# S-RL Toolbox: Environments, Datasets and Evaluation Metrics for State Representation Learning

Antonin Raffin, Ashley Hill, René Traoré, Timothée Lesort, Natalia Díaz-Rodríguez, David Filliat

**HAL Id: hal-01931713**

**https://hal.science/hal-01931713**

Submitted on 29 Nov 2018

# S-RL Toolbox: Environments, Datasets and Evaluation Metrics for State Representation Learning

Antonin Raffin     ANTONIN.RAFFIN@ENSTA-PARISTECH.FR

Ashley Hill     ASHLEY.HILL@ENSTA-PARISTECH.FR

René Traoré     RENE.TRAORE@ENSTA-PARISTECH.FR

Timothée Lesort     TIMOTHEE.LESORT@ENSTA-PARISTECH.FR

Natalia Díaz-Rodríguez     NATALIA.DIAZ@ENSTA-PARISTECH.FR

David Filliat     DAVID.FILLIAT@ENSTA-PARISTECH.FR

*U2IS, ENSTA ParisTech / INRIA FLOWERS Team* `http://flowers.inria.fr`
*Palaiseau, France*

## Abstract

State representation learning aims at learning compact representations from raw observations in robotics and control applications. Approaches used for this objective are auto-encoders, learning forward models, inverse dynamics or learning using generic priors on the state characteristics. However, the diversity in applications and methods makes the field lack standard evaluation datasets, metrics and tasks. This paper provides a set of environments, data generators, robotic control tasks, metrics and tools to facilitate iterative state representation learning and evaluation in reinforcement learning settings.

**Keywords:** Deep learning, reinforcement learning, state representation learning, robotic priors

## 1. Introduction

Robotics control relies on compact and expressive representations of sensor data, as the task objectives are often expressed in much smaller dimensions than the sensor space dimension (e.g., the position of an object versus the size of an image). These representations are usually hand crafted by human experts, but deep-learning now makes it possible to avoid this feature engineering by using end-to-end learning (e.g., learning a policy from raw pixels). However, such approach is mostly possible in simulation because it requires a huge amount of training data –usually millions of samples– that makes it impractical in the real world. To overcome this issue, State Representation Learning (SRL) methods (Jonschkowski, 2018) can be used to create an intermediate representation that should contain only useful information to control a robot and thus simplify the policy learning task.

Many different SRL approaches have been proposed (see (Lesort et al., 2018) for a review), but comparing their performances is challenging. A common approach to evaluate learned representations is to compare performance in a Reinforcement Learning (RL) setting. However, because of the instability of RL algorithms and their cost, this should not be the only method used to assess learned states. Moreover, this approach gives no mean to interpret a state representation, making it difficult to understand which information is encoded in this representation.

While Reinforcement Learning has well established benchmarks, SRL has no metrics nor universal criterion to compare the different approaches. With that in mind, we propose a set of environments with an increasing difficulty, designed for comparing State Representation Learning (SRL) methods for robotic control. We also introduce qualitative and quantitative metrics along with visualization tools to facilitate the development and the comparison of SRL algorithms. The proposed framework allows fast iteration and eases

research of new SRL methods by making it easy to produce statistically relevant results: the simulated environments run at 250 FPS on a 8-core machine that allows to train a RL agent on 1 Million steps in only 1h (or to generate 20k samples in less than 2 min)[1].

In this paper, we first present quickly the reinforcement learning framework and the main state representation learning approaches that are implemented, before presenting the SRL Toolbox environments and datasets, the qualitative and quantitative evaluation methods, and a set of experiments illustrating the performances of the implemented approaches.

## 2. Reinforcement Learning and State Representation Learning

In this section, we introduce Reinforcement Learning (RL), as well as the State Representation Learning (SRL) approaches we integrated in our framework.

### 2.1 Reinforcement Learning

In RL, an agent must learn to select the best action to maximize a reward it will receive. More formally, in a given state $s_t$ ($t$ denotes the current time-step), an agent performs an action $a_t$ and receives a reward $r_t$. The learned behaviour, that should maximize the long-term discounted reward by mapping states to actions, is called a policy: $a_t = \pi(s_t)$.

In the most common settings, the state $s_t$ is either a low dimension representation given by an expert human, or corresponds to the raw observation (end-to-end learning). In order to differentiate these cases, we introduce the observation $o_t$ that correspond to the raw sensor data, and use the term *state* only to refer to low dimensional representations that can be provided by humans, or can be learned using SRL.

In this paper, we work in the context of Markov Decision Processes (MDP), where the next state of the system only depends on the previous state and the taken action. As described above, the term *observations* refers to raw sensor data (mostly images), but do not imply partial observability as assumed in Partially Observable Markov Decision Processes (POMDP).

### 2.2 State Representation Learning

SRL (Lesort et al., 2018) aims at learning compact representations from raw observations (e.g., learn a position $(x, y)$ directly from raw pixels) without explicit supervision. Most of the time, the goal is to use that representation to solve a task with RL. The idea is that a *low-dimensional representation* should only keep the useful information and reduce the search space, thus contributing to address two main challenges of RL: sample inefficiency and instability. Moreover, a state representation learned for a particular task may be transferred to related tasks and therefore speed up learning in multiple task settings.

Using RL notations, SRL corresponds to learning a transformation[2] $\varphi$ from the observation space $\mathcal{O}$ to the state space $\mathcal{S}$. Then, a policy $\pi$, that takes a state $s_t \in \mathcal{S}$ as input and outputs action $a_t$, is learned to solve the task:

$$o_t \xrightarrow[SRL]{\varphi} s_t \xrightarrow[RL]{\pi} a_t \tag{1}$$

In the next sections, we present approaches of SRL that are implemented in our toolbox. Each method is not mutually exclusive and can be combined to create new models.

---

1. Environments, code and data are available at `https://github.com/araffin/robotics-rl-srl`
2. In practice, the learned transformation is a neural network

### 2.2.1 Auto-encoders (AE, VAE)

A first approach to learn a state representation is to compress the observation into a low dimensional state that is sufficient to reconstruct the observation. This approach does not take advantage of the robotic context because it ignores the possible actions, therefore it is often associated with different objectives (e.g. forward model), and provides a performance baseline.

We integrated Auto-Encoders (Baldi, 2012) and Variational Auto-encoders (VAE) (Kingma and Welling, 2013), i.e., auto-encoders that enforce the latent variables to follow a given distribution.

### 2.2.2 Robotic Priors

To build a relevant representation of states, one can use prior knowledge about the dynamics or physics of the world. This knowledge can account for temporal continuity or causality principles that reflect the interactions of an agent with its environment (Jonschkowski and Brock, 2015; Jonschkowski et al., 2017). *Robotic Priors* are defined as objective functions that constrain the state representation.

### 2.2.3 Forward and Inverse Models

The dynamics of the world can be integrated by learning a forward model that predicts state $s_{t+1}$ given state $s_t$ and action $a_t$. Constraints on the state representation can be added by constraining the forward model, for instance, by enforcing the system to follow linear dynamics (Watter et al., 2015).

Another approach is to learn an inverse model (Shelhamer et al., 2017; Pathak et al., 2017), which predicts the taken action $a_t$ given two successive states $s_t$ and $s_{t+1}$. This enforces the states to encode information about the dynamics, in order to recover the action needed for such transition.

### 2.2.4 Combining Approaches

Auto-encoders tend to reconstruct everything (that is salient enough in the observation), including static objects and irrelevant features for the task (distractors), whereas forward and inverse models focus on the dynamics, usually encoding the position of the controlled robot in the representation, but not a goal that is not controlled by the actions.

However, these approaches are not mutually exclusive and can be combined to create improved state representations. For instance, (Pathak et al., 2017) combine both inverse and forward models and (Zhang et al., 2018) additionally integrate an auto-encoder. In the same vein, Ha and Schmidhuber (2018) use a VAE with a recurrent forward model to learn a state representation.

## 3. Datasets and Environments

In this section, we describe a set of environments with incremental difficulty, designed to assess SRL algorithms for robotic control. They all follow the interface defined by OpenAI Gym (Brockman et al., 2016), which makes integration with RL algorithms easy.

### 3.1 Environments Details

The settings we propose (Fig. 1) are variations of two environments: a 2D environment with a mobile robot and a 3D environment with a robotic arm. In all settings, there is a controlled robot and one or more targets (that can be static, randomly initialized or
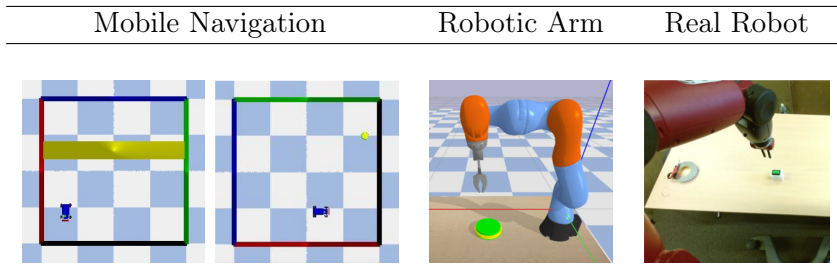
Figure 1: Environments and datasets for state representation learning.

moving). Each environment can either have a continuous or discrete action space, and the reward can be sparse or shaped, allowing us to cover many different situations.

*Static & random target mobile navigation*: This setting simulates a navigation task using a small car resembling the task of (Jonschkowski and Brock, 2014), with either a cylinder or a horizontal band on the ground as a goal, which can be fixed or moving from episode to episode. The car can move in four directions (*forward, backward, left, right*) and will get a +1 reward when reaching the target, -1 when hitting walls, and 0 otherwise.

*Static & random target robotic arm*: This setting simulates a robotic arm (Kuka), fixed on a table, with the task of pushing a button that may move or not in between episodes. The arm can be controlled either in the $x$, $y$ and $z$ position using inverse kinematics, or directly controlling the joints. The robot will get +1 reward when the arm pushes the button, -1 when it hits the table (this will end the episode), and 0 otherwise. A variant adds distractors, i.e., moving objects on the table irrelevant for solving the task.

*Simulated & real robotic arm*: We used a real Baxter robot arm (Gazebo in simulation) to perform the same button pushing task as the previous task, with the same actions and the same rewards. The goal is to test the different methods in a real world setup.

A ground truth state is defined in each scenario: the absolute robot position in static scenarios and the relative position (w.r.t. the target) in moving goal scenarios. Note that apart from providing all described environments, we also provide the corresponding datasets used in our evaluations: images are 224x224 pixels, navigation datasets use 4 discrete actions (*right, left, forward, backward*); robot arms use one more (*down*) action.

In section A.2, we provide baselines results (Ground Truth, Auto-Encoder, Raw Pixels) for each environment.

## 3.2 Motivation of the Goal-Based Robotics Tasks

The environments proposed have several characteristics that make them suitable for research and benchmarking.

*Designed for Robotics:* The proposed environments cover basic goal-based robotics tasks: navigation for a mobile robot and reaching a desired position for a robotic arm.

*Designed for State Representation Learning:* The simplicity of the environments makes the extracted features easier to interpret (correlation can be computed between learned states and position of relevant objects). It is also clear what a good state representation should encode because of the small number of important elements: there is only the controllable robot and the target.

*Designed for Research:* The environments have incremental difficulty, the minimal number of variables for describing each environment (minimal state dimension for solving the task with RL) is increasing from 2 (mobile robot with static target) to 6 (robotic arm with random target). Having simple environments of gradual difficulty is really important when developing new methods. The proposed environments are also easily customizable so that they cover all possibilities: reward can be sparse/dense, actions can be continuous/discrete.

Finally, our benchmark is completely free and fast (it runs at 250 FPS on a 8-core machine with one GPU).

## 4. Evaluation of Learned State Representations

The most practical evaluation of SRL is assessing if the learned states can be used for solving the task in RL. However, algorithm development can benefit from other metrics and visually assessing the validity of the representation being learned, for faster iteration and interpretation of the state embedding space. We provide tools and metrics for this.
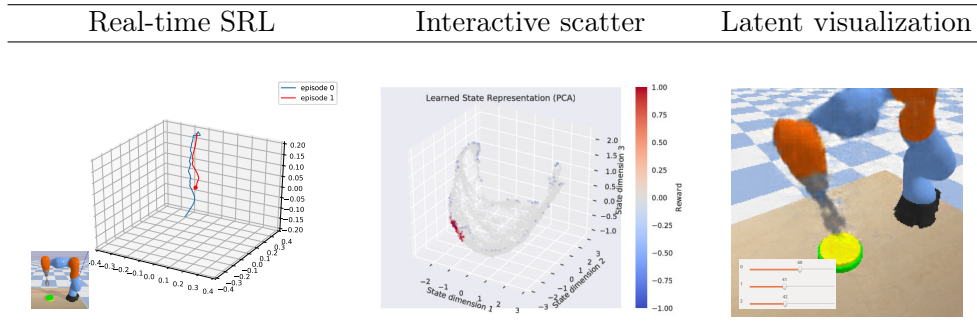
### 4.1 Qualitative Evaluation



Figure 2: Visual tools for analysing SRL; Left: Live trajectory of the robot in the state space. Centre: 3D scatter plot of a state space; clicking on any point displays the corresponding observation. Right: reconstruction of the point in the state space defined by the sliders. See complementary material for videos.

Qualitative evaluation in our case is the perceived utility of the state representation using visualization tools. The perceived utility depends on the task at hand. For example, the state representation of the robotic arm dataset is expected to have a continuous and correlated change with respect to the arm tip position. Three tools are proposed (Fig. 2):

*Real-time SRL*: This tool is used in conjunction with a graphical interface of the simulated environment. It shows the correspondence between observation and state by plotting current position of the observation in the state representation.

*Interactive scatter*: A clickable plot of the state representation, with the reward defining the colour for each point. Here, we expect to visualize structure in the state representation.

*Latent visualization*: It allows to navigate in the latent space by projecting the state to the observation space. This is achieved either by reconstructing the output (for AE and VAE), or by using a nearest neighbour approach for the models lacking a reconstruction.

To deal with state dimensions larger than three, PCA is used to visually assess the learned representations in a qualitative manner.

These tools allow to have better insights of the learned representation over different settings of state representation learning. This is especially useful in situations where the state dimension is greater than 3. This way, we are able to validate different configurations before running more exhaustive and time consuming methods.

### 4.2 Metrics

#### 4.2.1 KNN-MSE

We use an assessment of the representation's quality based on a Nearest-Neighbours approach (as in (Sermanet et al., 2017)). While the nearest neighbour coherence can be

assessed visually, KNN-MSE (Lesort et al., 2017) derives a quantitative metric from this information.

For a given observation $o$, we find the nearest neighbours of its associated state $s$ in the learned state space, and project them in the ground truth state space. Then we compute the average distance to its neighbours in the latter space:

$$\text{KNN-MSE}(s) = \frac{1}{k} \sum_{s' \in KNN(s,k)} ||\tilde{s} - \tilde{s}'||^2 \tag{2}$$

where $\text{KNN}(s, k)$ returns the $k$ nearest neighbours of $s$ (chosen with the Euclidean distance) in the learned state space $\mathcal{S}$, $\tilde{s}$ is the ground truth state associated to $s$, and $\tilde{s}'$ is the one associated to $s'$. A low KNN-MSE means that a neighbour in the ground truth is still a neighbour in the learned representation, and thus, local coherence is preserved.

### 4.2.2 Correlation

A Pearson $\rho$ correlation coefficient's matrix is computed for each dimension pair $(s, \tilde{s})$, where $\tilde{s}$ is the ground truth (GT) state, $s$ the learned state, and $\mu_s$ and $\sigma_s$ are the mean and standard deviation, respectively, of state $s$:

$$\rho_{s,\tilde{s}} = \frac{\mathbb{E}[(s - \mu_s) * (\tilde{s} - \mu_{\tilde{s}})]}{\sigma_s * \sigma_{\tilde{s}}} \tag{3}$$

We can visualize the correlation matrix to quantitatively assess the ability of a model to encode relevant information in the states learned. For instance, the correlation matrix in Figure 3 shows degrees of correlation between the mobile robot position and the learned states. The plot illustrates that for each dimension $i$ of the predicted states $s$, there is a correlation close to 1 (in absolute value) with at least one dimension $j$ of the agent's real position $\tilde{s}_j$. Therefore, this gives measurable evidences that the model was able to encode the position of the mobile robot.
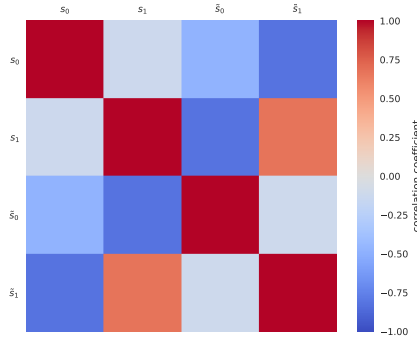
Figure 3: Correlation matrix for mobile robot navigation dataset (static target), between each dimension $s_i$ of predicted states $s$ and the ground truth $\tilde{s}_j$. We consider the ground truth to be the agent's real position. The states (dimension=2) are learned by combining a forward and an inverse model.

This visualization tool is quite useful for low-dimensional spaces. However, for state spaces with a high number of dimensions, looking at the correlation matrix becomes impractical. Therefore, we introduce the following metric, named $GTC$ for Ground Truth Correlation, that allows to compare the model's ability to encode relevant information:

$$GTC_{(i)} = \max_j |\rho_{s,\tilde{s}}(i, j)| \in [0, 1] \tag{4}$$

with $i \in [\![0, |\tilde{s}|]\!]$, $j \in [\![0, |s|]\!]$, $\tilde{s} = [\tilde{s}_1; ...; \tilde{s}_n]$, and $\tilde{s}_k$ being the $k^{th}$ dimension of the ground truth state vector.

For instance, in the Mobile Robot environment with random target, the ground truth state is composed of the 2D robot position and 2D target position. That is to say, the ground truth states have a dimension of 4: $|\tilde{s}| = 4$.

The vector $GTC$ gives for each component $i$ of the ground truth states $\tilde{s}$, the maximum absolute correlation value between $\tilde{s}_i$ and any component of the predicted states $s$. Therefore, $GTC$ measures the similarity per component, between the learned states $s$ and the ground truth states $\tilde{s}$.

We also introduce a metric, which is the mean of $GTC$, that allows to compare learned states using one scalar value:

$$GTC_{mean} = \mathbb{E}[GTC] \tag{5}$$

## 4.3 Quantitative Evaluation With Reinforcement Learning

Comparing the performance of RL algorithms, using the learned state representations, is the most relevant approach to evaluate the SRL methods. To do so, our framework integrates 8 algorithms (A2C, ACKTR, ACER, DQN, DDPG, PPO1, PPO2, TRPO) from Stable-Baselines (Hill et al., 2018) (a fork of OpenAI baselines (Dhariwal et al., 2017)), Augmented Random Search (ARS) (Mania et al., 2018), Covariance Matrix Adaptation Evolutionary Strategy (CMA-ES) (Hansen et al., 2003) and Soft Actor Critic (SAC) (Haarnoja et al., 2018).

## 5. Experiments

We perform experiments on the proposed datasets using states learned with the approaches described in Section 2.2 along with ground truth (GT). Here, we report results obtained with PPO.

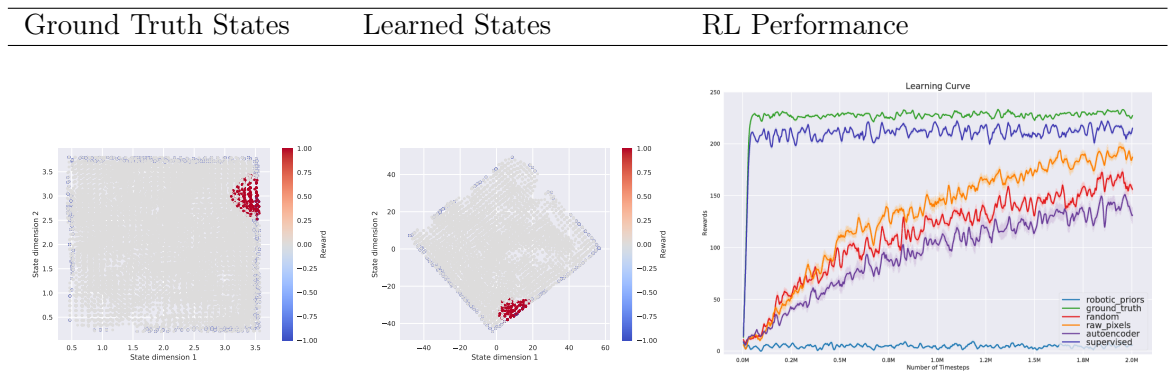| Ground Truth States | Learned States | RL Performance |
| --- | --- | --- |



Table 1: Ground truth states (left), states learned (Inverse and Forward) (centre), and RL performance evaluation (PPO) (right) in the mobile robot environment. Colour denotes the reward, red for positive, blue for negative and grey for null reward (left and center). The full resolution plot of the RL performance evaluation can be found in appendix Fig. 4 (right)

Table 1 illustrate the qualitative evaluation of a state space learned by combining forward and inverse models on the mobile robot environment. It also shows the performance of PPO based on the states learned by several approaches.

Table 2 shows the KNN-MSE for the different SRL approaches on the implemented environments.

| Dataset | Mobile-robot (1) | | Robotic-arm (3) | | Robotic-arm-real (4) |
| Static/Random Target | Static | Random | Static | Random | Static |
|---|---|---|---|---|---|
| *Ground Truth* | 0.0099 | 0.0164 | 0.0025 | 0.0025 | 0.0105 |
| *AE* | 0.0168 | 0.7213 | 0.00336 | 0.0027 | 0.0179 |
| *VAE* | 0.0161 | 0.1295 | 0.0032 | 0.0027 | 0.0177 |
| *Robotic Priors* | 0.0200 | 0.0900 | 0.0029 | 0.0027 | 0.0213 |
| *Forward* | 0.5111 | 1.1557 | 0.1425 | 0.2564 | 0.0796 |
| *Inverse* | 0.0191 | 0.7703 | 0.0182 | 0.2705 | 0.0521 |
| *Fwd+Inv* | 0.0164 | 0.7467 | 0.0176 | 0.2705 | 0.0368 |

Table 2: KNN-MSE results for each SRL method. Ground Truth (robot absolute or relative position with respect to target object), auto-encoders, forward (fwd), inverse (inv) models and combinations.

The table 3 gives the $GTC_{mean}$ metrics for several approaches and the associated RL performance using PPO. It shows that $GTC_{mean}$ is a good indicator for the performance than can be obtained in RL: a good disentanglement with a good correlation with the ground truth state (i.e. a high $GTC_{mean}$) lead to a higher mean reward in RL.

During the development of *S-RL Toolbox*, we learned some useful insights on SRL. We observed that auto-encoders do not reconstruct objects if they are too small. This is an issue if the object is relevant for the task. An inverse model is usually sufficient to learn a coherent state representation (in our case, the position of the controllable object). Continuity in the state space is important to perform well in RL. Using real-time SRL tools, we note discontinuities in the state space learned by AE, even if reconstruction error is low, and that hinders RL.

| GroundTruthCorrelation | $x_{robot}$ | $y_{robot}$ | $x_{target}$ | $y_{target}$ | *Mean* | *RL* |
|---|---|---|---|---|---|---|
| Robotic Priors | 0.2 | 0.2 | 0.41 | 0.66 | 0.37 | 5.4 ± 3.1 |
| Random | 0.68 | 0.65 | 0.34 | 0.31 | 0.50 | 163.4 ± 10.0 |
| Supervised | 0.69 | 0.73 | 0.70 | 0.72 | 0.71 | 213.3 ± 6.0 |
| Auto-Encoder | 0.52 | 0.51 | 0.24 | 0.23 | 0.38 | 138.5 ± 12.3 |
| GT | 1 | 1 | 1 | 1 | 1 | 229.7 +- 2.7 |

Table 3: $GTC$ and $GTC_{mean}$, and mean reward performance in RL (using PPO) per episode after 2 millions steps, with standard error for each SRL method in mobile robot with random target environment.

## 6. Related Work

In the RL literature, several classic benchmarks are used to compare algorithms. For discrete actions, Atari Games from OpenAI Gym suite (Brockman et al., 2016) are often adopted, whereas for continuous actions, MuJoCo locomotion tasks (Todorov et al., 2012) is favoured. OpenAI recently open-sourced robotics goal-based tasks (Plappert et al., 2018). These environments are close to what we propose; however, they use a non-free physics engine, observations are not images and customization is not easy. Also, our tools were designed with SRL in mind and offer a gradual difficulty.

Regarding the SRL literature, very diverse environments are used, without common metrics or visualizations. The SRL methods are usually only compared with learning from raw observations. Jonschkowski and Brock (2015) assess the quality of learned representations in a slot car scenario (similar to Lange et al. (2012)) and a mobile navigation task (following Sprague (2009); Boots et al. (2011)). More recently, Ha and Schmidhuber (2018)

present their results on Vizdoom and RaceCar environments. They show interesting insights of what was learned by navigating in the latent space and projecting states back to the pixel space (similar to Fig. 2). Zhang et al. (2018) use a subset of MuJoCo tasks, with joints as input, and a binary maze environment. They perform an insightful ablation study of their SRL model.

Our contribution is two-fold: we provide a framework integrated with RL algorithms, environments and tools for SRL, along with implementations of the main SRL methods.

## 7. Discussion and Conclusions

This paper presents a set of environments on which to perform SRL benchmarks of incremental difficulty to solve tasks in RL, specifically, in robotics control. Our proposed toolbox facilitates fast iterations, interpretability and reproducibility, with a set of qualitative and quantitative metrics and interactive visualization tools. We believe such a framework is needed to have fair comparisons, focused on robotics control, among SRL methods.

## Acknowledgments

---

## Appendix A. Implementation Details

### A.1 Datasets Details

In this section, we provide the parameters used to generate datasets, for the results presented in Table 2. The simulated datasets were created with a random policy, using *PyBullet* (Coumans et al., 2018) (generating up to 30k samples per minute on a 8-core machine [4]). The real Baxter dataset was recorded using ROS.

We used 10k samples of each dataset to learn a state representation. Reward is sparse (see Section 3 for details) and actions are discrete (encoded as integers) for all datasets [5].

Each target object can be fixed or randomly positioned between episodes; what we call "absolute" or "relative" dataset, refers to the fact of learning a relative or absolute position of the robot with respect to the target object. The images dimension in every dataset is 224x224 pixels.

Table 4: Dataset details. Distractors can both be static or moving

| Dataset | Reward | Actions | Distractors |
|---|---|---|---|
| Mobile-robot | Sparse | 4, Discrete ($\Delta$(X,Y) pos.) | No |
| Robotic-arm | Sparse | 5, Discrete ($\Delta$(X,Y,Z) pos.) | No |
| Robotic-arm-real | Sparse | 5, Discrete ($\Delta$(X,Y,Z) pos.) | Yes |

### A.2 Baselines Results

In this section, we provide baselines results for each environment.

| **Budget** (in timesteps) | 1 Million | 2 Million |
|---|---|---|
| Ground Truth | $198.0 \pm 16.1$ | $211.6 \pm 14.0$ |
| Raw Pixels | $177.9 \pm 15.6$ | $215.7 \pm 9.6$ |
| Auto-Encoder | $159.8 \pm 16.1$ | $188.8 \pm 13.5$ |

Table 5: Mean reward performance in RL (using PPO) per episode (average on 100 episodes) for different budgets, with standard error in Navigation 1D target environment.

| **Budget** (in timesteps) | 1 Million | 2 Million | 3 Million | 5 Million |
|---|---|---|---|---|
| Ground Truth | $227.8 \pm 2.8$ | $229.7 \pm 2.7$ | $231.5 \pm 1.9$ | $234.4 \pm 1.3$ |
| Raw Pixels | $136.3 \pm 11.5$ | $188.2 \pm 9.4$ | $214.0 \pm 5.9$ | $231.5 \pm 3.1$ |
| Auto-Encoder | $97.0 \pm 12.3$ | $138.5 \pm 12.3$ | $167.7 \pm 11.1$ | $192.6 \pm 8.9$ |

Table 6: Mean reward performance in RL (using PPO) per episode (average on 100 episodes) for different budgets, with standard error in Navigation 2D random target environment.

---

4. CPU: Intel Core i7-7700K GPU:Nvidia GeForce GTX 1080 Ti
5. Datasets can also be generated with continuous actions and dense (shaped) reward

| **Budget** (in timesteps) | *1 Million* | *2 Million* | *3 Million* | *5 Million* |
|---|---|---|---|---|
| Ground Truth | 4.1 ± 0.5 | 4.1 ± 0.6 | 4.1 ± 0.6 | 4.2 ± 0.5 |
| Raw Pixels | 0.6 ± 0.3 | 0.8 ± 0.3 | 1.2 ± 0.3 | 2.6 ± 0.3 |
| Auto-Encoder | 0.92 ± 0.3 | 1.6 ± 0.3 | 2.2 ± 0.3 | 3.4 ± 0.3 |

Table 7: Mean reward performance in RL (using PPO) per episode (average on 100 episodes) for different budgets, with standard error in robotic arm with random target environment.

| **Budget** (in timesteps) | *1 Million* | *2 Million* | *3 Million* | *5 Million* |
|---|---|---|---|---|
| Ground Truth | 4.3 ± 0.3 | 4.4 ± 0.2 | 4.4 ± 0.2 | 4.6 ± 0.2 |
| Raw Pixels | 0.8 ± 0.3 | 1.0 ± 0.3 | 1.2 ± 0.3 | 2.0 ± 0.3 |
| Auto-Encoder | 1.17 ± 0.3 | 1.5 ± 0.3 | 1.9 ± 0.4 | 3.0 ± 0.4 |

Table 8: Mean reward performance in RL (using PPO) per episode (average on 100 episodes) for different budgets, with standard error in robotic arm with moving target environment.

### A.3 Reinforcement Learning Experiments

#### A.3.1 Reinforcement Learning

We use the implementations from Stable-Baselines (Hill et al., 2018) for the RL experiments (except for CMA-ES, ARS and SAC which we implemented ourselves). Where we used the default hyperparameters set by OpenAI Baselines. With 10 random seeds in order to have quantitative results.

As for the network of the policies, the same architecture is used in all different methods. For the SRL and ground truth approaches, it is a 2-layers MLP, whereas for learning from raw pixels, it is the CNN from (Mnih et al., 2015) implemented in OpenAI baselines.

All CNN policies normalize the input image by dividing it by 255. Observations are not stacked. When learning from SRL, the states are normalized using a running mean/std average. Reinforcement learning metrics reported are the average returned rewards over 5 policies, independently trained using the same RL algorithm with a different seed.

#### A.3.2 State Learning Representation

For the SRL methods, we used an altered form of ResNet (Table 9), which is more compact for our needs. However, for forward and inverse models, we used a linear model in addition to the CNN for learning a state representation. We are using the Adam (Kingma and Ba, 2014) optimizer in all our models, with learning rates that vary from $10^{-2}$ to $10^{-3}$. The batch size of forward and inverse is 128, 256 for robotic priors, and 32 for the other models.

Table 9: CNN architecture for SRL methods. The first layers are inspired by ResNet architecture (He et al., 2015).

| Layer | Architecture |
| --- | --- |
| 1 | Conv2d(3, 64, kernel_size=7, stride=2, padding=3, bias=False) + BN + ReLU |
| 2 | MaxPool2d(kernel_size=3, stride=2, padding=1) |
| 3 | Conv2d(64, 64, kernel_size=3, stride=1, padding=1, bias=False) + BN + ReLU |
| 4 | MaxPool2d(kernel_size=3, stride=2) |
| 5 | Conv2d(64, 64, kernel_size=3, stride=2, padding=1, bias=False) + BN + ReLU |
| 6 | MaxPool2d(kernel_size=3, stride=2) |
| 7 | Linear(6 * 6 * 64, state_dim) |

#### A.3.3 Results

Fig. 4 shows the average reward during learning of the PPO algorithm.

We observed during the development of the *S-RL Toolbox*, that PPO was one of the best RL algorithm for SRL benchmarking. It allows us to obtain good performance and was consistent without needing to change any hyperparameters. Figure 5 illustrates this, in addition to the variability of the RL algorithms.

## Appendix B. Supplementary material

The visualization and interactive state space exploration tools are demonstrated in the following videos:

- S-RL Toolbox Showcase: `https://youtu.be/qNsHMkIsqJc`

- S-RL Toolbox Environments: `https://tinyurl.com/y973vhfy`
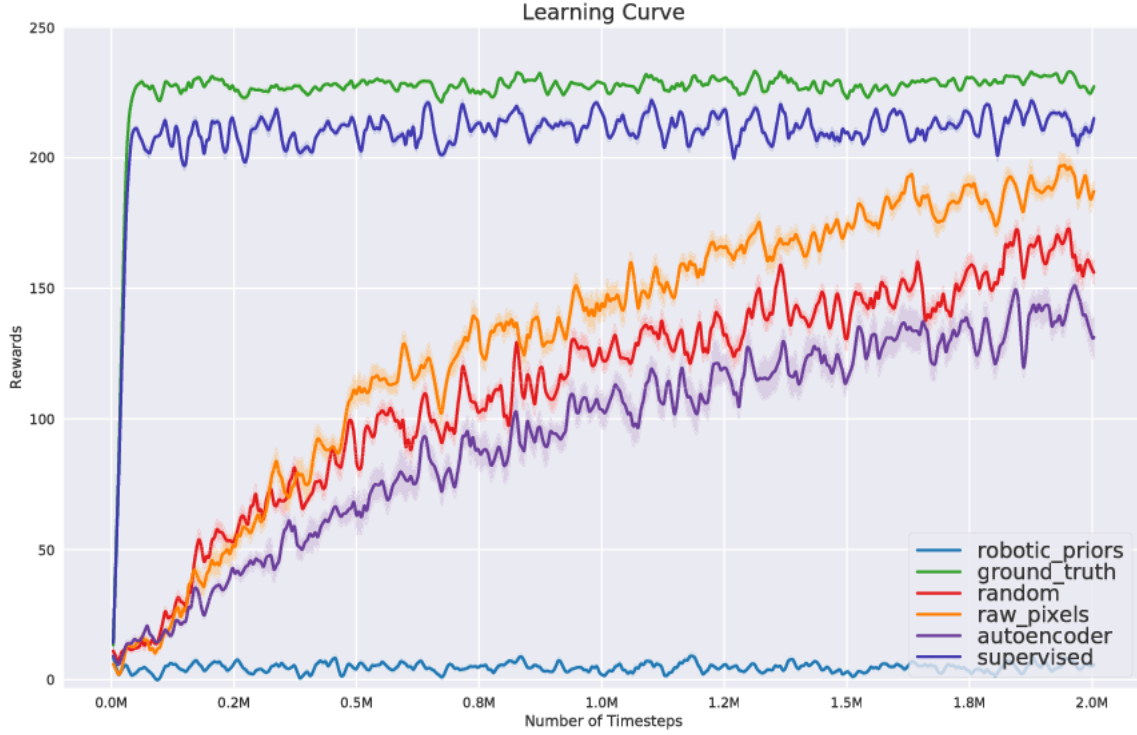
Figure 4: Performance (mean and standard error for 10 runs) for PPO algorithm for different state representations learned in mobile-robot-navigation (random target) environment
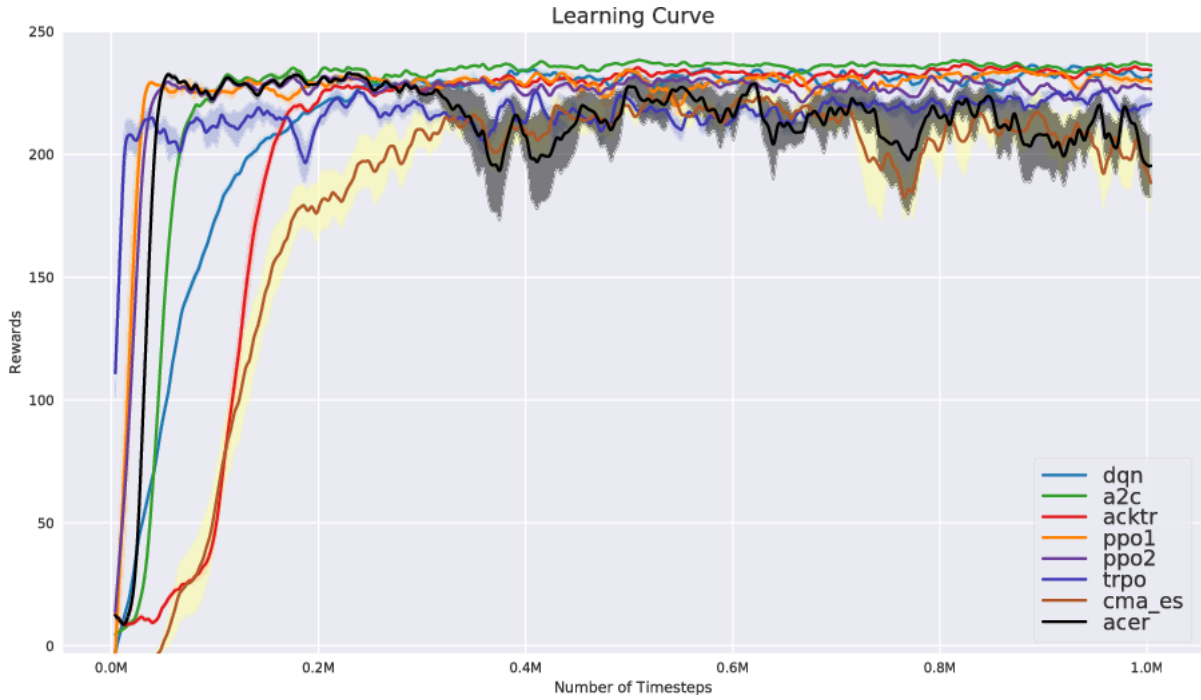


Figure 5: Performance (mean and standard error) on RL algorithms using ground truth states with mobile robot (random target) environment

- Kuka robot arm: RL running PPO (SRL trained with VAE): `https://tinyurl.com/yarpbs2c`

- Kuka robot arm environment: State Representation Learning Benchmark running PPO on ground truth states: `https://tinyurl.com/yd54l7o`

## References

Pierre Baldi. Autoencoders, unsupervised learning, and deep architectures. In *Proceedings of ICML Workshop on Unsupervised and Transfer Learning*, pages 37–49, 2012.

Byron Boots, Sajid M Siddiqi, and Geoffrey J Gordon. Closing the learning-planning loop with predictive state representations. *The International Journal of Robotics Research*, 30(7):954–966, 2011.

Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *CoRR*, abs/1606.01540, 2016. URL `http://arxiv.org/abs/1606.01540`.

E. Coumans, Y. Bai, and J. Hsu. Pybullet physics engine. `http://pybullet.org/`, 2018.

Prafulla Dhariwal, Christopher Hesse, Oleg Klimov, Alex Nichol, Matthias Plappert, Alec Radford, John Schulman, Szymon Sidor, and Yuhuai Wu. Openai baselines. `https://github.com/openai/baselines`, 2017.

D. Ha and J. Schmidhuber. World Models. *ArXiv e-prints*, March 2018.

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *arXiv preprint arXiv:1801.01290*, 2018.

Nikolaus Hansen, Sibylle D Müller, and Petros Koumoutsakos. Reducing the time complexity of the derandomized evolution strategy with covariance matrix adaptation (cma-es). *Evolutionary computation*, 11(1):1–18, 2003.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015. URL `http://arxiv.org/abs/1512.03385`.

Ashley Hill, Antonin Raffin, Rene Traore, Prafulla Dhariwal, Christopher Hesse, Oleg Klimov, Alex Nichol, Matthias Plappert, Alec Radford, John Schulman, Szymon Sidor, and Yuhuai Wu. Stable baselines. `https://github.com/hill-a/stable-baselines`, 2018.

Rico Jonschkowski. Learning robotic perception through prior knowledge. 2018.

Rico Jonschkowski and Oliver Brock. State representation learning in robotics: Using prior knowledge about physical interaction. In *Proceedings of Robotics: Science and Systems*, July 2014.

Rico Jonschkowski and Oliver Brock. Learning state representations with robotic priors. *Autonomous Robots*, 39(3):407–428, 2015. ISSN 0929-5593.

Rico Jonschkowski, Roland Hafner, Jonathan Scholz, and Martin A. Riedmiller. PVEs: Position-Velocity Encoders for Unsupervised Learning of Structured State Representations. *CoRR*, abs/1705.09805, 2017. URL `http://arxiv.org/abs/1705.09805`.

D. P Kingma and M. Welling. Auto-Encoding Variational Bayes. *ArXiv e-prints*, December 2013.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014. URL `http://arxiv.org/abs/1412.6980`.

Sascha Lange, Martin Riedmiller, and Arne Voigtlander. Autonomous reinforcement learning on raw visual input data in a real world application. In *Neural Networks (IJCNN), The 2012 International Joint Conference on*, pages 1–8. IEEE, 2012.

Timothée Lesort, Mathieu Seurin, Xinrui Li, Natalia Díaz-Rodríguez, and David Filliat. Unsupervised state representation learning with robotic priors: a robustness benchmark. *CoRR*, abs/1709.05185, 2017. URL `http://arxiv.org/abs/1709.05185`.

Timothée Lesort, Natalia Díaz-Rodríguez, Jean-François Goudou, and David Filliat. State representation learning for control: An overview. *Neural Networks*, 2018. ISSN 0893-6080. doi: https://doi.org/10.1016/j.neunet.2018.07.006. URL `http://www.sciencedirect.com/science/article/pii/S0893608018302053`.

Horia Mania, Aurelia Guy, and Benjamin Recht. Simple random search provides a competitive approach to reinforcement learning. *arXiv preprint arXiv:1803.07055*, 2018.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

Deepak Pathak, Pulkit Agrawal, Alexei A. Efros, and Trevor Darrell. Curiosity-driven exploration by self-supervised prediction. In *ICML*, 2017.

Matthias Plappert, Marcin Andrychowicz, Alex Ray, Bob McGrew, Bowen Baker, Glenn Powell, Jonas Schneider, Josh Tobin, Maciek Chociej, Peter Welinder, Vikash Kumar, and Wojciech Zaremba. Multi-goal reinforcement learning: Challenging robotics environments and request for research. *CoRR*, abs/1802.09464, 2018. URL `http://arxiv.org/abs/1802.09464`.

Pierre Sermanet, Corey Lynch, Jasmine Hsu, and Sergey Levine. Time-contrastive networks: Self-supervised learning from multi-view observation. *CoRR*, abs/1704.06888, 2017. URL `http://arxiv.org/abs/1704.06888`.

Evan Shelhamer, Parsa Mahmoudieh, Max Argus, and Trevor Darrell. Loss is its own reward: Self-supervision for reinforcement learning. *arXiv preprint arXiv:1612.07307*, 2017.

Nathan Sprague. Predictive projections. In *IJCAI*, pages 1223–1229, 2009.

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.

Manuel Watter, Jost Springenberg, Joschka Boedecker, and Martin Riedmiller. Embed to control: A locally linear latent dynamics model for control from raw images. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems 28*, pages 2746–2754. Curran Associates, Inc., 2015.

Amy Zhang, Harsh Satija, and Joelle Pineau. Decoupling dynamics and reward for transfer learning. *arXiv preprint arXiv:1804.10689*, 2018.