VeriDream Deliverable 6

Guidelines for using DREAM methods

This project has received funding from the
European Union's Horizon 2020 research and innovation
programme under Grant Agreement No. 951992

Project	
Grant number	951992
Acronym	VeriDream
Deliverable	
Number	6
Rel. Number	2.1
Workpackage	2
Nature	R
Dissemination	PU
Due date	31.05.2021
Lead	Sorbonne
Contributors	Sorbonne, DLR, ARMINES

Document History

Version	Date	Partner	Description
0.1	19.11.2020	DLR	Creation of method template document
0.2	25.03.2021	Sorbonne	First draft of method template document
0.3	26.04.2021	Sorbonne	Quality Diversity section
0.4	28.04.2021	ARMINES	Continual Learning section
0.5	04.05.2021	DLR	Pose Estimation section
0.6	04.05.2021	ARMINES, DLR	State Representation Learning section
0.7	05.05.2021	Sorbonnes, DLR	Review
0.8	05.05.2021	GoodAI	Review
1.0	12.05.2021	DLR	Final version

1 Introduction

The aim of this deliverable is to provide an overview of the machine learning algorithms used in the VeriDream project. In particular, we highlight the metaparameters and limitations of the method, which is very important when applying the algorithms to real-world challenges. To which industrial use cases the algorithms are applied is described in D1.1 "Use Case Requirements and Matchmaking Results". This deliverable essentially describes the algorithm *as is*, before their application to these industrial use cases. How the algorithms are adapted during the project to generalize and robustify them will be documented in D2.2 "Closing the innovation loop", and D2.3/D2.4 on "Generalization and Robustification".

VeriDream's main focus is on the application of machine learning algorithms to robotics. Programming robots in an uncontrolled environment that can further change along time is a challenge that learning algorithms can help to deal with. Although machine learning has made significant progresses over the last years, with noticeable successes in many different fields from autonomous translation to image search, recommendations or games, it remains hard to apply it to robotics. Contrary to most other AI systems, a robot is connected to the real world through sensors and actuators. It can thus both perceive and act without any mediation from a human. This autonomous loop of interaction with the environment creates constraints that are specific to robotics.

Robot perceptions have not been selected or annotated by humans, they arrive at real time and need to be taken into account, no matter if they contain occlusions or noise. They are also frequently in large dimensions (video images) and thus cannot be used to decide what to do without extracting relevant information from them. Knowing the state of the system, i.e. the information necessary to make a decision, from sensors is thus a challenge per se. Section 2 presents a method designed to extract the pose of an object from raw perception, which is an important information for any task involving object recognition or manipulation. Section 3 presents approaches that are more general to extract this state from robot experience.

Robots can act and these actions influence future perceptions. Actions need to be carefully designed to allow the robot to reach its goal while not spending too much time exploring the space of possibilities. Diversity and quality-diversity algorithms, introduced in Section 4, can be used to this end, as well as for exploring the effects that the robot can generate in a trial and error loop.

Finally, some methods are dedicated to a changing environment. They are presented in the continual learning section (Section 5).

After each section, we list selected publications on these topics that were made by the VeriDream participants during the two preceding H2020 projects on which VeriDream builds, i.e. DREAM[1] and RobDREAM[2]. Publications made *during* VeriDream are listed on its website[3], and the project progress reports. External publications are listed in separate bibliographies.

Contents

[1]DREAM – "Deferred Restructuring of Experience in Autonomous Machines" Grant ID: 640891.
[2]RobDREAM – "Optimising Robot Performance while Dreaming". Grant ID: 645403.
[3]VeriDREAM website: `https://veridream.eu/publications/`

2 Pose Estimation: Augmented Autoencoder

Researchers involved: Maximilian Durner (DLR)

2.1 Description

The main idea is to learn an implicit representation of an object's orientation, instead of using explicit descriptions such as rotation matrices or Euler angles. The key insight is that such an implicit representation can be obtained by training an adapted autoencoder, where the training data consists of synthetically rendered object views, i.e. it does not require any human-labelled training data. During the inference phase, a detected object view is passed to a learned encoder, and the resulting latent code is used to retrieve a set of closest orientations from a previously created codebook.

2.2 What problem does it solve?

Robots manipulating objects in dynamically changing environments require the knowledge of the object's location. Depending on the manipulation task as well as the robotic gripper, the pure location is not enough information and a precise 6D pose consisting of the 3D translation and 3D orientation is required. The presented method is able to estimate such 6D pose in a 2-step manner.

Besides the manipulation of objects the information of the object's pose can further be used in augmented reality use-cases.

2.2.1 How does the method work?

The Augmented Autoencoder (AAE) pipeline estimates a 6D pose of an object purely trained on synthetic data (Figure 1). In order to estimate the pose of an object a bounding box detector first has to detect and classify the object. The resulting region of interest (ROI) is then forwarded to the AAE, which predicts the 3D orientation of the corresponding object. To do so, the AAE generates a latent representation of the object appearance, which can be translated to a 3D orientation via a codebook. Based on this rotation information the translation can be computed via the projective distance. The resulting pose can be either directly used or further refined by a Iterative Closest Point (ICP) approach. Please note, while the AAE only requires a single RGB image the ICP approach runs on the corresponding depth image.

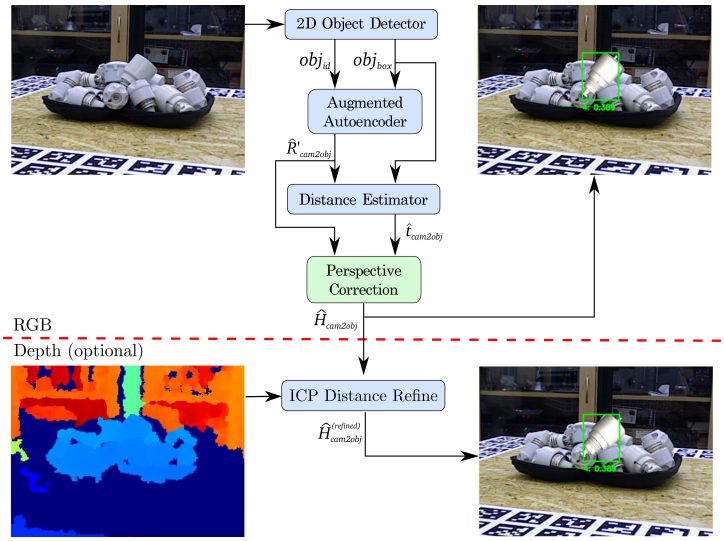

Figure 1: Illustration of the AAE pose estimation pipeline

2.2.2 What meta-parameters does the method have?

- The optimization hyper-parameters (learning rate, optimizer, batch size, . . .)
- The training budget (training epochs . . .)
- The augmentations during training
- The minimum number of equidistant views rendered from a view-sphere views to generate the code-book
- The applied background images

2.2.3 What are the limitations of the method?

While the 3D orientation predicted by the AAE achieves sufficient accuracy, the 3D translation, especially in the z-direction, can possibly limit the performance. As described above the 3D translation is computed via the projective distance based on the bounding box estimation. Thus, the quality of the bounding box highly affects the 3D translation estimation. The resulting pose can be used as the initial pose for a pose refinement like an ICP approach. The ICP requires a high sensor to noise ratio in order to work properly. This however, cannot always be assured in real world applications.

2.2.4 What inputs does the method expect, and what output does it return?

The input of the AAE pipeline is a single RGB image. After the bounding box detection the ROI is reshaped to a 128x128x3 image. For the optional refinement step an additional depth image of the same scene is required.

The output of the pipeline is the object class and a 6D pose consisting of the 3D translation as well as the 3D rotation of the corresponding object.

2.2.5 References

- [Sundermeyer et al., 2018] presents the initial idea of the AAE pose estimation approach.
- [Sundermeyer et al., 2020] presents an advanced AAE version which is able to generate a generalized object representation.
- Source code is available at `https://github.com/DLR-RM/AugmentedAutoencoder`

Publications by DREAM/RobDREAM partners

[Sundermeyer et al., 2020] Sundermeyer, M., Durner, M., Puang, E. Y., Marton, Z.-C., Vaskevicius, N., Arras, K. O., and Triebel, R. (2020). Multi-Path Learning for Object Pose Estimation Across Domains. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 13913–13922, Seattle, WA, USA. IEEE.

[Sundermeyer et al., 2018] Sundermeyer, M., Marton, Z.-C., Durner, M., Brucker, M., and Triebel, R. (2018). Implicit 3D Orientation Learning for 6D Object Detection from RGB Images. In *Computer Vision – ECCV 2018*, volume 11210, pages 712–729. Springer International Publishing, Cham. Series Title: Lecture Notes in Computer Science.

3 State Representation Learning

Researchers involved: Antonin Raffin (DLR), David Filliat (Armines), Natalia Diaz (Armines)

3.1 Description

Reinforcement Learning (RL) is an approach that makes it possible to learn controllers for systems with very few priors, only by interacting with them and by optimizing a reward that defines the system objective. Many reinforcement learning algorithms exist and thanks to deep learning, performances of the recent Deep Reinforcement Learning algorithms now makes them closer to concrete industrial applications. They also make it possible to learn controllers using as input low-level unprocessed data such as camera images.

A key element to efficient control in general is the state representation of the system. An adequate state representation usually leads to a good policy. However, extracting useful features from data by hand may be tedious and requires domain knowledge. Sometimes, even with expert insights, the state representation is hard to design (e.g. when the number of sensors is limited, or when physical modifications of the system are impossible). A solution consists in learning the controller in an end-to-end manner (e.g. from pixels to torques) but this makes it hard to interpret in case of failure, as the state is distributed throughout the entire network.

State representation learning (SRL) tackles both issues of automatically extracting features while remaining explainable. The goal is to learn, by extracting features from raw data, a low dimensional state that contains sufficient information to solve a task. A controller policy then sits on top of the SRL layer, relying on a smaller number of input features than an end-to-end system would, bringing down both the training time and the requirements on the amount of training data. The approach is illustrated in figure 2. The learning is done in an unsupervised way and can combine raw sensor data along with information about the actuator (e.g. joints position).

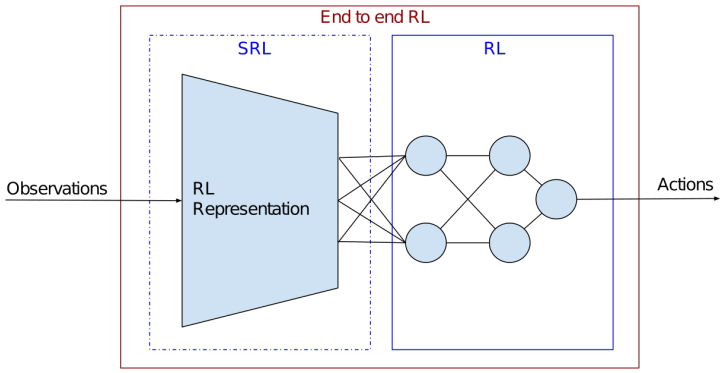

Figure 2: Illustration of the State Representation Learning approach in combination with reinforcement learning.

We developed a toolbox (SRL Toolbox) with implementation of many of the reference SRL methods. State representation learning can be applied with many control methods, e.g. simple PID controllers, Model Predictive Control, or reinforcement learning, but we applied it mostly with reinforcement learning and developed a library (Stable Baselines) dedicated to provide reference implementations of many state of the art algorithms.

3.2 What problem does it solve?

Reinforcement learning is a solution to learn controllers for systems that are difficult to model, or for systems where we don't want to make a model by hand. RL is also useful when defining the task goal is easier than designing a controller (e.g. knowing if an object is grasped is much easier than designing a grasp strategy).

State Representation Learning is first a way to accelerate Reinforcement Learning and make it more robust. Compared to end-to-end learning, SRL explicitly separates feature extraction from the control (see the figure 2). This allows to explore directly the state that was learned as a basis for control. This is particularly useful when trying to improve the method because failure cases can be interpreted. Last but not least, because those

features encode information about the environment of the robot, they can be re-used by different methods across different tasks, which is usually not possible when learning in an end-to-end manner.

State Representation Learning can be used to produce a relevant representation of a system when the relation between sensor data and state is complex. For example it can be used to control an existing system where the sensors are not sufficient to provide a proper state estimate, but we don't want to physically modify the system. Using state representation learning on the existing sensor data, possibly taking into account the sensor history, is a way to improve or implement a new controller without having to add new sensors.

Finally, State Representation Learning can also be used when we don't want an engineer to intervene when the system is defined or modified. It can be a way to automatically update state representation when a system is modified by a user for example.

3.3 State Representation Learning - SRL Toolbox

3.3.1 How does the method work?

State representation learning aims at learning compact representations from raw observations in robotics and control applications. Approaches used for this objective fall in five main categories:

- auto-encoders: the state representation is trained to contain enough information to reconstruct the original observation.
- learning forward models: the state representation is trained by predicting the next state representation, given a performed action.
- inverse dynamics: the state representation is trained by predicting the performed action given the states before and after the action.
- learning using generic priors on the state characteristics: the state representation is trained using general knowledge (e.g., temporal continuity, proportionality to a given signal, ...).
- contrastive learning: the state representation is trained by making the projections of two augmented views of the same observation close to each other.

These objectives can be used independently, but often they are combined. As an example, an autoencoder objective can be used, and to integrate information about the control in the features, it can be combined with a forward kinematics objective. This allows to learn features about things that are controllable.

However, the diversity in applications and methods makes the field lack standard evaluation datasets, metrics and tasks. The SRL toolbox therefore provides a set of environments, data generators, robotic control tasks, metrics and tools to facilitate iterative state representation learning, the evaluation of the learned representations in reinforcement learning settings and the visual analysis of their behavior.

In the project, the implementations of the SRL methods, along with the metrics and analysis tools can be reused for new tasks by writing an interface.

3.3.2 What meta-parameters does the method have?

The main meta-parameters are:

- The choice of the objectives for a specific task: often the auto-encoder and forward or inverse dynamics are a good baseline for applying SRL to a new problem. These objectives can be complemented with task-specific priors that need to be defined according to the task.
- The choice of the input for the state representation. The SRL approach was mainly used with RGB images as input, but it can use other configurations: RGBD images, multiple RGB images, joints configurations... For non observable problems (with a single observation), a recurrent network or a network taking a time window as input can be used.
- The choice of the state representation dimension. Experiments show that a higher dimension than the theoretical minimum often leads to better results, and splitting the state in several sub-parts can also help [Raffin et al., 2019].

3.3.3 What are limitations of the method?

The main limitations are:

- the need of many interactions to learn a state representation, which makes it difficult to apply directly on a real robot. It is preferable to apply it on a simulator, which is possible either if the task is in simulation, or if it is possible to simulate precisely enough the real system.
- the evaluation of the quality of the state space is difficult. The most objective is the performance of the control system based on the learned state, but this is often very time consuming. Various analysis tools of the SRL toolbox can be used to visualize the behavior of the state representation in order to qualitatively estimate its quality.
- SRL objectives are surrogate objectives which means that the state representation does not necessarily encode enough information to solve the task.
- the state representation is only valid for the training distribution. If the test distribution is too far away from it, retraining is needed otherwise the state representation fails to encode useful information.

3.3.4 What inputs does the method expect, and what output does it return?

State Representation Learning needs a system to control, that is defined by the observation that can be made on the system (e.g., RGB or RGBD images, joint positions,...) and the actions the system can perform (e.g., motor commands). It can also use a task definition for the system that is defined as a reward function. For this system a sequence of (observation; action; observation) should be gathered to form a training dataset.

As an output, SRL will provide a neural network that takes observations as an input and outputs a state representation of the system. This network can be reused with reinforcement learning or any other control method.

3.3.5 References

- [Lesort et al., 2018] presents a review of the state of the art in State Representation Learning.
- [Raffin et al., 2018] presents the SRL-Toolbox and baseline results.
- [Raffin et al., 2019] and [Lesort et al., 2019] present applications of SRL on robotics setups.
- Source code is available at `https://github.com/araffin/robotics-rl-srl`

3.4 Reinforcement Learning - Stable Baselines

3.4.1 How does the method work?

Stable Baselines3 provides open-source implementations of deep reinforcement learning (RL) algorithms in Python. The implementations have been benchmarked against reference codebases, and automated unit tests cover 95% of the code. The algorithms follow a consistent interface and are accompanied by extensive documentation, making it simple to train and compare different RL algorithms on a given problem once the interface is written.

3.4.2 What meta-parameters does the method have?

The main meta parameters of the method are:

- The algorithm [4] (e.g. sample efficiency vs wall clock time)
- The policy network structure
- The optimization hyper-parameters (learning rate, optimizer, batch size, ...)

[4] `https://stable-baselines3.readthedocs.io/en/master/guide/rl_tips.html`

- The training budget (how many interactions with the environment is allowed for training)
- The number of parallel workers
- The exploration noise magnitude

3.4.3 What are limitations of the method?

The main limitation of Reinforcement Learning is the need for many interactions to learn a controller, which makes it difficult to apply directly on a real robot. It is preferable to apply it on a simulator, which is possible either if the task is in simulation, or if it is possible to simulate precisely enough the real system.

3.4.4 What inputs does the method expect, and what output does it return?

Reinforcement learning needs a system to control, that is defined by the observation that can be made on the system (e.g., RGB or RGBD images, joint positions,...) and the actions the system can perform (e.g., motor commands). It also needs a task for the system that is defined as a reward function.

It will output a policy, i.e., a controller mapping observations to actions that will maximize the rewards.

3.4.5 References

Our documentation, examples and source-code are available at `https://github.com/DLR-RM/stable-baselines3`.

Publications by DREAM/RobDREAM partners

[Lesort et al., 2018] Lesort, T., Díaz-Rodríguez, N., Goudou, J.-F., and Filliat, D. (2018). State Representation Learning for Control: An Overview. *Neural Networks*, 108:379–392.

[Lesort et al., 2019] Lesort, T., Seurin, M., Li, X., Díaz-Rodríguez, N., and Filliat, D. (2019). Deep unsupervised state representation learning with robotic priors: a robustness analysis. In *IJCNN 2019 - International Joint Conference on Neural Networks*, pages 1–8, Budapest, Hungary. IEEE.

[Raffin et al., 2018] Raffin, A., Hill, A., Traoré, R., Lesort, T., Díaz-Rodríguez, N., and Filliat, D. (2018). S-rl toolbox: Environments, datasets and evaluation metrics for state representation learning. *arXiv preprint arXiv:1809.09369*.

[Raffin et al., 2019] Raffin, A., Hill, A., Traoré, R., Lesort, T., Díaz-Rodríguez, N., and Filliat, D. (2019). Decoupling feature extraction from policy learning: assessing benefits of state representation learning in goal based robotics. In *SPiRL 2019 : Workshop on Structure and Priors in Reinforcement Learning at ICLR 2019*, Nouvelle Orléans, United States. Github repo: https://github.com/araffin/srl-zoo Documentation: https://srl-zoo.readthedocs.io/en/latest/, As part of SRL-Toolbox: https://s-rl-toolbox.readthedocs.io/en/latest/. Accepted to the Workshop on Structure and Priors in Reinforcement Learning at ICLR 2019.

4 Diversity and Quality-Diversity algorithms

Researchers involved: Stephane Doncieux, Alexandre Coninx, Achkan Salehi

4.1 Description

When learning to control a robot, many attempted policies do not generate any relevant effects. A robot that would have to grasp an object may spend a long time testing movements that do not touch the object at all. A robot that learns to navigate may spend a long time stuck against a wall with all the attempted actions having no significant effect. The trace of an agent behavior when controlled by a given policy is in a high dimension space. It contains the trajectory of the agent and the modifications of the environment it has made. In general, this information can be projected into a small dimension outcome space that captures what is relevant with respect to what the robotic agent did. In the grasping experiment, for instance, the end position of the object is likely to be of interest. In the navigation task, the end position of the robot and eventually intermediate positions will probably be important.

Diversity and Quality Diversity algorithms try to find a set of policies to cover, as well as possible, this outcome space, also called a "behavior space". They are divergent algorithms, in the sense that they do not find a single optimal policy, but they converge to a set of policies that is as large as possible.

Diversity algorithms like Novelty Search explore the behavior space without taking into account any performance measure. Quality-Diversity algorithms combine this search for diversity with a performance measure that allows to keep, for policies that lead to similar outcomes, the most performing ones.

4.2 What problem does it solve?

Generating many policies instead of a single one can be useful for several reasons:

- avoiding premature convergence: the performance can be misleading and guide the search process towards suboptimal solutions. Exploring the whole behavior space avoids this issue and allows the search to discover a set of reachable optimal solutions, thus increasing the chances to find globally optimal solutions;
- illuminating the reachable behavior space: for an engineer, knowing what is actually within reach is a precious information. Integrated in the design phase, it can help compare different designs and validate a final choice;
- providing designers with elementary actions: the set of generated policies can be considered as pieces that a designer can assemble to generate an expected higher-level behavior;
- crossing the reality gap: learning algorithms require to test many different policies. To accelerate the process and avoid to damage the robot or its environment, it is thus convenient to do it in simulation. The issue is that learning may exploit features that are specific to the simulation and do not transfer well to the real platform. The diversity of generated behaviors multiplies the chances to find at least one that transfers well;
- dealing with unanticipated situations: the generated set of policies is diverse and thus contains policies that may use different approaches to reach a given effect. While some may become inapplicable in a particular context, others may still work and thus opens the possibility for the robot to speedily adapt to the situation if these policies are found rapidly.

4.3 Novelty search

Novelty search is a diversity algorithm. It is a divergent search algorithm that generates a set of diverse solutions instead of a single one. It is an evolutionary algorithm that is driven by a novelty criterion. It has been shown that it converges towards a uniform sampling of the reachable behavior space, which is an interesting property as this space cannot be directly sampled.

4.3.1 How does the method work?

Being an evolutionary algorithm, Novelty Search relies on variation and selection principles. It considers a set of solutions, called a population, that it will improve through a given number of iterations called generations. At each generation, a set of new solutions is generated and evaluated. New solutions are modified copies of solutions that are in the current population. The specificity of evolutionary algorithms is that these modifications are blind to any performance measure. Mutations act on a single solution and modify it randomly, with changes that are statistically small. Crossover operators combine several solutions and generate new solutions that benefit from features of their "parents".

Novelty search maximizes a novelty criterion. The novelty of an individual i measures the average distance of i to its k nearest neighbors in the behavior space. The neighbors are selected in the current population and in an archive containing a sampling of past solutions. At each generation, the new population is made with the set of the most novel solutions among the last population and the set of new solutions.

The outcome of novelty search is either the archive or the set of all generated solutions.

4.3.2 What meta-parameters does the method have?

As for most evolutionary algorithms, Novelty Search requires to define a population size and a number of generations. Population size p is typically around 100 to 300. The number of generations g depends on the available resources: more generations are likely to lead to better results. The most computationally demanding part of the algorithm is the evaluation of a solution. Novelty Search typically requires $p \times g$ evaluations as the set of new solutions is, in general, of size p. Parameters g and p are thus tuned to fit with available resources, while g needs to be large enough to allow convergence from the initial random solutions. It is typically above 100 and can go beyond 10000 depending on the complexity of the problem.

The number of nearest neighbors, k, is another meta parameter. It has been shown that the results do not critically depend on its value [Gomes et al., 2015]. A value of 15 is typically used.

Different strategies are used to fill the archive. Initially, solutions that were above a given novelty threshold were added, but in most recent works, the strategy is just to add a certain number of solutions that may either be the most novel or randomly selected solutions. The number of solutions added to the archive is not a critical parameter. Typical values are around 6.

Mutation and crossover operators may have their own parameters, notably their rate. Novelty search has been shown to be very robust to different levels of mutation rates [Gomes et al., 2015].

4.3.3 What are the limitations of the method?

The behavior space needs to be defined and to capture the dimensions that are worth exploring. Defining this space may not always be easy. Some recent works try to extract it automatically [Cully, 2019, Paolo et al., 2020].

Novelty search is very robust and converges in general to a set of solutions that covers the behavior space. One of its main features is that it makes the population move a lot in the behavior space from one generation to the next [Doncieux et al., 2020]. It may create issues for behaviors that are very fragile, like walking with an unstable robot (a humanoid, for instance) or navigating in narrow corridors.

4.3.4 What inputs does the method expect, and what output does it return?

The policy needs to be described, with its inputs and outputs. In many cases, a neural network with one or two hidden layers and 5 to 10 neurons per layer may do the job, but it depends on the kind of policies and on the inputs and outputs they expect (sensors and effectors). The parameter space explored by Novelty Search depends on this definition.

A simulation also needs to be provided, as these methods require up to millions of evaluations. The simulation needs to be carefully tuned to reproduce the behavior of the real robot and thus to prevent learning from converging to policies that would generate behaviors on the real robot that are too different from the behaviors in simulation.

The behavior space needs to be defined. It describes the space to explore.

The outcome of the algorithm is a set of policy parameters that lead to different behaviors. It should be noted that, in general, the policies are evaluated from the same initial state. Each policy parameter in the generated set of solutions thus indicates how to generate a given behavior from this initial state.

4.3.5 References

- [Lehman and Stanley, 2011]: the article introducing the algorithm
- [Stanley and Lehman, 2015]: a perspective on Novelty Search and on the objective-driven search processes
- [Gomes et al., 2015]: an empirical study in which the impact of different parameter values are investigated
- [Doncieux et al., 2019, Doncieux et al., 2020]: theoretical studies on novelty search

4.4 Quality-Diversity algorithms

As for Novelty Search, Quality-Diversity algorithms aim at covering a behavior space. The difference is that they take into account quality to generate solutions that are both diverse and of good quality.

4.4.1 How does the method work?

There are two main families of Quality Diversity methods. The first one relies on Novelty Search and adds a quality measure: this is Novelty Search with Local Competition (NSLC). The second family relies on a different algorithm called MAP-Elites.

Novelty Search with Local Competition is identical to Novelty Search, except that it relies on a multi-objective evolutionary algorithm (NSGA-II) and maximizes a local quality besides the novelty. It is possible to combine novelty with a fitness, but it results in an exploration that is less uniform. The reason is that the fitness is used as a global criterion. It introduces a bias in the exploration of the behavior space towards solutions with the highest fitnesses. To avoid this bias, NSLC defines a local competition criterion that counts the number of neighbors (among the k nearest used to compute novelty) that have a lower fitness. The comparison is thus local and does not significantly impact exploration.

MAP-Elites is a completely different approach. It is still an evolutionary algorithm that follows the principles of variation and selection, but it is not a standard one. In MAP-Elites, the behavior space is discretized and divided in a grid. At the initial step, random policies are generated and evaluated. Each is then placed in the cell corresponding to its behavior. Only one policy is allowed per cell and the best one is kept. Once the initial step is finished, policies are randomly selected and copied, mutated and eventually crossed. These new policies are then evaluated before trying to include them in the grid. As for the first step, a new policy is added if the corresponding behavior cell is empty or if the policy it contains is less performing. Many variants of MAP-Elites have been defined. They mostly change the way parents of new solutions are selected or how the grid is defined.

4.4.2 What meta-parameters does the method have?

NSLC does not introduce new parameters with respect to Novelty Search. Multi-objective optimization considers the two objectives to optimize at the same level and thus do not require weights to combine them.

MAP-Elites only needs to define the grid and the number of initial random solutions. As the grid size defines the maximum number of solutions considered in parallel, it should not be too large, otherwise the selection pressure will be too low and the algorithm will need a huge number of iterations to fill the cells. Typical grids contain up to thousands of cells. In vanilla MAP-Elites, the discretization needs to be manually defined, but some approaches automatically builds the grid and just need to be given the desired number of cells (CVT Map-Elites [Vassiliades et al., 2017]).

4.4.3 What are the limitations of the method?

For MAP-Elites, the main limitation is with the definition of the grid that is not always straightforward.

For NSLC, the limitations are the same as for Novelty Search. The resolution is not imposed, which may be an advantage in some situations. The advantage of MAP-Elites is its simplicity and the control it offers on the resolution of the set of solutions.

4.4.4 What inputs does the method expect, and what output does it return?

The inputs and outputs are the same as for Novelty search.

4.4.5 References

- [Mouret and Clune, 2015]: seminal work on the topic
- [Pugh et al., 2016, Cully and Demiris, 2017]: the main references on Quality Diversity
- [Kim et al., 2021]: development of a QD method adapted to simple interactions with objects
- `https://quality-diversity.github.io/`: contains an updated list of articles as well as a tutorial explaining the methodology in details and pointers to libraries implementing it

Publications by DREAM/RobDREAM partners

[Doncieux et al., 2019] Doncieux, S., Laflaquière, A., and Coninx, A. (2019). Novelty search: a theoretical perspective. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 99–106.

[Doncieux et al., 2020] Doncieux, S., Paolo, G., Laflaquière, A., and Coninx, A. (2020). Novelty search makes evolvability inevitable. In *Proceedings of the 2020 Genetic and Evolutionary Computation Conference*, pages 85–93.

[Kim et al., 2021] Kim, S., Coninx, A., and Doncieux, S. (2021). From exploration to control: learning object manipulation skills through novelty search and local adaptation. Robotics and Autonomous Systems, 136, 103710.

[Mouret and Clune, 2015] Mouret, J.-B. and Clune, J. (2015). Illuminating search spaces by mapping elites. *arXiv preprint arXiv:1504.04909*.

[Paolo et al., 2020] Paolo, G., Laflaquiere, A., Coninx, A., and Doncieux, S. (2020). Unsupervised learning and exploration of reachable outcome space. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2379–2385. IEEE.

External References

[Cully, 2019] Cully, A. (2019). Autonomous skill discovery with quality-diversity and unsupervised descriptors. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 81–89.

[Cully and Demiris, 2017] Cully, A. and Demiris, Y. (2017). Quality and diversity optimization: A unifying modular framework. *IEEE Transactions on Evolutionary Computation*, 22(2):245–259.

[Gomes et al., 2015] Gomes, J., Mariano, P., and Christensen, A. L. (2015). Devising effective novelty search algorithms: A comprehensive empirical study. In *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation*, pages 943–950.

[Lehman and Stanley, 2011] Lehman, J. and Stanley, K. O. (2011). Abandoning objectives: Evolution through the search for novelty alone. *Evolutionary computation*, 19(2):189–223.

[Pugh et al., 2016] Pugh, J. K., Soros, L. B., and Stanley, K. O. (2016). Quality diversity: A new frontier for evolutionary computation. *Frontiers in Robotics and AI*, 3:40.

[Stanley and Lehman, 2015] Stanley, K. O. and Lehman, J. (2015). *Why greatness cannot be planned: The myth of the objective*. Springer.

[Vassiliades et al., 2017] Vassiliades, V., Chatzilygeroudis, K., and Mouret, J.-B. (2017). Using centroidal voronoi tessellations to scale up the multidimensional archive of phenotypic elites algorithm. *IEEE Transactions on Evolutionary Computation*, 22(4):623–630.

5 Continual Learning

Researchers involved: David Filliat (Armines), Natalia Diaz Rodriguez (Armines), Pavan Vasishta (Armines)

5.1 Description

Because the world is constantly changing, we would like to refine the learned representation, controllers or models over time, updating them as more data becomes available, without forgetting past knowledge in case the system eventually reverts to a previous state. Given a potentially unlimited stream of data, a Continual Learning (CL) method should learn from a sequence of partial experiences where all data is not available at once. Continual learning methods may have to deal with imbalanced or scarce data problems, catastrophic forgetting, or data distribution shifts [Lesort et al., 2019]. Continual learning methods have a long history in machine learning, and face a renewed interest in the context of deep learning where the amount of training data is much larger than before, giving a renewed interest in the capacity to continually improve neural networks with new data instead of being forced to entirely retrain a network using all the past data. In this context, "continual" does not imply that the method learns online at all time steps, which is incompatible with the night phases in DREAM methods, but that it learns in an open-ended fashion, which is compatible with the DREAM paradigm

5.2 What problem does it solve?

During the DREAM project, we developed and applied CL method to learning representations and reinforcement learning. We proposed a first approach that integrates representation learning, monitoring the evolution of the data distribution to detect task evolution, and incremental training of the state representation that avoids catastrophic forgetting [Caselles-Dupré et al., 2019]. We proposed a second approach to perform CL in a Reinforcement Learning scenario on real robots by incrementally updating learned policies using distillation [Traoré et al., 2019], showing successful transfer of the resulting policies to a real robot.

5.3 Continual representation learning

5.3.1 How does the method work?

This method is based on the generative replay approach to continual learning. It works in three steps:

- During initial environment exploration, observations are recorded and used to train a Variational Auto Encoder (VAE) whose latent space is used as state representation.
- A statistical test on the VAE error then makes it possible to detect a change in the environment.
- When a change is detected, new observations are recorded in the new environment, and observations from the previous environment are generated using the VAE. These two sets of observations are merged to train a new VAE which is then used as state representation valid in both environments and to monitor further change.

5.3.2 What meta-parameters does the method have?

The main hyper-parameters are:

- The structure of the VAE (number of layers, number of neurons, size of the latent space, ...) that should be adapted depending on the size and complexity of the observations. It should be tuned on a set of observations representative of the problem.
- The number of observations needed to train the VAE, and the VAE training hyperparameters (learning rate, batch size, number of epoch). It should be set using standard Deep Learning guidelines;

5.3.3 What are the limitations of the method?

The main limitation of the method is the autonomous training of the VAE after change detection. Finding hyper-parameters that will work with any environment change is difficult: it is always possible to find parameters for a given change, but not so easy to find a 'generic' set of hyperparameters. Supervision of the training is therefore often required to check if learning the new representation succeeded. This is a general limitation of generative replay approaches in continual learning. In an industrial context, it could be easier to use standard replay by storing old samples and using them to retrain the VAE. Standard replay is a simple approach that often show very good performances in practice.

A second limitation is that this method is limited to evolution in the appearance of the environment (e.g. new objects appearing or new room discovered) but will not be sensitive to change in the dynamics of the environment.

5.3.4 What inputs does the method expect, and what output does it return?

Typical input on which we made experiments are reasonable size RGB images (e.g. 224x224x3) that are collected using a random exploration of the environment. However, the method could be adapted to other sensors such as RGBD camera or vacuum sensor.

The output of the method will be a neural network for state representation that maps an observation to a low dimension state and a statistical test to monitor environment change.

5.3.5 References

The approach is described in [Caselles-Dupré et al., 2019].

[Lesort et al., 2019] presents a review on the state of the art in continual learning.

5.4 Continual Reinforcement learning

5.4.1 How does the method work?

The general idea is to apply the replay approach for continual learning to a reinforcement learning problem where there are several tasks to learn sequentially. Instead of performing continual reinforcement learning, we continually learn the policy using distillation to update the previous policy with the new task. The method works in 3 steps:

- Train a policy on the current task using reinforcement learning (optionally including state representation learning to learn faster)
- Generate a dataset of (observations, action) using the learned policy and append it to the previous dataset
- Distill this new dataset to a new policy, i.e., train a new policy using supervised learning from this dataset. The learned policy will solve all tasks encountered up to now.

5.4.2 What meta-parameters does the method have?

The main hyper-parameters are:

- The structure of the policy networks (number of layers, number of neurons, ...) should be adapted depending on the size and complexity of the observations and the task. It should be tuned on a task representative of the problem.
- The number of observations needed for distillation and the training hyper parameters (learning rate, batch size, number of epoch, ...). It should be set using standard Deep Learning guidelines depending on the policy network size.

5.4.3 What are limitations of the method?

This method requires an external detection of the task change. It is not currently adapted to gradual task modification.

This method requires that tasks are compatible (i.e. the action for a given observation is the same for all tasks), which is not the case in all settings.

5.4.4 What inputs does the method expect, and what output does it return?

The method's input will be a sequence of reinforcement learning tasks, defined by different reward functions in the same state space (e.g. RGBD images), with the same action space (e.g., robot motor commands).

The method's output will be a sequence of policies (mapping observations to actions), each policy solving all the tasks up to the current one.

5.4.5 References

The approach is described in [Traoré et al., 2019].

Publications by DREAM/RobDREAM partners

[Caselles-Dupré et al., 2019] Caselles-Dupré, H., Garcia-Ortiz, M., and Filliat, D. (2019). S-trigger: Continual state representation learning via self-triggered generative replay. *arXiv preprint arXiv:1902.09434*.

[Lesort et al., 2019] Lesort, T., Lomonaco, V., Stoian, A., Maltoni, D., Filliat, D., and Díaz-Rodríguez, N. (2019). Continual Learning for Robotics: Definition, Framework, Learning Strategies, Opportunities and Challenges. *Information Fusion*.

[Traoré et al., 2019] Traoré, R., Caselles-Dupré, H., Lesort, T., Sun, T., Cai, G., Filliat, D., and Díaz-Rodríguez, N. (2019). DISCORL: Continual reinforcement learning via policy distillation. In *NeurIPS workshop on Deep Reinforcement Learning*, Vancouver, Canada.