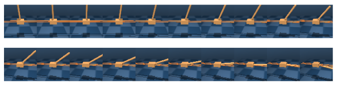
[4]Even though the task only has one positional degree of freedom (the angle of the pendulum), we need at least two dimensions if we want a Euclidean metric to make sense in this space, such that there are no jumps as from 360 to 0 degrees in an angular representation.


(a) Learned representation (moving camera)


(b) Learned representation (static camera)


(c) Encoded sequence (moving camera)


(d) Encoded sequence (static camera)


(e) Observation sequence (moving camera)

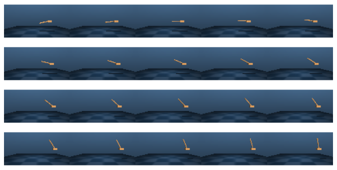
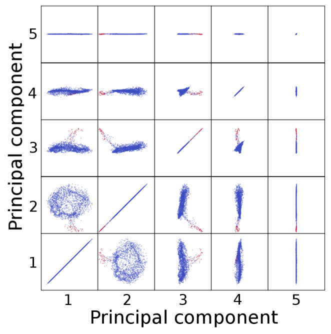
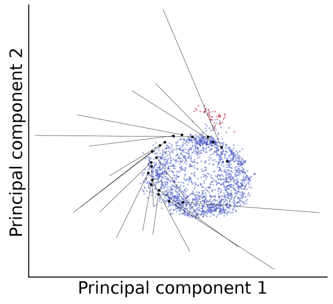

(f) Observation sequence (static camera)

Fig. 4: For cart-pole, PVEs learn equivalent state representations from different observations. **Supplementary videos**: learning process for the moving camera http://youtu.be/RKlciWWuJfc and static camera http://youtu.be/MYxrA1Bw6MU, learned PVE with the moving camera http://youtu.be/67QZRsLNTAE.


(a)


(b) Inverted pendulum

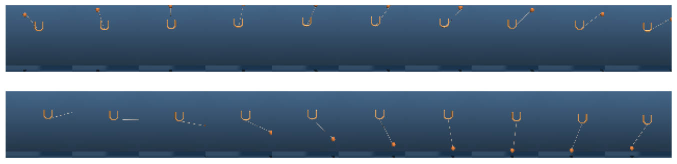

(c) Observation sequence

Fig. 5: Learned position-velocity representation for ball in cup. **Supplementary videos**: http://youtu.be/3fLaSL8d4TY shows the learning process, http://youtu.be/lIhEGv5kLFo demonstrates the learned PVE.

of which corresponds to the position of the cart, while the circular part represents the position of the pole. Here, the PVE uses three of the five dimensions and thereby discovers the three-dimensional nature of the given task.

The observation sequence from the moving camera (Fig. 4e) shows the cart moving to the left while the pole falls down on the right side. The PVE represents this trajectory (Fig. 4c) by moving from the high-reward red region to the blue region, which reflects the movement of the pole, and to the right side, which corresponds to sideways movement of the cart. The observation sequence from the static camera (Fig. 4f) shows the pole swinging up while the cart moves to the right. Accordingly, the encoded trajectory (Fig. 4d) goes to the red region and to the right side (right and left a swapped between these two representations).

3) Ball in Cup: The results for this task are preliminary. The task is challenging due to the movement of the cup, which is inconsistent with some of our robotic priors. The cup is confined to a small region and controlled by the robot allowing rapid movements and changes of direction. The cup can be moved from one end of its position range to the other end in a few time steps. Therefore, the slowness prior does not hold here (unless we sampled observations at a higher frequency). Additionally, the robot can apply large forces on the cup, leading to large accelerations and jerky movements, which are again inconsistent with many of our priors on changes in velocity. As a result, PVEs struggle with encoding the cup, which we will quantify in the following section.

To approach this problem, we added the controllability prior, which enforces that things controlled by the robot are encoded into the state. This improved the resulting state representation (see Fig. 5). While the semantics of the state representation are not as clear as for the previous tasks, the representation uses four dimensions, which makes sense for two objects in a plane. Additionally, the goal states (ball in cup) are clearly separated from the other states. As we will see in the following section, the information about the cup is still very noisy, which is probably why reinforcement learning based on PVEs does not reach the same performance as in the other tasks. This result makes the ball in cup task a good candidate for a next step on extending PVEs by revising and adding robotic priors.

C. Regression to True Positions and Velocities

To measure the quality of the learned position-velocity representation, we performed regression from the learned postion-velocity state to true positions and velocities of relevant objects. Here, we trained a fully connected neural network with 3 hidden ReLu layers of 256 units each for 200 steps with Adam. We normalized the true positions and velocities and performed supervised learning from the learned position-velocity state to the true features minimizing mean squared error. After training on observations from 1000 times 20 steps, we tested with 100 times 20 test samples. The resulting test errors are shown in Table II.

TABLE II: Comparison of mean squared test errors.

Inverted pendulum	
$\cos(\theta_{\text{pole}})$	0.0003
$\sin(\theta_{\text{pole}})$	0.0002
$\dot{\theta}_{\text{pole}}$	0.0003

Cart-pole (different cameras)		
	moving	static
x_{cart}	0.0007	0.0015
$\cos(\theta_{\text{pole}})$	0.0013	0.0021
$\sin(\theta_{\text{pole}})$	0.0012	0.0033
$\dot{x}_{\text{cart}}$	0.0069	0.0198
$\dot{\theta}_{\text{pole}}$	0.0110	0.0264

Ball in cup	
x_{cup}	0.0622
y_{cup}	0.0645
x_{ball}	0.0187
y_{ball}	0.0294
$\dot{x}_{\text{cup}}$	0.6654
$\dot{y}_{\text{cup}}$	0.6372
$\dot{x}_{\text{ball}}$	0.1535
$\dot{y}_{\text{ball}}$	0.2359

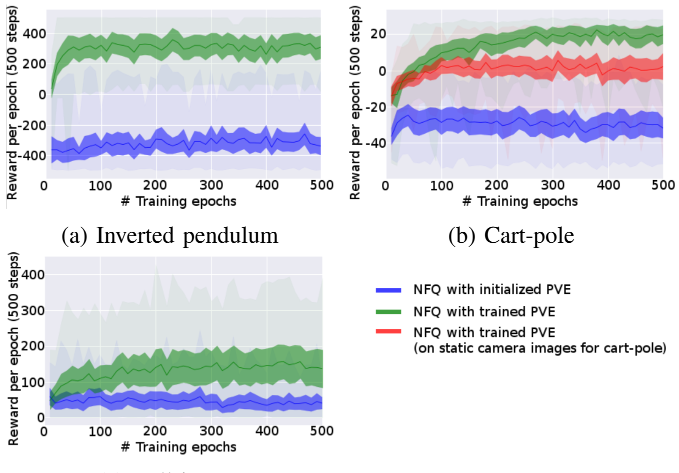
When we compare these errors, we find that the errors are lowest for the pendulum task, which makes sense because the range of possible observations is so small in this task, that it is well covered by the training data. For the cart-pole the errors are still very low for position, but higher for the estimated velocities because noise in the position states is increased when computing velocities from finite differences. Also, the errors double when we go from the moving camera setting to the static camera setting. From this difference, we can predict that control should be easier in the first setting. Finally, for ball in cup, the errors are again much larger for the reasons discussed earlier. The estimation of the cup velocity is particularly challenging.

Note that we performed this regression test to measure how well these properties are encoded in the state. We do not use the state labels for training the representation and we do not use them for learning control. In the following section, we will measure the utility of the learned representation by reinforcement learning performance based on these representations.

D. Enabling Reinforcement Learning

In this experiment, we learn control for these tasks with neural fitted Q-iteration (NFQ, [15]) based on the encoding learned by PVEs. As a baseline, we use untrained PVEs with randomly initialized encodings in this preliminary work (we will thoroughly compare to other methods in future work). For the policy, we used a fully connected neural network with two hidden layers of 250 sigmoid units per layer. We trained it two times for 30 episodes after each training epoch. We rescaled rewards to be non-positive and ommitted discounting. We repeated actions for multiple time steps (4 for the pendulum and cart-pole tasks, 6 for the ball in cup task).

The resulting learning curves are shown in Figure 6. The blue curves show the baselines with random encodings, which do not allow learning any of the three tasks. The green and red curves show reinforcement learning based on PVEs that were trained on a batch of 1000 trajectories of 20 steps. For the inverted pendulum and for the cart-pole task, the green curves



(a) Inverted pendulum (b) Cart-pole (c) Ball in cup

Fig. 6: Reinforcement learning performance for different tasks based on state representations learned by PVEs. Lines show means of 50 trials, darker shading shows standard errors, lighter shading shows range from minimum to maximum values.

reach optimal performance after only 50 and 300 epochs. The red curve, which shows the performance based on the static camera perspective does not reach optimal performance, probably due to the more noisy state estimation discussed in the previous section. At this point, it is not clear whether this issue comes from the low resolution in the input or from the fact that the position of the pole and the cart are more strongly coupled in these observations which makes learning the state encoding more difficult. Lastly, for the ball in cup task, the learned control beats the baseline consistently and (as the light green maximum shading shows) more successful control using the learned representation is possible. But due to the noisy state estimation, this is not sufficient for solving the task consistently. Future work could start from here and investigate which priors are missing to solve this and more realistic robotic tasks.

V. Conclusion and Future Work

We have presented position-velocity encoders (PVEs), which are able to learn state representations structured into a position and a velocity part without supervision and without requiring image reconstruction. The keys to PVEs are to constrain the model to estimate velocities in the correct way from positions and to train the position encoder by optimizing consistency with robotic priors, which are specific to positions, velocities, and accelerations. We have shown how structuring the state space into positions and velocities opens up new opportunities for formulating useful constraints and learning objectives. In future research, we will work towards adding further structure into the state space, revising and extending the list of robotic priors, and combining these approaches with end-to-end reinforcement learning.

References

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[2] Arunkumar Byravan and Dieter Fox. SE3-nets: Learning rigid body motion using deep neural networks. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2017.

[3] Tuomas Haarnoja, Anurag Ajay, Sergey Levine, and Pieter Abbeel. Backprop kf: Learning discriminative deterministic state estimators. In *Advances in Neural Information Processing Systems (NIPS)*, pages 4376–4384, 2016.

[4] Max Jaderberg, Volodymyr Mnih, Wojciech Marian Czarnecki, Tom Schaul, Joel Z Leibo, David Silver, and Koray Kavukcuoglu. Reinforcement learning with unsupervised auxiliary tasks. In *International Conference on Learning Representations (ICLR)*, 2016.

[5] Max Jaderberg, Karen Simonyan, Andrew Zisserman, et al. Spatial transformer networks. In *Advances in Neural Information Processing Systems (NIPS)*, pages 2017–2025, 2015.

[6] Nikolay Jetchev, Tobias Lang, and Marc Toussaint. Learning grounded relational symbols from continuous data for abstract reasoning. In *Autonomous Learning Workshop at the IEEE International Conference on Robotics and Automation*, 2013.

[7] Rico Jonschkowski and Oliver Brock. Learning state representations with robotic priors. *Autonomous Robots*, 39(3):407–428, 2015.

[8] Rico Jonschkowski and Oliver Brock. End-to-end learnable histogram filters. In *Workshop on Deep Learning for Action and Interaction at NIPS*, December 2016.

[9] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations (ICLR)*, 2014.

[10] Yann LeCun, Yoshua Bengio, et al. Convolutional networks for images, speech, and time series. *The handbook of brain theory and neural networks*, 3361(10), 1995.

[11] Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. End-to-end training of deep visuomotor policies. *Journal of Machine Learning Research (JMLR)*, 17(39):1–40, 2016.

[12] Piotr Mirowski, Razvan Pascanu, Fabio Viola, Hubert Soyer, Andy Ballard, Andrea Banino, Misha Denil, Ross Goroshin, Laurent Sifre, Koray Kavukcuoglu, Dharshan Kumaran, and Raia Hadsell. Learning to navigate in complex environments. In *International Conference on Learning Representations (ICLR)*, 2016.

[13] Thomas M. Mitchell. *Machine Learning*. McGraw-Hill, Inc., New York, NY, USA, 1997.

[14] Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *International Conference on Machine Learning (ICML)*, pages 807–814, 2010.

[15] Martin Riedmiller. Neural fitted Q iteration – first experiences with a data efficient neural reinforcement learning method. In *European Conference on Machine Learning (ECML)*, pages 317–328, 2005.

[16] Jonathan Scholz, Martin Levihn, Charles Isbell, and David Wingate. A physics-based model prior for object-oriented MDPs. In *International Conference on Machine Learning (ICML)*, 2014.

[17] Florian Schroff, Dmitry Kalenichenko, and James Philbin. FaceNet: A unified embedding for face recognition and clustering. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 815–823, 2015.

[18] Russell Stewart and Stefano Ermon. Label-free supervision of neural networks with physics and domain knowledge. In *AAAI Conference on Artificial Intelligence*, 2016.

[19] Wen Sun, Arun Venkatraman, Byron Boots, and J. Andrew Bagnell. Learning to filter with predictive state inference machines. In *International Conference on Machine Learning (ICML)*, pages 1197–1205, 2016.

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

[21] Manuel Watter, Jost Tobias Springenberg, Joschka Boedecker, and Martin Riedmiller. Embed to control: A locally linear latent dynamics model for control from raw images. In *Advances in Neural Information Processing Systems (NIPS)*, pages 2746–2754, 2015.

[22] Laurenz Wiskott and Terrence J. Sejnowski. Slow feature analysis: unsupervised learning of invariances. *Neural Computation*, 14(4):715–770, 2002.