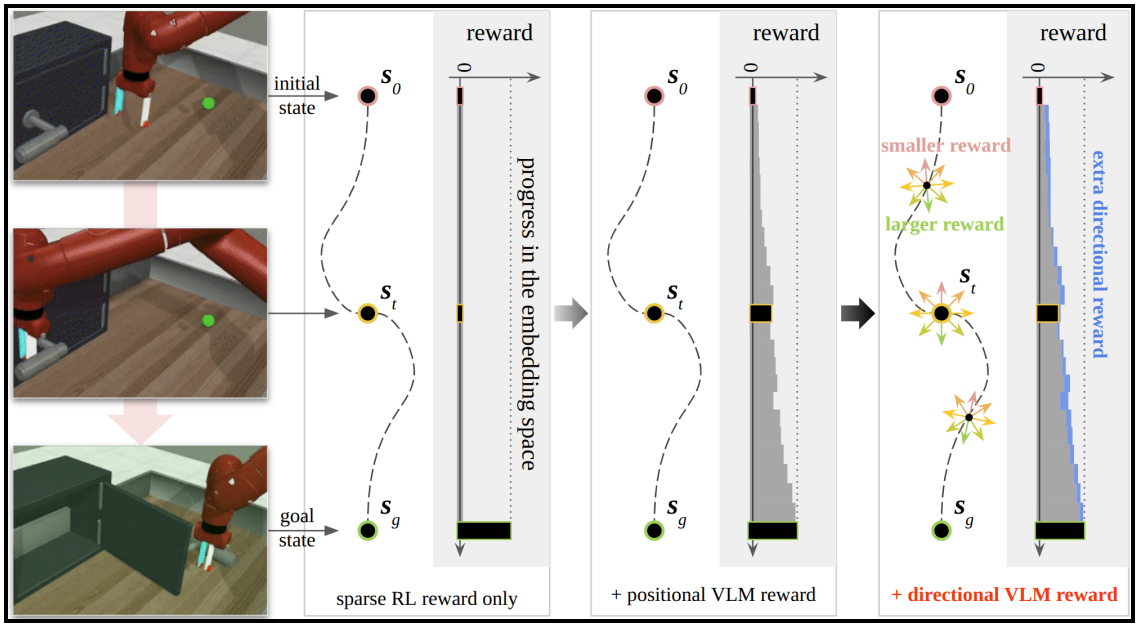
CHALMERS
UNIVERSITY OF TECHNOLOGY



Shaping Rewards with Temporal Information to Guide Reinforcement Learning

Master's thesis in Systems, Control and Mechatronics

LINUS LUNDGREN

DEPARTMENT OF ELECTRICAL ENGINEERING

CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2025
www.chalmers.se

Master's thesis 2025

Shaping Rewards with Temporal Information to Guide Reinforcement Learning

LINUS LUNDGREN

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Electrical Engineering
Division of Systems and Control
Chalmers University of Technology
Gothenburg, Sweden 2025

Shaping Rewards with Temporal Information to Guide Reinforcement Learning
LINUS LUNDGREN

Supervisors: Wenhao Lu and Zhitao Liang, Electrical Engineering
Examiner: Karinne Ramirez-Amaro, Electrical Engineering

Master's Thesis 2025
Department of Electrical Engineering
Division of Systems and Control
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 70 847 4446

Cover: VLM reward shaping with positional and directional VLM rewards.

Typeset in LaTeX
Printed by Chalmers Reproservice
Gothenburg, Sweden 2025

Shaping Rewards with Temporal Information to Guide Reinforcement Learning
LINUS LUNDGREN
Department of Electrical Engineering
Chalmers University of Technology

Abstract

Reinforcement learning (RL) methods that apply pretrained Vision-Language Models (VLMs) to compute rewards typically use a single observation of the environment to do so. This is problematic because any information emerging from the sequential nature of RL, i.e. temporal information, is therefore disregarded. This thesis explored how temporal information can be incorporated into the VLM reward computation, by first distinguishing between fixed and adaptive temporal information. In fixed temporal information, additional inputs are provided to describe the environment's progression through time, but these inputs remain unchanging throughout each episode. In contrast, adaptive temporal methods take additional inputs that can change as the episode progresses. Positional and directional rewards were defined to take advantage of fixed and adaptive temporal information respectively, along with new supervised finetuning methods for the directional reward functions. Evaluated with a sample efficiency metric over 6 robotic manipulation tasks, the best new positional rewards performed **18.4**% better than previous methods, while directional rewards performed **23.0**% better. Combining positional and directional rewards showed a **25.4**% improvement, which was the best performance achieved by any method in this thesis.

Keywords: VLM, reinforcement learning, machine learning, transfer learning, neural networks

Acknowledgements

I would like to express my deepest gratitude to my supervisors, Wenhao Lu and Zhitao Liang, for their invaluable guidance and insightful feedback throughout the course of this Master's thesis. Our weekly meetings over the past half-year were instrumental in shaping my research and overcoming challenges. Their expertise and encouragement were truly indispensable.

The computations were enabled by resources provided by the National Academic Infrastructure for Supercomputing in Sweden (NAISS), partially funded by the Swedish Research Council through grant agreement no. 2025-22597.

Linus Lundgren, Gothenburg, September 2025

List of Acronyms

Below is the list of acronyms that have been used throughout this thesis listed in alphabetical order:

CV	Computer Vision
NLP	Natural Language Processing
MDP	Markov Decision Process
MLP	Multi-Layer Perceptron
RL	Reinforcement Learning
SAC	Soft Actor-Critic
VLM	Vision-Language Model

Nomenclature

Below is the nomenclature of indices, sets, parameters, and variables that have been used throughout this thesis.

Indices

t, i, j	Indices for time steps

Sets

$\mathcal{S}$	Set of states in environment
$\mathcal{A}$	Set of actions in environment
$\mathcal{R}, \mathcal{F}$	Sets of rewards, from environment or artificial

Parameters

W_L	Trainable parameters in language network
W_I	Trainable parameters in image network
α	Strength of goal-baseline regularization
ρ	Magnitude of the VLM reward
λ	Magnitude of the directional reward
γ	Discount rate in reinforcement learning

Variables

l_g, l_b	Text description of goal and baseline
g, b	VLM text embedding of l_g and l_b
o_t	Image observation at time step t in an episode
s_t	VLM image embedding of o_t

Functions

$f^P_{W_I}, f^P_{W_L}, f^D_{W_I}, f^D_{W_L}$	Positional and directional MLP heads
$r^{VLM}_{P1}, \ldots, r^{VLM}_{P4}$	Positional reward functions
$r^{VLM}_{D1}, \ldots, r^{VLM}_{D5}$	Directional reward functions
p	Environment dynamics
π	Learned policy in reinforcement learning
csim	Cosine similarity

Contents

List of Figures

List of Tables

1

Introduction

In the past decade or so, rapid improvements in computer hardware has led to an explosion in processing power available to data scientists. In conjunction with this development, the average person now owns multiple devices that they use on a daily basis. With the internet and social media, this has made recording and sharing your lived experiences almost a societal expectation.

While this shift in behaviour has undoubtedly contributed to a great deal of psychological distress, the aforementioned data scientist is silently ecstatic, as the sheer volume of text, image, audio and video data has provided an unprecedented resource for training machine learning models. With the increasingly widespread availability of both high-level processing power and data with which it can be applied, it is not surprising that these years have also seen rapid developments in artificial intelligence research.

Two fields in particular stand out when considered in this context: computer vision (CV), and natural language processing (NLP). In CV, machine learning models can now recognize and track objects in real-time video, or generate realistic images from a variety of different inputs [1, 2]. And in NLP, the introduction of transformers has led to the development of large language models, which has achieved both academic and commercial success [3].

At the intersection of these fields, there are multi-modal vision-language models (VLMs) that take both text and image inputs [4]. These models are widely applicable, like in reinforcement learning (RL), where they can be used to compute rewards [5, 6, 7, 8]. Reinforcement learning is a machine learning paradigm which aims to optimize the actions of an agent in a dynamic environment, by maximizing the accumulated reward over time.

VLMs can create additional rewards by evaluating the similarity between the task description and an image of the current state of the system, which can then be used as a reward in RL. This is useful when rewards are rarely received (sparse rewards), such as upon task completion, in which case the task reward can be complemented by an additional VLM reward computed in every time step (dense rewards). This thesis will explore new ways in which a VLM can be used to produce dense rewards, focusing specifically on how temporal information can be exploited to do so, to be evaluated using a robot arm agent with object manipulation tasks.

1.1 Purpose

In previous research, dense VLM rewards have typically been computed as the similarity between an image observation o_t of the environment and a textual description l_g of the task goal, such as "open door". This is done by first computing the image embedding $s_t = \phi_I(o_t)$ from the image o_t observed in time step t, and the textual goal embedding $g = \phi_L(l_g)$, using image and text encoders ϕ_I and ϕ_L from a pretrained VLM.

The similarity can then be computed with the cosine similarity metric

$$\text{csim}(s_t, g) = \frac{s_t \cdot g}{\|s_t\|_2 \cdot \|g\|_2}$$

to be used as a dense reward in RL. Unfortunately, as this method only takes a single observation s_t into account, it is possible that temporal information emerging from the sequential nature of RL is lost.

The purpose of this thesis is to investigate how this temporal information can be represented, how it can be incorporated into a reward function for RL, and finally to determine the impact that this will have on the RL model's ability to learn how to perform a given task.

The hypothesis is that incorporating temporal information in the rewards will improve the RL algorithm's ability to learn the task, because temporal information should capture the change observed in the system, not just the current state. Modeling the value of the change should be better because the change represents the direct consequence of the actions that the agent is trying to optimize.

2

Theory

2.1 Reinforcement Learning

Reinforcement learning (RL) is a type of machine learning for sequential decision-making tasks, where the decision maker, commonly referred to as the agent, interacts with an environment to learn which actions are optimal through experience. The reinforcement learning problem is formalized as a Markov decision process (MDP) [9, p. 67]. The elements of an MDP are

$\mathcal{S}$	The set of states
$\mathcal{A}$	The set of actions available to the agent
$p : \mathcal{S}, \mathcal{A} \to \mathcal{S}, \mathcal{R}$	The dynamics of the environment
$\gamma \in \mathbb{R}$	The discount rate
$\mathcal{R}$	The set of rewards that the agent can receive

The set $\mathcal{S}$ determines the states in which the agent can find itself, and the set $\mathcal{A}$ represents the choices that the agent can make in these states. The dynamics $p(\hat{s}_{t+1}, r_{t+1} \mid \hat{s}_t, a_t)$ describes the state-transition and reward probabilities given that the agent takes the action a_t when it finds itself in state $\hat{s}_t$, where the possible rewards are taken from $\mathcal{R}$. The goal of an RL problem is for the agent to learn how to maximize the accumulated reward over time, also known as the return

$$G_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}$$

given the discount rate γ, which defines the present value of future rewards.

The RL states use the notation $\hat{s}$ to differentiate from s, which is used to represent image embeddings. The $\hat{s}$ states are read directly from the environment, and are simply vectors describing the positions of objects in the environment.

During training, an RL algorithm learns a policy $\pi(a \mid \hat{s})$ which maps states $\hat{s} \in \mathcal{S}$ to actions $a \in \mathcal{A}$. In this thesis, the algorithm used is Soft Actor-Critic (SAC), which is an RL algorithm that uses neural networks to learn the policy directly [10].

The policy is learned by training on collected experience, stored in a replay buffer as $(\hat{s}_i, a_i, \hat{s}_{i+1}, r_{i+1})$ tuples, representing the observed state transitions and their corresponding rewards. In this thesis, these tuples have an additional member s_i, to

save image embeddings for the purpose of training the VLM reward function. Typical RL tasks are episodic, which means that these tuples are gathered in discrete attempts at solving the task, and the environment is reset between each attempt. These attempts are more commonly referred to as episodes, or trajectories, which are positive if the agent succeeded, and negative if it failed.

The rewards received by the agent from the environment can be characterized by their "sparseness". A sparse reward is a reward which is received less often, such as upon task completion. On the other end of the sparseness spectrum are dense rewards, which are received often, if not in every time step. For example, a dense reward could be computed for a pole balancing task in every time step from the angle between the ground and a pole.

Dense rewards are typically better for RL, so long as a larger return is a proxy for task completion. They are more useful because they make the policy more generalizable to unseen states, as the RL algorithm has access to more data. For the pole balancing task with access to dense rewards, the agent will easily be able to learn to keep the angle perpendicular to the ground. But it would be much harder, for example, if it only received a reward once it manages to balance it for an entire episode. Dense rewards are also particularly useful in the beginning of training, as the agent will have no rewards to guide it until the first success, if that is the only time a reward is received.

Reinforcement learning has a wide range of applications, including finance, energy efficiency, transportation, signal processing, robotics, and more [11]. Robotics is the focus of this thesis, as the new methods will be evaluated by tasking a robot arm to solve manipulation tasks.

2.2 Reward Shaping

Reward shaping is the method of computing "artificial" rewards outside of the task reward provided by the environment [12]. From a previously "pure" MDP $(\mathcal{S}, \mathcal{A}, p, \gamma, \mathcal{R})$, a new MDP $(\mathcal{S}, \mathcal{A}, p, \gamma, \mathcal{R} + \mathcal{F})$ can be constructed with additional rewards $\mathcal{F}$. Reward shaping is especially useful to provide dense rewards in environments where the reward is sparse. In these cases, the new dense reward can help guide the agent where the sparse reward is insufficient.

For instance, if the agent only gets a reward upon task completion, then a new shaping reward could be the distance between the current state and a goal state, which can easily be computed in each time step. If an image can be generated of the environment, then a VLM can also be used to compute a shaping reward as the similarity between state images and a task description. This is the method which will be further developed in this thesis.

2.3 Vision-Language Models

Vision-Language Models (VLMs) are neural network models that exist in the intersection of computer vision and natural language processing [4]. A wide variety of VLMs have been developed, but this thesis will focus on contrastive-based VLMs. These are VLMs that are trained on a dataset of image-text pairs $\mathcal{P}$. The pairs in this dataset become the positive pairs, while negative pairs are constructed with the same dataset, but with images and text that are not paired in the original dataset. Then a metric like cosine similarity is made larger for positive pairs and smaller for negative pairs, for instance by minimizing a loss function like infoNCE

$$\mathcal{L}_{\text{infoNCE}} = -\sum_{i \in \mathcal{P}} \log \left(\frac{\exp(\text{csim}(s_i, c_i))/\tau}{\sum_{j=1}^{N} \exp(\text{csim}(s_i, c_j))/\tau} \right)$$

where s_i is the embedding of an image with corresponding caption embedding c_i from the positive pairs, while c_j are caption embeddings that are not paired with s_i, and τ is a temperature parameter to control the scale of the similarity scores.

Contrastive-based VLMs have been found to be especially useful for RL, as the similarity between the current image of the system and a textual description of the goal can be used as the additional reward for reward shaping [5, 6, 7, 8].

An alternative objective for training a VLM is to use Language-Image Value (LIV) learning, which is a pretraining objective specifically developed to encode a notion of task completion based on tasks specified through language or images [13]. These VLMs are therefore especially useful for reward shaping with language-specified tasks, which is why this thesis uses the image and text encoder from a LIV trained VLM to produce image and text embeddings.

2.4 Transfer Learning

Neural networks are machine learning models that can perform incredibly well on a wide variety of problems, but they still require relatively large datasets to be able to recognize the relevant features in the data. This is a problem if it is going to be applied to a domain where gathering data is difficult and/or costly. In these scenarios, a method called transfer learning can be used to reduce the amount of data necessary to train the model [14].

Transfer learning works by letting the model initially train on a similar task, for which the data collection is easier. This is called pretraining, and allows the network to learn features that will most likely be useful for the downstream task as well, due to the similarity between the two task domains. This pretrained model can then be finetuned using the smaller dataset collected for the downstream task, so that it can learn which of the features learned in pretraining are relevant for the task at hand.

Typically, the model contains a main body neural network with a multilayer perceptron (MLP) head attached to the end to produce task predictions. In transfer learning, the main body is the same for the pretraining and downstream task, but

the head is replaced. Figure 2.1 shows an illustration of this process. This thesis will transfer a LIV pretrained VLM for the reward function of an RL model, where new heads are attached to the image and text encoder networks. These heads will then be finetuned to produce better rewards using image embeddings gathered during the RL training process.

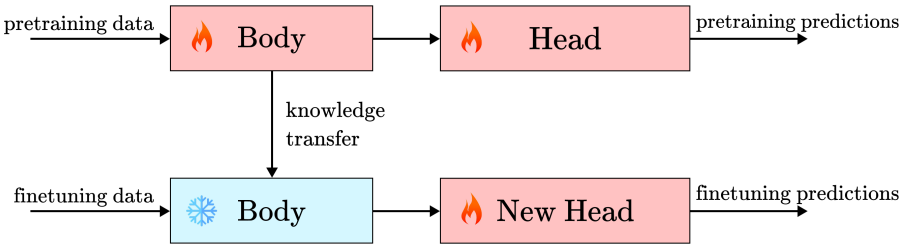


Figure 2.1: Illustration of transfer learning. First, the model is pretrained on a task for which the data collection is easy, with a head to make predictions for the pretraining objective. Once the pretraining is complete, the main body parameters are transferred, but a new head replaces the old head to produce predictions for the new task. Typically, only the parameters in the new head are finetuned for the new task, while the transferred main body parameters are frozen. This process transfers knowledge from the pretraining task through the parameters in the body, which is more effective if the pretraining and downstream tasks are similar. Parameters being trained are marked with the fire symbol, while frozen parameters are marked with the snowflake symbol.

3

Preliminaries

This chapter will first lay out the current state of research regarding the use of temporal information in the VLM RL setting. Then the purpose of this thesis will be formalized with 4 research questions, and an appropriate scope will be set.

3.1 Related Work

3.1.1 Delta Features

In [6] they use VLMs for reward shaping, and introduced a method of transforming their embeddings using what they call delta features. In this method, they subtract a baseline text embedding b from the goal text embedding g, where they represent the initial and end state of the task if successfully performed, such as "closed microwave" and "open microwave" respectively. Then the initial image embedding s_0 is subtracted from the image embedding s_t, and the reward is computed as the cosine similarity between these two components

$$r^{VLM}(s_t) = \text{csim}(s_t - s_0, g - b)$$

This can be interpreted as using temporal information because they compute the visual component $s_t - s_0$ using image embeddings from two different time steps. Additionally, their method of defining their goal and baseline can also be interpreted as using temporal information, as they describe what should be observed before and after the task has been completed.

3.1.2 Goal-baseline regularization

Goal-baseline regularization is another method introduced in [7] which, like [6], incorporates a textual baseline b. This baseline is used to remove components of the image embedding s_t that are orthogonal to $g - b$, since these components should be less relevant for solving the task. Typical cosine similarity can be rewritten as

$$\text{csim}(s_t, g) = \frac{s_t \cdot g}{\|s_t\|_2 \|g\|_2} = 1 - \frac{1}{2}\|s_t - g\|_2$$

so long as s_t and g are normalized to unit vectors in the final expression.

In [7] they use this reformulation to define a new state representation. This new representation is a weighted sum between s_t and the projection of s_t onto the line spanned by g and b. This gives a new VLM reward function

$$r^{VLM}(s_t) = 1 - \frac{1}{2}\|\alpha \text{proj}_L s_t + (1-\alpha)s_t - g\|$$

where L is the line spanned by g and b, and α represents the regularization strength, i.e. how much of the components orthogonal to L will be removed through projection. As explained in Section 3.1.1, this could be interpreted as a temporal information method, if the goal and the baseline are defined as they are in [6].

3.1.3 The FuRL Algorithm

The problem with the VLM rewards in [6, 7], and also [5], is that they are *zero-shot*. This means that they transfer the VLM, but perform no finetuning. The authors of [5] found that maximizing the zero-shot VLM reward generally also led to a maximization of their ground truth reward. But with no finetuning, these VLM rewards can be inaccurate, because the VLM was not specifically pretrained on images of the environment that the agent finds itself in.

One desirable behavior of the reward function could be for it to be strictly increasing on all trajectories from the initial state to the goal state. This way, the agent can simply learn to take actions for which the reward increases the most. But since the VLM was likely not pretrained on images of the given environment, it is unlikely that the VLM reward function would behave this way. This means that using the raw, zero-shot similarity as a reward risks the agent getting stuck in non-goal points where the VLM reward function is locally maximized.

To solve this, the authors of [8] attach two MLP heads f_{W_I} and f_{W_L} to the VLM image and text encoders. These heads are then used to compute the VLM reward as

$$r^{VLM}(s_t) = \text{csim}(f_{W_I}(s_t), f_{W_L}(g))$$

where s_t is the image embedding from the VLM's image encoder ϕ_I, and g is the goal text embedding from the VLM's text encoder ϕ_L. The f_{W_I} MLP head is attached to ϕ_I, with its trainable parameters W_I, while f_{W_L} is its equivalent for ϕ_L, with trainable parameters W_L. Figure 3.1 shows the reward computation flowchart for this VLM reward.

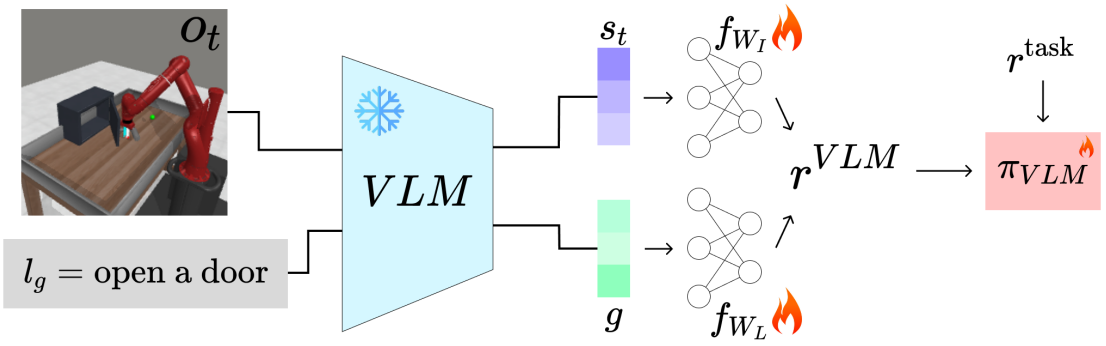


Figure 3.1: Flowchart of the FuRL reward computation. Parameters in transferred VLM remain frozen (snowflake symbol), while MLP heads f_{W_I} and f_{W_L} are finetuned (fire symbols).

But these heads are not pretrained, so the authors also introduced the Fuzzy VLM rewards aided RL (FuRL) finetuning algorithm. The FuRL algorithm works by collecting and distinguishing between negative and positive trajectories, and finetuning with states from these trajectories, by minimizing the ranking loss ℓ_δ:

$$\ell_\delta(s_p, s_n) = \max(0, r^{VLM}(s_n) - r^{VLM}(s_p) + \delta)$$

The ranking loss takes two image embeddings s_p and s_n, either from the same trajectory or two different trajectories, and assigns a preference to the s_p state by increasing its reward relative to the s_n state reward. In essence, it will minimize the reward for the s_n state and maximize the reward for the s_p state, but only if the current reward for s_p is at most δ higher than the reward for s_n. If it exceeds that threshold, then the (s_p, s_n) pair is disregarded for the optimization step.

The FuRL finetuning algorithm proceeds in two separate stages: before the first positive trajectory has been collected, and after the first positive trajectory has been collected. The supervised finetuning of the reward function is done interchangeably with the RL training, where the following reward is provided to the RL algorithm (which is Soft Actor-Critic):

$$r = r_t^{\text{task}} + \rho \cdot r^{VLM}(s_t)$$

In summary, the reward provided to the RL algorithm in time step t is the sparse task reward r_t^{task} plus the dense VLM reward $r^{VLM}(s_t)$, the magnitude of which is controlled by ρ.

3.1.3.1 Stage 1

In the first stage, the FuRL algorithm only has access to states from negative trajectories for the finetuning process. This makes it somewhat inefficient at producing a valuable reward, but a preference can still be given to states based on ℓ_2 distance to the goal state. This preference can be expressed with the ranking loss as follows:

$$\mathcal{L}_{neg-neg} = \ell_\delta(s_i^n, s_j^n) \quad (3.1)$$
$$\text{s.t. } \|s_i^n - s_g\|_2 \leq \|s_j^n - s_g\|_2$$

In this loss, s_g is an image embedding for a state where the task has been successfully performed, and the n superscript indicates that the image embeddings are taken from negative trajectories. The constraint ensures that the goal s_g is closer to s_i^n than s_j^n, so that the ranking loss preference is given to the state that is closer to the goal. The finetuning with (3.1) is illustrated in Figure 3.2.

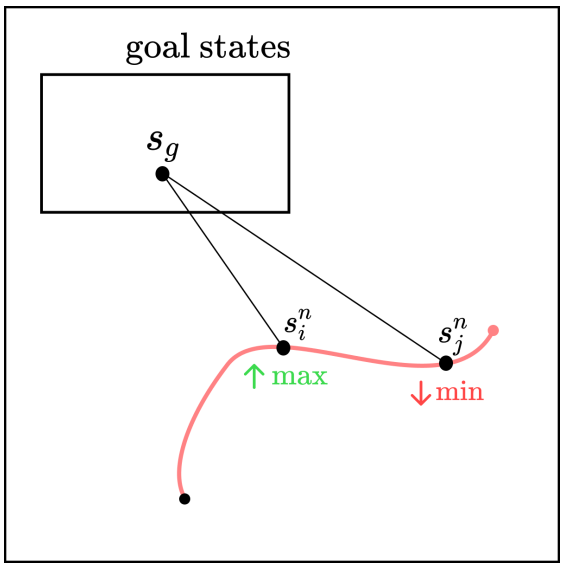


Figure 3.2: The larger square represents the set of all image embeddings, while the smaller "goal states" rectangle represents the set of embeddings in which the task has been completed successfully. The ↑ max and ↓ min annotations indicate the states for which the reward is maximized or minimized respectively, and the red line represents a negative trajectory. As illustrated, when only negative trajectories are available, the state closer to the goal state gets a higher reward, while the state further away gets a lower reward.

Minimizing (3.1) will generally make the reward higher as the agent gets closer to the goal, which should help guide the agent towards collecting the first positive trajectory, so that the FuRL algorithm can move on to its second stage.

3.1.3.2 Stage 2

With access to positive trajectories, the objective is replaced by

$$\mathcal{L} = \mathcal{L}_{pos-neg} + \mathcal{L}_{pos-pos}, \text{ where} \tag{3.2}$$
$$\mathcal{L}_{pos-neg} = \ell_\delta(s^p, s^n), \text{ and} \tag{3.3}$$
$$\mathcal{L}_{pos-pos} = \ell_\delta(s_i^p, s_{i-n}^p) \tag{3.4}$$

The p superscript indicates that the states are taken from positive trajectories. The first ranking loss $\mathcal{L}_{pos-neg}$ (3.3) gives preference to states from positive trajectories over states from negative trajectories, which should make the reward higher if the agent retraces previous positive trajectories. The second ranking loss $\mathcal{L}_{pos-pos}$ (3.4) gives preference to states further along the positive trajectories, which should make the reward strictly increasing along the positive trajectories, as desired.

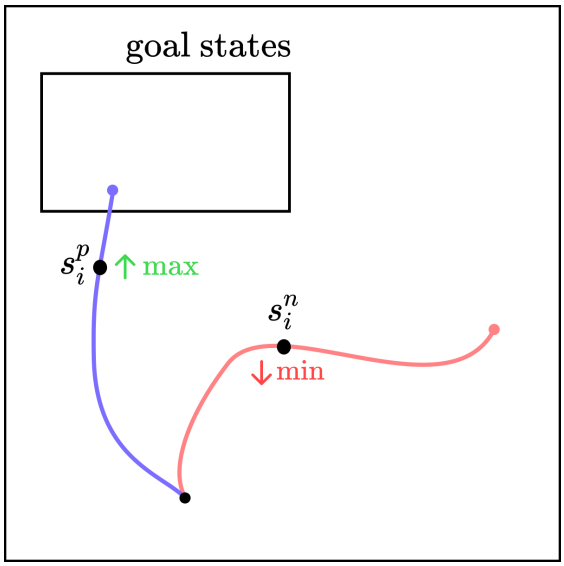


Figure 3.3: Finetuning with $\mathcal{L}_{pos-neg}$ (3.3). Same setup as in Figure 3.2, but with a positive trajectory illustrated by a blue line. The optimization step will maximize reward for the state from the positive trajectory and minimize it for the state from the negative trajectory, until the positive state reward is at least δ higher than that of the negative state.

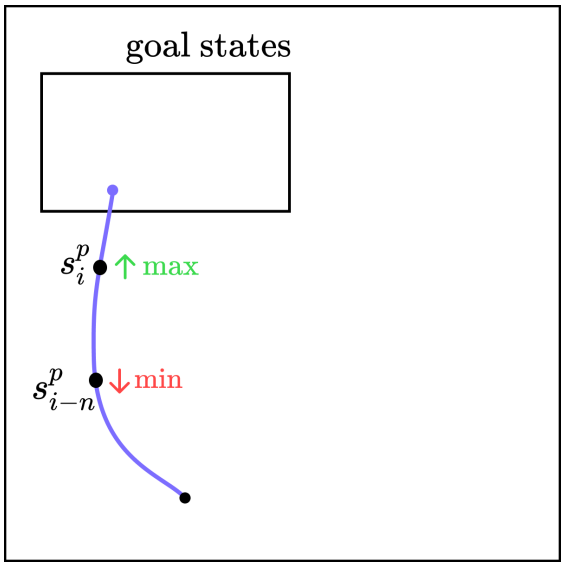


Figure 3.4: Finetuning with $\mathcal{L}_{pos-pos}$ (3.4). Same setup as in 3.2, but with with only a single positive trajectory (blue line). The optimization step will maximize the reward for the state further along the positive trajectory, and minimize it for the state further back, as long as the reward for the later state is not already δ higher.

The second stage of the FuRL algorithm could be interpreted as using temporal information, since it aims to make the rewards for positive trajectory states in later time steps larger than earlier than the rewards for earlier states. However, it is important to note that the FuRL algorithm does not use temporal information in its reward computation, rather only in how the reward function is trained. The trained FuRL reward function does not take the temporal information present in the current observation into account. Instead, it looks at the current state, and then makes an inference as to when that state would have been observed in previous positive trajectories, and computes a reward based on that inferred moment in time.

The FuRL algorithm will be the basis of the methods developed in Chapter 4. However, this thesis will go beyond the finetuning algorithm, to also consider how temporal information can be incorporated into the inputs of the reward function itself.

3.2 Research Questions

With this background in mind, the purpose of this thesis can be formalized with four central research questions:

1. How is temporal information defined, and how can it be represented for efficient reward shaping in a sparse reward RL context?
2. How can a VLM reward function be modeled such that it takes advantage of temporal inputs, and what should its training procedure look like?
3. How can the performance of these new VLM RL methods be evaluated?
4. Given this evaluation, how do the new methods compare to existing methods?

3.3 Scope

This project can potentially be taken in a bunch of different directions, so it is important to be specific regarding what will or will not be investigated. This delimitation can be summarized in the following list of points:

- The developed algorithms will be evaluated using robotic manipulation tasks from the Metaworld benchmark [15].
- The fixed-goal versions of the Metaworld environments will be used, to reduce randomness and make training faster.
- Only model free reinforcement learning methods will be considered.
- All new methods will be based on the FuRL finetuning method introduced in [8].
- The major focus will on how the algorithm can use negative *and* positive trajectories to produce a better reward, not how negative trajectories can be used to collect positive trajectories. Each reward and algorithm will therefore be developed with the performance in the second stage in mind.

4

Methods

The purpose of this chapter is to describe how temporal information can be used for reward shaping in an RL setting. But first, a formal definition for temporal information is given.

4.1 Defining Temporal Information

Temporal information can be summarized as being any information that describes the progression of the environment through time. But this definition is still very general. For instance, in delta features the visual component of cosine similarity is $s_t - s_0$. The result can be interpreted as a type of temporal information, since it describes how the environment has changed since the start of the episode. But what it does not capture is the *current* change present in the environment, because for example, the state s_{t-1} has no effect on $s_t - s_0$, so long as s_t remains the same.

With this distinction in mind, temporal information can be divided into two categories. The first is *fixed* temporal information, which is temporal information provided through inputs that do not change as the episodes progresses. This includes comparisons to fixed states like the initial state s_0 or goal states s_g. Rewards that only use fixed temporal temporal information will be referred to as positional rewards, because they still represent the value of the environment's position, but viewed from a different (fixed) perspective. Rewards that use s_t but do not use any temporal information, like simply $\text{csim}(s_t, g)$, are also positional rewards because they also model the value of the position.

But if the additional inputs can change throughout the episode, like s_{t-k} for some constant k, then this is instead *adaptive* temporal information. Rewards that use adaptive temporal information are directional rewards, since they can model the value of the direction in which the environment is currently changing.

In this thesis, any temporal information present in textual inputs will be fixed temporal information, because the text descriptions will remain fixed throughout each episode. This does not mean that a textual input cannot contain adaptive temporal information. For instance, the environment could continuously output text observations, then these could be used to capture the current change. This is not something that will be explored in this thesis, however.

Positional and directional rewards will be computed separately, which means that it is not a choice of one or the other. If positional rewards contribute to helping the RL algorithm learn the task, and adding directional rewards is also beneficial, then they can both be used together to optimize performance. The final VLM reward becomes

$$r^{VLM}(s_t) = r_P^{VLM}(s_t) + \lambda \cdot r_D^{VLM}(s_t)$$

where r_P^{VLM} and r_D^{VLM} are the positional and directional reward functions respectively, and the λ parameter controls the size of the directional reward in relation to the positional reward. The VLM reward is then summed with the sparse task reward as in FuRL. The reward observed by the RL algorithm is therefore

$$r_t = r_t^{\text{task}} + \rho \cdot \left(r_P^{VLM}(s_t) + \lambda \cdot r_D^{VLM}(s_t) \right)$$

4.2 The Positional Reward

The shaping reward used in the FuRL paper [8] is an example of a positional reward, representing the value of the current environment position. This positional reward can potentially be improved by replacing the state and goal embedding with delta features [6],

$$r^{VLM}(s_t) = \text{csim}(s_t - s_0, g - b)$$

This new reward function contains fixed temporal information in its visual component $s_t - s_0$ because it takes states from two different time steps into account, where s_0 is fixed. It can also contain fixed temporal information in the textual component, depending on how the goal and baseline texts are defined.

Alternatively, goal-baseline regularization [7]

$$r^{VLM}(s_t) = 1 - \frac{1}{2}\|\alpha \text{proj}_L s_t + (1-\alpha)s_t - g\|$$

can be used instead of typical cosine similarity, where fixed temporal information can be provided through g and b (L is the line spanned by g and b).

The contribution of this thesis for positional rewards is the combination of delta features [6] and goal-baseline regularization [7] with FuRL [8]. To combine these methods, MLP heads $f_{W_I}^P$ and $f_{W_L}^P$ need to be incorporated into the reward function, where the superscript P indicates that they are used in a positional reward function.

4.2.1 Delta Features

The new positional rewards are still based on the FuRL method, so they finetune $f_{W_I}^P$ and $f_{W_L}^P$ using the same two-stage finetuning method with the ranking loss

$$\ell_\delta(s_p, s_n) = \max(0, r^{VLM}(s_n) - r^{VLM}(s_p) + \delta)$$

In the first stage, the finetuning objective is

$$\begin{aligned} &\mathcal{L}_{neg-neg}^P = \ell_\delta(s_i^n, s_j^n) \\ &\text{s.t. } \|s_i^n - s_g\|_2 \leq \|s_j^n - s_g\|_2 \end{aligned} \tag{4.1}$$

where all states are from negative trajectories, as no positive trajectories have been collected yet. Then, once the first positive trajectory has been collected, $f^P_{W_I}$ and $f^P_{W_L}$ are instead finetuned on

$$\mathcal{L}^P = \mathcal{L}^P_{pos-neg} + \mathcal{L}^P_{pos-neg}, \text{ where} \tag{4.2}$$
$$\mathcal{L}^P_{pos-neg} = \ell_\delta(s^p, s^n), \text{ and} \tag{4.3}$$
$$\mathcal{L}^P_{pos-pos} = \ell_\delta(s^p_i, s^p_{i-k}) \tag{4.4}$$

The finetuning objectives for delta features still use the same FuRL ranking loss, but the major difference is how the VLM reward is computed, along with how the MLP heads $f^P_{W_I}$ and $f^P_{W_L}$ can be incorporated. The first formulation for delta features, referred to as P1 since it is the first positional reward function, is

$$r^{VLM}_{\text{P1}}(s_t) = \text{csim}(f^P_{W_I}(s_t - s_0), f^P_{W_L}(g - b))$$

where the subtraction is computed before the heads, in the VLM embedding space, i.e. with the output from the VLM encoders directly. The flowchart for this computation can be found in Figure 4.1.

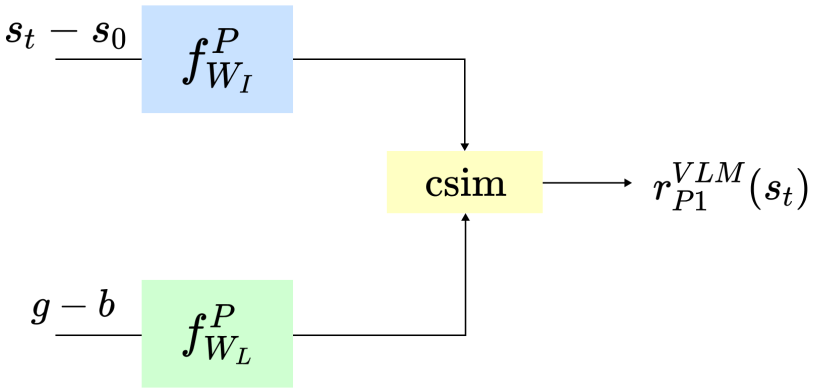


Figure 4.1: Flowchart of delta features computation, where the difference is computed before $f^P_{W_I}$ and $f^P_{W_L}$, i.e. $r^{VLM}_{\text{P1}}(s_t) = \text{csim}(f^P_{W_I}(s_t - s_0), f^P_{W_L}(g - b))$.

With the first formulation computing the subtraction before the MLP heads, a natural alternative is to compute the subtraction after the heads instead, giving

$$r^{VLM}_{P2}(s_t) = \text{csim}(f^P_{W_I}(s_t) - f^P_{W_I}(s_0), f^P_{W_L}(g) - f^P_{W_L}(b))$$

which is illustrated in Figure 4.2.

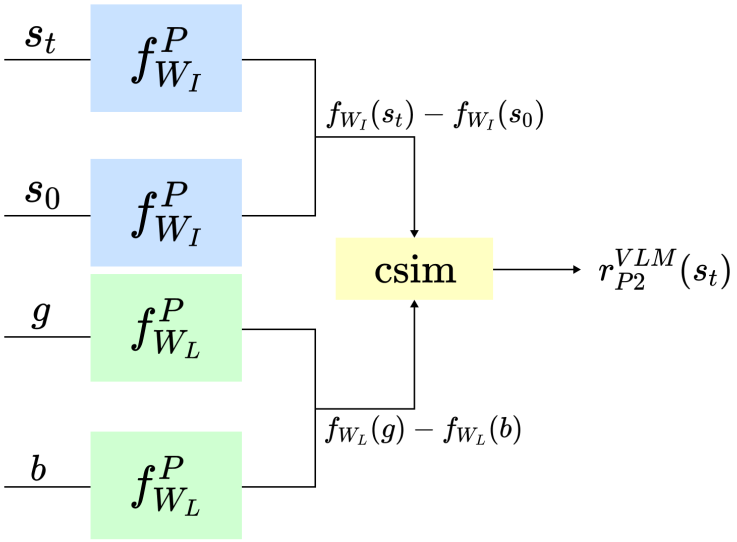


Figure 4.2: Flowchart of delta features computation, where the difference is computed after $f^P_{W_I}$ and $f^P_{W_L}$, i.e. $r^{VLM}_{\text{P2}}(s_t) = \text{csim}(f^P_{W_I}(s_t) - f^P_{W_I}(s_0), f^P_{W_L}(g) - f^P_{W_L}(b))$.

4.2.2 Goal-Baseline Regularization

For goal-baseline regularization, the finetuning objectives are still the same as in FuRL, but with the rewards replaced by goal-baseline regularized cosine similarity

$$r^{VLM}(s_t) = 1 - \frac{1}{2}\|\alpha \text{proj}_L s_t + (1-\alpha)s_t - g\|_2^2 \tag{4.5}$$

where $\text{proj}_L s$ is the projection of s_t onto L, which is the line spanned by the textual goal embedding g and textual baseline embedding b.

In [7] where they introduced goal-baseline regularization, they perform this regularization by first producing VLM image embeddings s and text embeddings g and b, and then just computing VLM reward as described in (4.5). This thesis requires a different formulation, as $f^P_{W_I}$ and $f^P_{W_L}$ also have to be accounted for.

These heads can be incorporated by simply running the outputs from the VLM image and text encoders directly into $f^P_{W_I}$ or $f^P_{W_L}$, and using the result to compute a goal-baseline regularized similarity

$$r^{VLM}_{\text{P3}}(s_t) = 1 - \frac{1}{2}\|\alpha \text{proj}_L(f^P_{W_I}(s_t)) + (1-\alpha)f^P_{W_I}(s_t) - f^P_{W_L}(g)\|_2^2$$

where L would be $f_{W_L}(g) - f_{W_L}(b)$. This computation is illustrated in Figure 4.3.

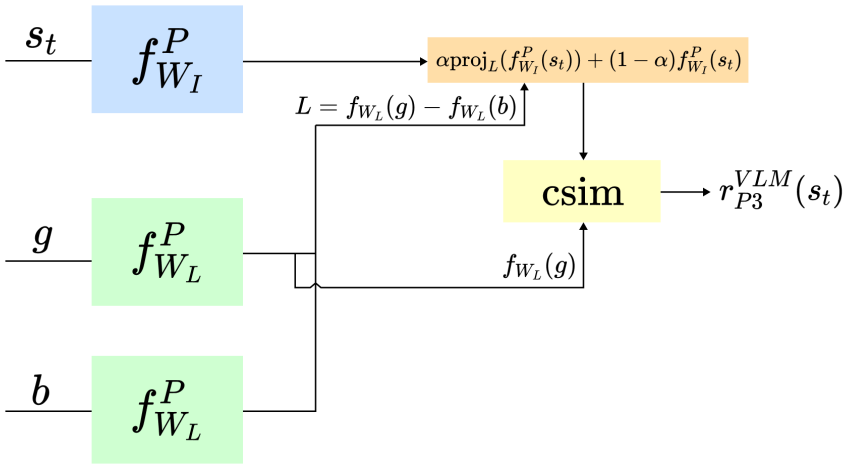


Figure 4.3: Flowchart of goal-baseline regularized reward computation, where the regularization is computed after $f^P_{W_I}$ and $f^P_{W_L}$, i.e. $r^{VLM}_{\text{P3}}(s_t) = 1 - \frac{1}{2}\|\alpha\text{proj}_L(f^P_{W_I}(s_t)) + (1-\alpha)f^P_{W_I}(s_t) - f^P_{W_L}(g)\|^2_2$.

However, just as the subtraction can be performed before or after $f^P_{W_I}$ and $f^P_{W_L}$ in delta features, an alternative implementation is to perform the goal-baseline regularization first, and then pass the result to $f^P_{W_I}$. Since goal-baseline regularization is essentially a reformulation of the cosine similarity such that the state s_t is replaced by the weighted sum $\alpha\text{proj}_L s_t + (1-\alpha)s_t$, this weighted sum can instead be computed in the VLM embedding space, resulting in

$$r^{VLM}_{\text{P4}}(s_t) = 1 - \frac{1}{2}\|f^P_{W_I}(\alpha\text{proj}_L s_t + (1-\alpha)s_t) - f^P_{W_L}(g)\|^2_2$$

where L is simply $g - b$. The flowchart for this reward computation is shown in Figure 4.4.

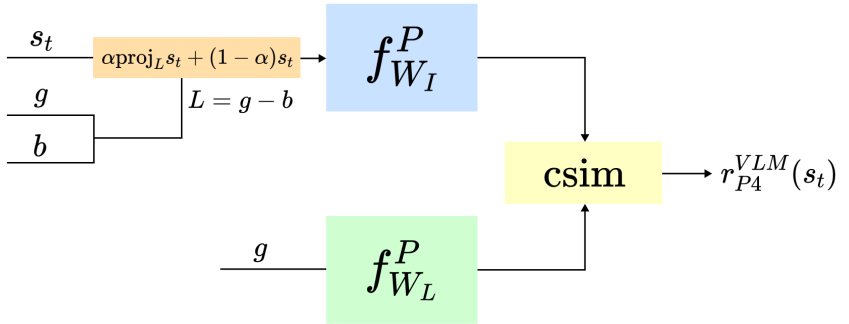


Figure 4.4: Flowchart of goal-baseline regularized reward computation, where the regularization is computed before $f^P_{W_I}$ and $f^P_{W_L}$, i.e. $r^{VLM}_{\text{P4}}(s_t) = 1 - \frac{1}{2}\|f^P_{W_I}(\alpha\text{proj}_L s_t + (1-\alpha)s_t) - f^P_{W_L}(g)\|^2_2$.

4.3 The Directional Reward

The directional reward is computed using the same transferred VLM image and text encoders ϕ_I and ϕ_L, but finetunes a new pair of MLP heads $f^D_{W_I}$ and $f^D_{W_L}$, with superscript D because they are used for the directional reward.

This chapter will introduce 5 different directional reward functions $r^{VLM}_{\text{D1}}, \ldots, r^{VLM}_{\text{D5}}$. Temporal reward shaping with both positional and directional rewards, i.e.

$$r_t = r^{\text{task}} + \rho \cdot \left(r^{VLM}_{\text{P}}(s_t) + \lambda \cdot r^{VLM}_{\text{D}}(s_t) \right)$$

can be illustrated with a flowchart, as in Figure 4.5. This figure shows an example process of going from image/text inputs to directional and positional rewards for RL policy training, using r^{VLM}_{D3} and standard FuRL positional reward.

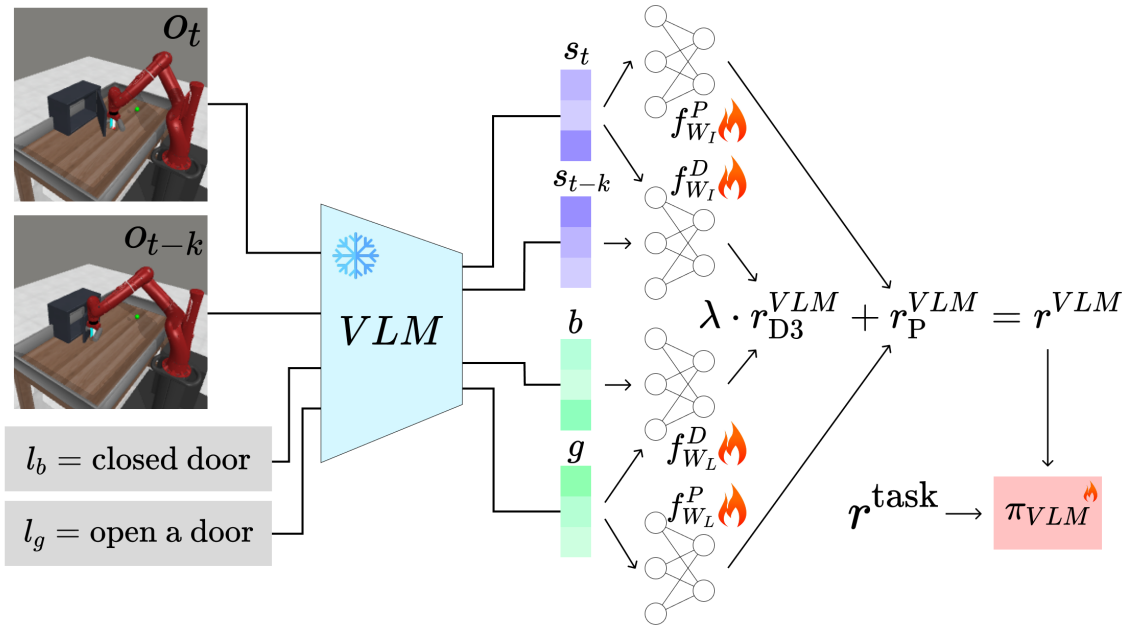


Figure 4.5: Example computational flowchart of temporal VLM reward shaping, including both positional and directional rewards, with r^{VLM}_{D3} and original FuRL positional reward. Frozen VLM encoders take image and text inputs, and produces embeddings s_t, s_{t-k}, b and g. These are fed into positional and directional MLP heads $f^P_{W_I}$, $f^P_{W_L}$, $f^D_{W_I}$ and $f^D_{W_L}$ that will be finetuned. Positional and directional rewards are then summed and sent to the RL algorithm along with the sparse task reward to train the policy π_{VLM}.

The directional reward inputs are similar to the positional reward inputs in that they both compute the cosine similarity between a visual and textual component. The difference is that the visual component in the directional reward will incorporate adaptive temporal information, as will be explained in Section 4.3.2. The textual component of the directional reward is $g-b$ as in delta features, although with a different interpretation, which will be explained in Section 4.3.1.

4.3.1 The Textual Component

In delta features, $g-b$ can be interpreted as the information in the baseline description (b, describing the task as unsolved) subtracted from the goal description (g, describing the task as solved). This should create a goal embedding of higher quality, because the irrelevant information present in both the baseline and the goal should be removed, so that only the difference between them remains.

To elaborate, the door-open Metaworld environment can be taken as an example, where the goal and baseline descriptions could be

$$l_g = \text{"open a door", and}$$
$$l_b = \text{"closed door"}$$

In the positional interpretation, the subtraction of b from g will remove information present in both textual embeddings, such as the fact that there is a door. Regardless of whether the agent finds itself in a state where the task is unsolved or solved, we would always expect this information to be present in the environment, so it's not relevant for solving the task. Instead, the critical information is whether the door is opened or closed, which should still be present after subtraction.

Similarly, the subtraction of s_0 from s_t will remove irrelevant visual information, such as the table in front of the agent, or other properties of the environment that remain the same whether the task is solved or not. For delta features, computing the similarity between the textual and visual components will therefore constitute a comparison of only the most relevant information in each of them, which should make it easier for the model to recognize the more critical elements.

This interpretation of $g-b$ is essentially a new goal embedding, improved by subtracting away irrelevant information. But in the directional reward, $g-b$ can instead be interpreted as the generally desirable direction, moving from the baseline towards the goal. The directional reward will therefore be computed as the similarity between the direction the agent should move in, $g-b$, and the direction that was actually observed, described by the visual component, as defined in the following subsection.

4.3.2 The Visual Component

The visual component was constructed in five different ways, leading to five directional reward functions. Adaptive temporal information is provided through the input Δs_t. This term contains adaptive temporal information describing the change $s_{t-k} \rightarrow s_t$, through subtraction. As in previous methods, this difference can be computed before or after $f^D_{W_I}$. However, to make notation more interpretable, this difference will be represented as $\Delta s_t = s_t - s_{t-k}$ regardless of when the subtraction is computed. With this in mind, the first two directional reward functions are

$$r^{VLM}_{\text{D1}}(\Delta s_t) = \text{csim}(f^D_{W_I}(s_t - s_{t-k}), f^D_{W_L}(g-b)), \text{ and}$$
$$r^{VLM}_{\text{D2}}(\Delta s_t) = \text{csim}(f^D_{W_I}(s_t) - f^D_{W_I}(s_{t-k}), f^D_{W_L}(g) - f^D_{W_L}(b))$$

The moment where the subtraction is computed in relation to $f^D_{W_I}$ is not explicitly provided in the input, but is instead implied based on the chosen reward function

(D1 or D2). Reward computation flowcharts for these reward functions can be found in Figures 4.6 and 4.7.

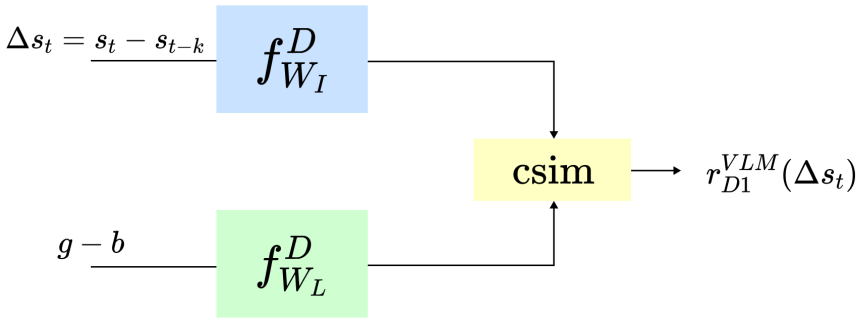


Figure 4.6: Flowchart of directional reward computation, using only directional input, where the difference is computed before $f^D_{W_I}$ and $f^D_{W_L}$, i.e. $r^{VLM}_{\text{D1}}(\Delta s_t) = \text{csim}(f^D_{W_I}(s_t - s_{t-k}), f^D_{W_L}(g - b))$

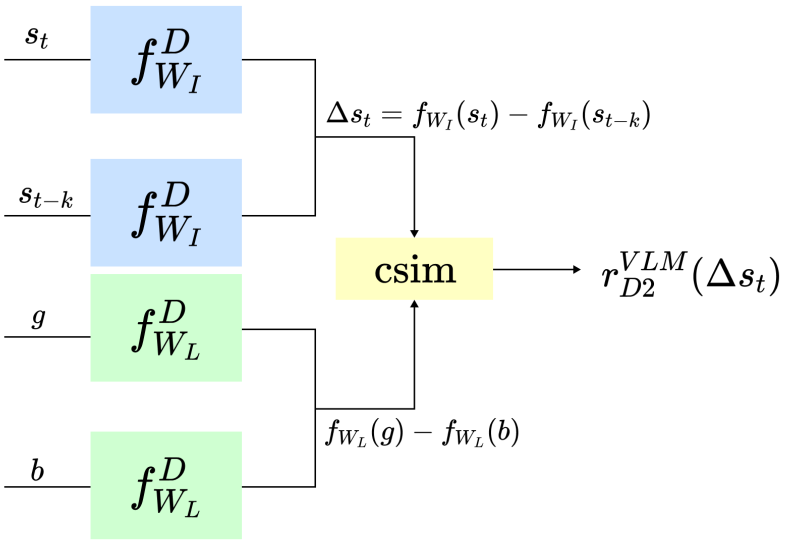


Figure 4.7: Flowchart of directional reward computation, using only directional input, where the difference is computed after $f^D_{W_I}$ and $f^D_{W_L}$, i.e. $r^{VLM}_{\text{D2}}(\Delta s_t) = \text{csim}(f^D_{W_I}(s_t) - f^D_{W_I}(s_{t-k}), f^D_{W_L}(g) - f^D_{W_L}(b))$

A potential problem with these functions is that only the direction Δs_t is provided, while the directional reward is likely also dependent on the current state/position of the environment and the agent. To explain, imagine that the environment is a simple 2D plane, where the task is for the agent to move to a specific point on the plane, and the observations s_i or s_j are potential agent positions. This example is illustrated in Figure 4.8, showing that the optimal direction is $s_g - s_i$ in s_i, and $s_g - s_j$ in s_j, where s_g is the goal state.

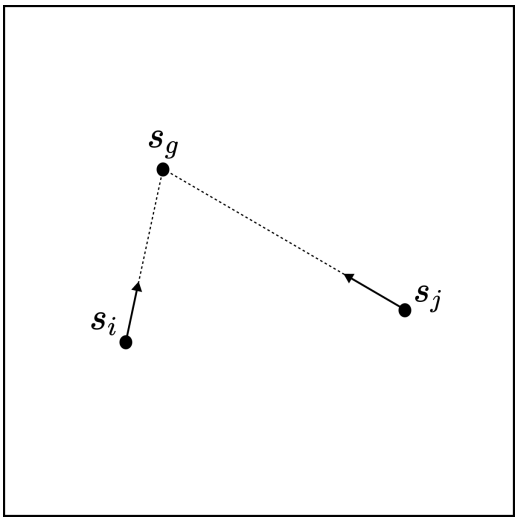


Figure 4.8: Simple 2D environment demonstrating that the optimal direction depends on the current position.

As can be seen in this example, the optimal direction is dependent on the current position s_i/s_j, which is why a third definition for the visual component can be $s_t \oplus \Delta s_t$, where the $\oplus$ operator is concatenation, so that both the position and direction is taken into consideration when computing the directional reward. With this concatenated input, the third directional reward function becomes

$$r_{\text{D3}}^{VLM}(s_t, \Delta s_t) = \text{csim}(f_{W_I}^D(s_t \oplus \Delta s_t), f_{W_L}^D(g - b))$$

which is shown in Figure 4.9.

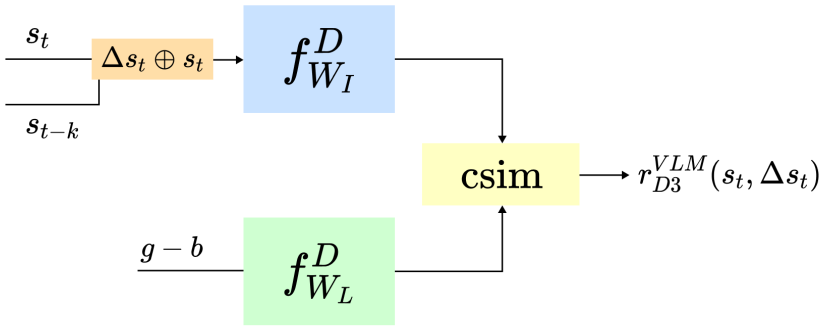


Figure 4.9: Flowchart of directional reward computation, using concatenated states with position s_t, i.e. $r_{\text{D3}}^{VLM}(s_t, \Delta s_t) = \text{csim}(f_{W_I}^D(s_t \oplus \Delta s_t), f_{W_L}^D(g - b))$

The fourth directional reward function replaces the positional half s_t of the concatenation with delta features, resulting in

$$r_{\text{D4}}^{VLM}(s_t, \Delta s_t) = \text{csim}(f_{W_I}^D((s_t - s_0) \oplus \Delta s_t), f_{W_L}^D(g - b))$$

which is shown in Figure 4.10.

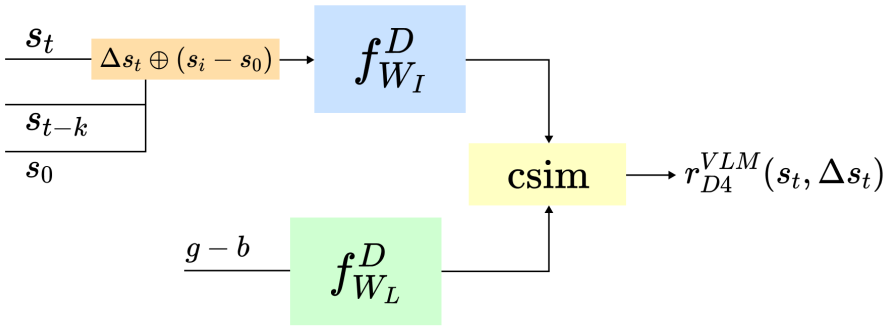


Figure 4.10: Flowchart of directional reward computation, using concatenated states with position $s_i - s_0$, i.e. $r^{VLM}_{\text{D4}}(s_t, \Delta s_t) = \text{csim}(f^D_{W_I}((s_t - s_0) \oplus \Delta s_t), f^D_{W_L}(g - b))$

The final directional reward function replaces s_t with a goal-baseline regularized state,

$$r^{VLM}_{\text{D5}}(s_t, \Delta s_t) = \text{csim}(f^D_{W_I}((\alpha \text{proj}_L s + (1 - \alpha)s) \oplus \Delta s_t), f^D_{W_L}(g - b))$$

which is shown in Figure 4.11.

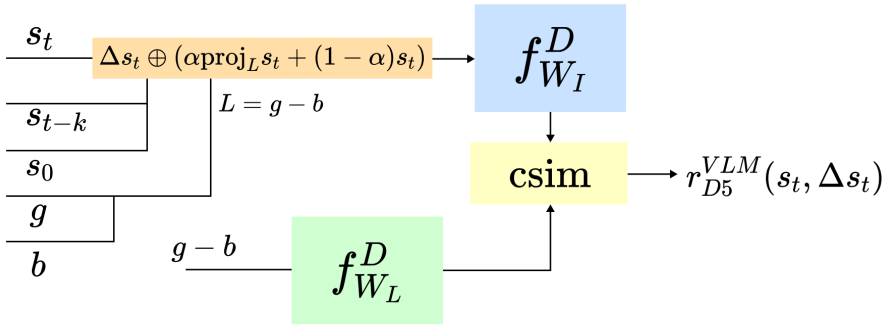


Figure 4.11: Flowchart of directional reward computation, using concatenated states with position $\alpha \text{proj}_L s + (1 - \alpha)s$, i.e. $r^{VLM}_{\text{D5}}(s_t, \Delta s_t) = \text{csim}(f^D_{W_I}((\alpha \text{proj}_L s + (1 - \alpha)s) \oplus \Delta s_t), f^D_{W_L}(g - b))$.

For concatenated inputs, there is not a formulation for which the directional embedding is computed after $f^D_{W_I}$. This is because otherwise, both the positional and directional input would have to be run through $f^D_{W_I}$, which would mean that $f^D_{W_I}$ would have to learn how to differentiate positional and directional embeddings. Or alternatively, an additional MLP head could be created for the positional input, but this would likely just increase the complexity of the model unnecessarily. Computing the subtraction and concatenation before running it through $f^D_{W_I}$ would be better, since then $f^D_{W_I}$ can easily be trained to award/penalize good/bad directions, and is given direct access to all of the information that it needs to do so.

For each of these versions of the visual component, Δs_t and the positional input are both normalized separately before being concatenated and passed to $f^D_{W_I}$. This is

necessary for the finetuning, partly because of the specific loss that is used to train it (as will be explained in Section 4.3.3), but also to avoid creating a magnitude bias, by ensuring that the position and direction are weighted equally.

4.3.3 Finetuning the directional reward

The construction of the inputs to $f^D_{W_L}$ and $f^D_{W_I}$ is important to ensure that they have access to the information necessary for producing a valuable directional reward. However, as these heads are not pretrained, a finetuning algorithm is necessary to ensure that they actually use this information.

But the desired behavior of the directional reward is not the same as the desired behavior of the positional reward. More specifically, for the positional reward, the best outcome would be for the reward to constantly increase along positive trajectories, since it is better for the agent to be positioned closer to the goal state s_g. Trajectories are made up of positions, so it makes sense for the positional reward to be dependent on the location of the position in the trajectory, as specified by the $\mathcal{L}_{pos-pos}$ loss in the FuRL algorithm.

But the directional reward should not be dependent on the corresponding position's location in a trajectory. Taking an optimal positive trajectory as an example, the directional reward should be maximal so long as the agent retraces the optimal trajectory exactly. In other words, the size of the directional reward should not be relative to the corresponding state's progression towards solving the task. Instead, for every position, the directional reward should simply be positive in the directions leading to s_g, and negative in the directions that it lead away from it.

Since directional rewards model something fundamentally different from positional rewards, that means a new finetuning objective is necessary for the directional reward functions. However, the FuRL finetuning algorithm can still be used as inspiration, particularly how it performs the finetuning in two stages, determined by whether it has access to positive trajectories.

4.3.3.1 Stage 1

The first stage is the time before any positive trajectories have been collected, so the finetuning algorithm only has access to negative trajectories. Given that the directions in the negative trajectories lead to failure, we can use these directions as negative training examples in the finetuning. In other words, the reward for these training examples should be minimized, so that the agent is discouraged from moving in those directions in the future.

However, if the finetuning algorithm only has access to directions for which the reward should minimized, then the directional reward function will simply produce the minimum possible reward each time, regardless of the input. Positive training examples are therefore also necessary to address this issue. In the first stage of the FuRL algorithm, the state for which the reward is maximized/minimized is determined by its ℓ_2 distance to the goal state, s_g. This works for the positional

reward, because the reward should generally be larger for positions that are closer to the goal. Given that s_g is already used by the positional finetuning algorithm, we can also use this to define preferred directions for the directional finetuning algorithm, pointing from s_t towards s_g.

In summary, the first stage minimizes the reward for directions $\Delta s_t^n = s_t^n - s_{t-k}^n$ and maximizes it for directions $\hat{\Delta} s_t^n = s_g - s_t^n$. Something that should be addressed here is that the magnitude of Δs_t^n will likely be far smaller than the magnitude of $\hat{\Delta} s_t^n$. This is a problem, as the directional networks might simply learn to give high rewards to directional vectors with larger magnitudes. But as mentioned previously in the final paragraph of Section 4.3.2, this can be addressed by redefining the directional reward function such that Δs_t is always normalized. Figure 4.12 illustrates the directional finetuning in the first stage.

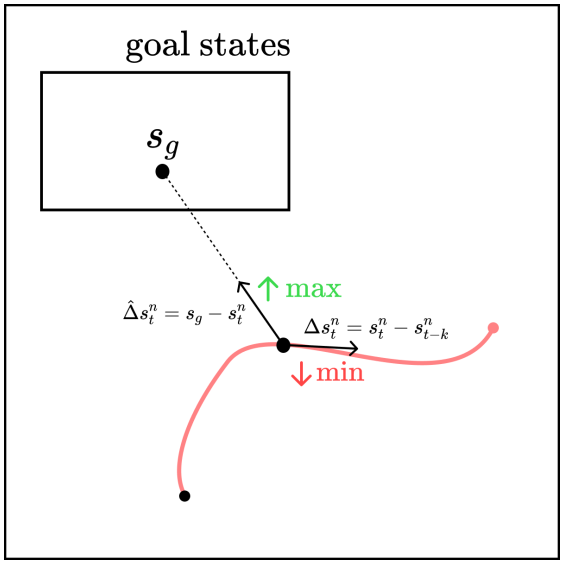


Figure 4.12: Supervised finetuning loss for the directional reward function when the algorithm only has access to negative trajectories. Directional reward is maximized for the direction $\hat{\Delta} s_t^n$ pointing towards the goal state s_g, and minimized for the direction Δs_t^n actually observed in the negative trajectory.

From this, a finetuning objective can be defined for each directional reward laid out in Section 3.2.2. When the only input to r_{D}^{VLM} is Δs_t, this objective is

$$\mathcal{L}_{neg}^{D} = r_{\text{DX}}^{VLM}(\Delta s_t^n) - r_{\text{DX}}^{VLM}(\hat{\Delta} s_t^n)$$

where X is either 1 or 2, or

$$\mathcal{L}_{neg}^{D} = r_{\text{DX}}^{VLM}(s_t^n, \Delta s_t^n) - r_{\text{DX}}^{VLM}(s_t^n, \hat{\Delta} s_t^n)$$

where X is 3, 4 or 5. Minimizing this objective will lead to a higher reward in the direction of the goal state, and a lower reward in the direction taken in the negative trajectory.

The good thing about using concatenated inputs is that the finetuning algorithm is given both negative and positive training examples in the exact same position. So if the agent ever finds itself in the same position again, then the directional reward should know which direction works and which direction does not work, given the current position.

4.3.3.2 Stage 2

With positive trajectories, an additional term $\mathcal{L}^D_{pos}$ can be added to the finetuning objective, using positive trajectory samples. In the second stage, the finetuning therefore minimizes $\mathcal{L}^D{=}\mathcal{L}^D_{neg}{+}\mathcal{L}^D_{pos}$. The preferred direction with positive samples is the direction actually taken in the trajectory, i.e. $\Delta s^p_t{=}s^p_t{-}s^p_{t-k}$. And the non-preferred direction can simply be the opposite direction, $\bar{\Delta} s^p_t{=}s^p_{t-k}{-}s^p_t$. With these directions, $\mathcal{L}^D_{neg}$ is complemented by

$$\mathcal{L}^D_{pos} = r^{VLM}_{\text{DX}}(\bar{\Delta} s^p_t) - r^{VLM}_{\text{DX}}(\Delta s^p_t)$$

when X is 1 or 2, and

$$\mathcal{L}^D_{pos} = r^{VLM}_{\text{DX}}(s^p_t, \bar{\Delta} s^p_t) - r^{VLM}_{\text{DX}}(s^p_t, \Delta s^p_t)$$

when X is 3, 4 or 5. This additional term will maximize the reward in the direction taken in the positive trajectory, and minimize the reward in the opposite direction (at the same position if the concatenated input is used). This optimization is illustrated in Figure 4.13.

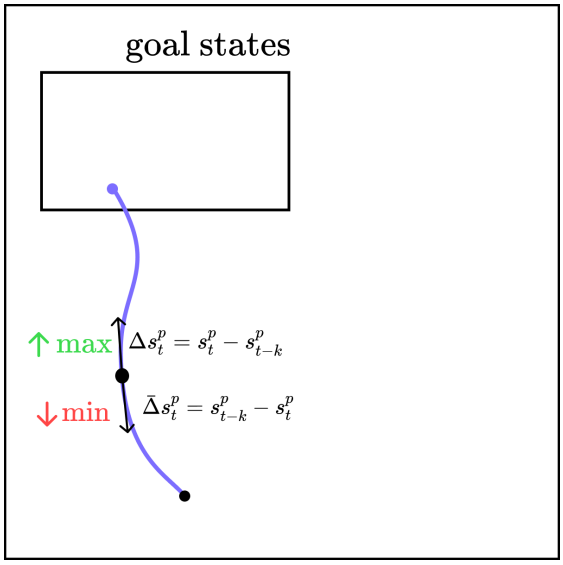


Figure 4.13: Additional loss for the directional reward function when the algorithm also has access to positive trajectories. Directional reward is maximized for direction Δs^p_t observed in the positive trajectory, and minimized in the opposite direction $\bar{\Delta} s^p_t$.

5

Results

The results in this chapter were compiled for the purpose of answering the research questions posed in the beginning of this thesis:

1. How is temporal information defined, and how can it be represented for efficient reward shaping in a sparse reward RL context?
2. How can a VLM reward function be modeled such that it takes advantage of temporal inputs, and what should its training procedure look like?
3. How can the performance of these new VLM RL methods be evaluated?
4. Given this evaluation, how do the new methods compare to existing methods?

The evaluation was performed on 6 different manipulation tasks from the Metaworld benchmark [15], shown in Figure 5.1 below. These tasks have fixed and random goal versions. In the fixed goal versions, the initial positions of all objects in the environment is the same whenever the environment is reset, while they are randomized for the random goal versions. In this thesis, the fixed goal versions were chosen for evaluation, to reduce randomness and make training faster.

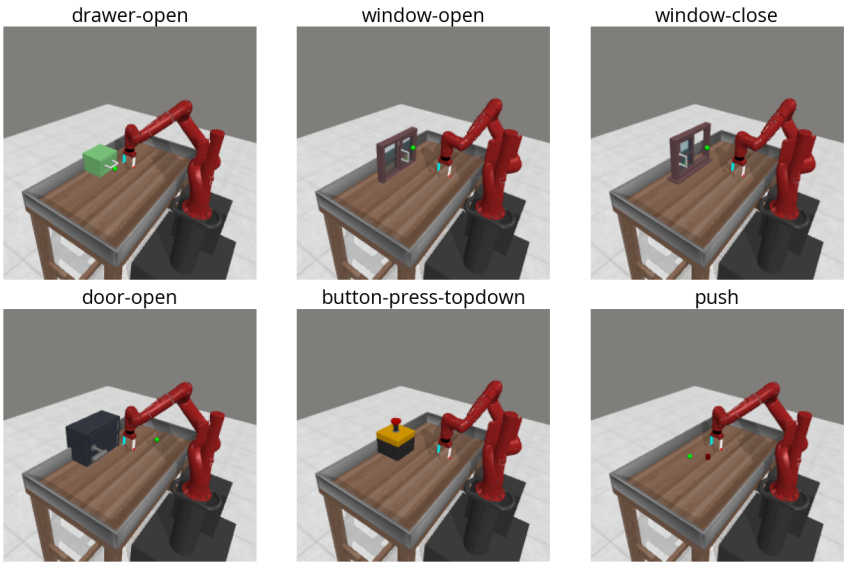


Figure 5.1: Tasks from the Metaworld benchmark, chosen based on difficulty such that they were not too easy nor too difficult for the agent to learn.

The tasks chosen were:

`drawer-open`	open a drawer by pulling it out.
`door-open`	open a door with a revolving joint.
`window-open`	open a sliding window.
`window-close`	close a sliding window.
`push`	push a puck on top of the table to a goal position.
`button-press-topdown`	push a red button from the top.

For RL, the states $\hat{s}$ will be read directly from the environment, describing the positions of the agent and the objects in the environment. In each environment step t, an action a_t is sampled from the Soft Actor-Critic (SAC) agent π_{VLM} given the current state $\hat{s}_t$, and is used to take a step in the environment, resulting in $\hat{s}_{t+1}$. Then a new image o_{t+1} is rendered of the environment which is used to produce the image embedding $s_{t+1} = \phi_I(o_{t+1})$ using the VLM image encoder ϕ_I. This embedding is then used with the goal text embedding $g = \phi_L(l_g)$ produced by the text encoder ϕ_L, potentially along with other image and text embeddings, to compute the positional and directional rewards.

Once each episode terminates, they are labeled either positive or negative and then inserted into the replay buffer from where they can later be sampled to train the RL model and finetune the reward function(s). This training occurs at set time intervals, taking a single step in both the RL optimization and the supervised finetuning of the reward function(s).

5.1 Evaluation Metrics

In Figure 5.2, the running average success rate for standard FuRL with ρ=0.05 is plotted along with the Time To First Success (TTFS) metric, which corresponds to the length of the first stage. From this figure, it is clear that the randomness in the first stage is problematic, as TTFS is a dominant factor for success rate convergence. One run fails to achieve even a single success, staying at 0% for the entire 1,000 episode duration.

With this randomness, it becomes difficult to evaluate the performance in the second stage, as the success rate at any specific point is almost entirely determined by when the first success was achieved. For this reason, the evaluation of each stage was performed separately. For the evaluation of the first stage, two metrics were used: Time To First Success (TTFS) and Time To 50 Successes (TT50S).

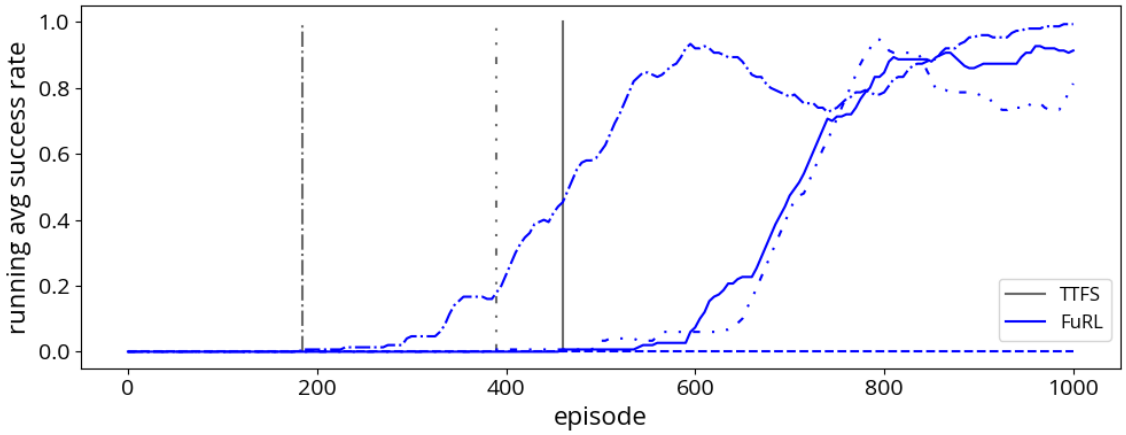


Figure 5.2: Blue lines show the running average success rate over 150 episodes training the original FuRL algorithm with ρ=0.05 (following the FuRL configuration [8]), for 4 runs in the door-open environment. Grey lines correspond to the moment the first success is achieved, i.e. Time To First Success (TTFS).

Let $z_i \in \{0, 1\}$ indicate the success in episode i. Then TTFS is

$$\text{TTFS} = \min\{i \mid z_i = 1\}$$

and TT50S is

$$\text{TT50S} = \min\left\{i \;\middle|\; \sum_{j=1}^{i} z_j = 50\right\}$$

Something that these metrics do not account for is when no success is ever achieved, or when there are less than 50 successes in the TT50S case. In these scenarios, TTFS and TT50S are simply evaluated to be 1,000 episodes, since that is the time horizon for each run.

To evaluate the rewards in the second stage, the first stage was skipped by preloading 50 positive trajectories into the replay buffer before the training even began. The reason for choosing 50 positive trajectories rather than just 1 is that some trajectories are more likely to be useful for learning the task than others. Taking the door-open task as an example, if the robot arm spins around in a circle first before opening the door, then the corresponding positive trajectory would not be as useful as a trajectory where the arm simply reaches towards the door and opens it right away.

If the one successful trajectory happens to be of the former sort, then the agent will start off by having to explore more until it collects a successful trajectory of higher quality. To ensure that the algorithm has access to higher-quality trajectories from the beginning, the replay buffer is therefore preloaded with more than 1 successful trajectory, in this case 50. This preloading of 50 positive trajectories is also the reason why the first stage evaluation includes TT50S, to assess the performance up to the point where the second stage evaluation would begin.

With this setup, the second stage was evaluated using the running average success rate over a window size w (number of episodes), with a time step budget e determined

by task difficulty. The motivation for using a variable time step budget is that with this evaluation setup, the success rate almost always managed to converge to 100% in the 1,000 episode time frame, regardless of which reward shaping method was used. For this reason it would be better to evaluate the success rate at an earlier time step e, which is then determined by the difficulty of the given task, since the difficulty determines the convergence rate.

To evaluate the second stage, the Running Average Success (RAS) metric was developed, formally

$$\mathrm{RAS}(e; w) = \frac{1}{w} \sum_{i=e-w+1}^{e} z_i$$

For all second stage evaluation in this thesis, the window size was $w = 150$, and the time step budget for each task can be found in Table 5.1.

Table 5.1: Episode budget for second stage RAS evaluation, for each Metaworld task used in evaluation. Budget was determined by task difficulty.

Task	Budget e
drawer-open	300
door-open	400
window-open	200
window-close	200
push	1,000
button-press-topdown	400

To gain an idea of task difficulty, please refer to Figure 5.3, which shows the optimal policy in the door-open environment.

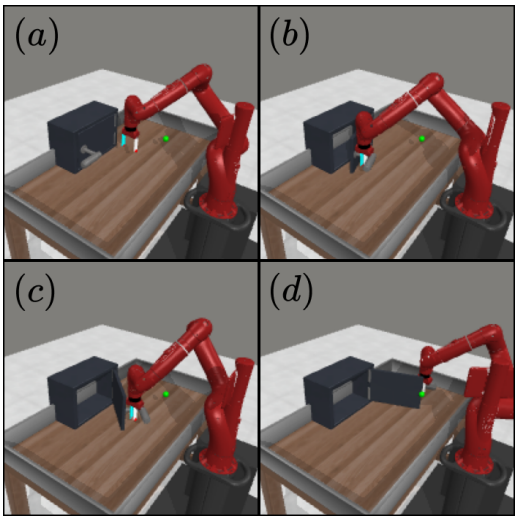


Figure 5.3: Optimal policy for the door-open task. (a) initial positions, door is closed; (b) agent has grasped the handle and is opening the door; (c) door is opening, task success imminent; (d) door is open, task success.

5.2 The First Stage

RAS was chosen as the main metric to evaluate the methods, so the first stage was only evaluated for the positional and directional rewards that achieved the best RAS score. The chosen reward functions were r_{P2}^{VLM} (delta features, difference computed after MLP heads), r_{P3}^{VLM} (goal-baseline regularization performed after MLP heads), and r_{D3}^{VLM} (concatenated input with position s_t). The evaluation was performed with ρ=0.05, following the optimal configuration found in the FuRL paper [8], and λ=80 for r_{D3}^{VLM}, as this resulted in optimal RAS. The positional reward when evaluating r_{D3}^{VLM} was standard FuRL. For a baseline comparison, FuRL with ρ=0.05 was also included. The results with this setup can be found in Table 5.2. TT50S performance with the same configuration can then be found in Table 5.3. These results again demonstrate that there is a lot of randomness in the amount of time spent in initial exploration, before the first success is achieved.

Table 5.2: Time To First Success (TTFS) for 4 different shaping rewards. Evaluated by mean TTFS over 4 runs (4 different seeds) for each task/reward pair, and standard deviation. Then the average over all task means is displayed in the bottom row. Configuration: ρ=0.05 (following FuRL paper [8]), and λ=80 for directional rewards. Best performance is illustrated by bold font. Grey background indicates baseline comparison; white background indicates contribution made by this thesis.

Task	FuRL	r_{P2}^{VLM}	r_{P3}^{VLM}	FuRL + r_{D3}^{VLM}
drawer-open	379 ± 214	389 ± 353	**205** ± 57	271 ± 29
door-open	**420** ± 335	591 ± 286	478 ± 245	556 ± 324
window-open	**171** ± 129	178 ± 75	395 ± 350	335 ± 77
window-close	**214** ± 23	592 ± 408	238 ± 58	484 ± 317
push	349 ± 172	285 ± 101	**189** ± 95	284 ± 103
button-press-topdown	559 ± 268	626 ± 376	**399** ± 75	856 ± 249
Average	349	444	**317**	464

Table 5.3: TT50S with the same configuration as in Table 5.2. Best performance is illustrated by bold font. Grey background indicates baseline comparison; white background indicates contribution made by this thesis.

Task	FuRL	r_{P2}^{VLM}	r_{P3}^{VLM}	FuRL + r_{D3}^{VLM}
drawer-open	710 ± 275	570 ± 255	**439** ± 34	732 ± 205
door-open	**608** ± 238	766 ± 241	665 ± 197	691 ± 245
window-open	**375** ± 102	405 ± 192	538 ± 271	591 ± 110
window-close	520 ± 138	655 ± 349	**456** ± 201	704 ± 200
push	880 ± 208	931 ± 119	1000 ± 0	**765** ± 183
button-press-topdown	709 ± 220	759 ± 259	**620** ± 36	986 ± 24
Average	634	681	**620**	745

5.3 The Second Stage

The new rewards were evaluated mainly by their performance in the second stage, with 50 preloaded positive trajectories. The metric used was RAS, as defined in Section 5.1. Baselines included were standard Soft Actor-Critic (SAC, no reward shaping), and the original FuRL reward with no temporal information.

5.3.1 The Positional Reward

The positional rewards were evaluated with ρ=0.05, and compared to two baseline methods: FuRL baseline (also with ρ=0.05), and standard SAC (no reward shaping). The running average success rate can be examined for a qualitative analysis in Figure 5.4, in the door-open environment.

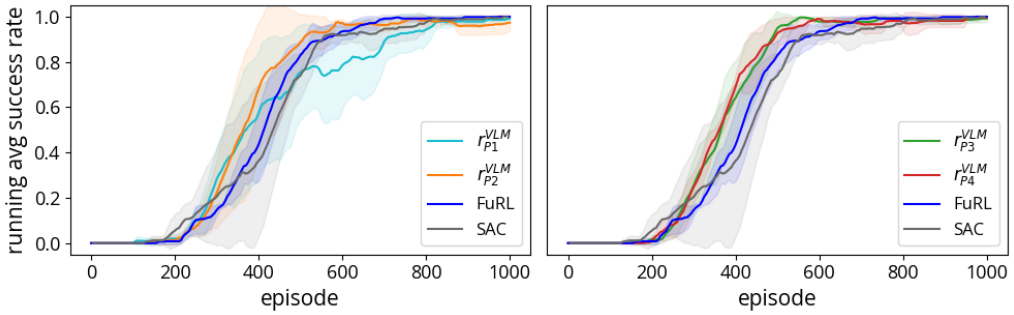


Figure 5.4: Running average success rate and standard deviation over 150 training episodes for different positional rewards in the door-open environment. For goal-baseline regularization, α=0.4.

For a more quantitative analysis, Table 5.4 contains RAS scores for all positional rewards and all tasks.

Table 5.4: Average Running Average Success (RAS) scores (in %) over 4 runs (4 different seeds) for each reward/task pair, using the second stage evaluation setup (50 preloaded positive trajectories). Configured with ρ=0.05, following [8]. For goal-baseline regularization, α=0.4. Best performance is illustrated by bold font. Grey background indicates baseline comparison; white background indicates contribution made by this thesis.

Task	SAC	FuRL	r^{VLM}_{P1}	r^{VLM}_{P2}	r^{VLM}_{P3}	r^{VLM}_{P4}
drawer-open	**53.5**	41.7	52.2	52.0	41.7	41.5
door-open	35.2	41.5	55.0	65.7	**66.5**	63.0
window-open	39.7	47.7	46.7	**51.8**	46.2	50.8
window-close	32.8	24.5	23.3	42.2	**42.8**	19.5
push	28.0	35.0	**36.3**	26.7	28.2	33.5
button-press-topdown	56.2	76.8	68.2	**77.7**	68.8	73.8
Average	40.9	44.5	46.9	**52.7**	49.0	47.0

5.3.1.1 Positional Baseline

One central question in the new positional rewards is how to choose the baseline, and what effect the baseline has on performance (if any). In this thesis, two types of baselines were evaluated.

The first was a general baseline that is applicable in all tasks, which was simply "robot arm beside table," since this describes the non-task-specific elements of the environment for all 6 tasks. This was the baseline used to produce the results seen previously in Figure 5.4 and Table 5.4.

The second type of baseline was inspired by the baselines used in the delta features paper [6]. In this paper, they define the baseline as the task when unsolved. For instance, if the task was to "open microwave," then the baseline might be "closed microwave." The baselines of the second type were therefore tailored to the specific task in question, such as "closed drawer" for the open-drawer task. The performance of this type of baseline, i.e. a baseline tailored for the specific task, can be seen in Figure 5.5 and Table 5.5.

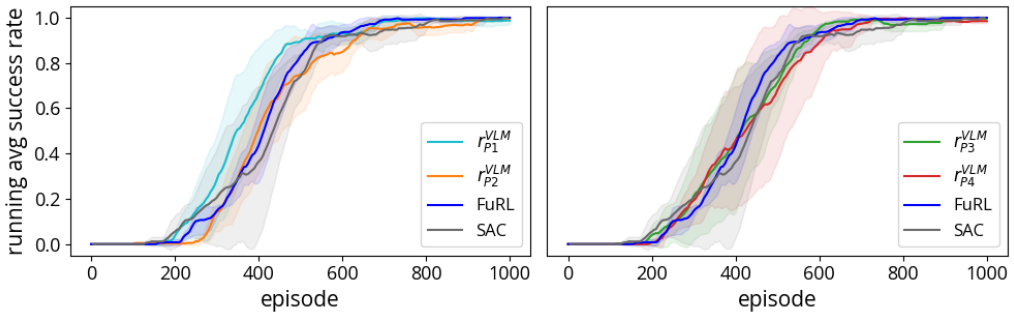


Figure 5.5: Same setup as in Figure 5.4, but with a textual baseline that is tailored to the given task, e.g. "closed door" for the door-open task.

Table 5.5: Average RAS scores (in %), with the same setup and configuration as in Table 5.4, but with a textual baseline that is tailored to the given task, e.g. "closed door" for the door-open task. Best performance is illustrated by bold font. Grey background indicates baseline comparison; white background indicates contribution made by this thesis.

Task	SAC	FuRL	r_{P1}^{VLM}	r_{P2}^{VLM}	r_{P3}^{VLM}	r_{P4}^{VLM}
drawer-open	**53.5**	41.7	51.3	32.7	34.0	41.3
door-open	35.2	41.5	**64.3**	48.5	44.2	43.0
window-open	39.7	47.7	46.0	47.8	**53.8**	48.3
window-close	32.8	24.5	31.2	**34.7**	33.3	26.2
push	28.0	**35.0**	23.5	31.7	29.7	**35.0**
button-press-topdown	56.2	**76.8**	70.8	75.5	56.7	61.7
Average	40.9	44.5	**47.9**	45.1	41.9	42.6

5.3.1.2 Tuning Positional Reward Size

In the upcoming evaluation of the directional rewards, the λ parameter was tuned to optimize the RAS score. To make the comparison to FuRL more fair, the ρ parameter was also tuned with RAS for the FuRL baseline. The results from tuning ρ for optimal RAS can be found in Table 5.6. These results deviate from the FuRL paper [8], which found ρ=0.05 to be optimal.

Table 5.6: Average RAS scores (in %), with the same setup and configuration as in Table 5.4, for original FuRL with varying ρ. Best performance is illustrated by bold font.

Task	$\rho = 0.05$	$\rho = 0.25$	$\rho = 1.0$	$\rho = 2.0$	$\rho = 4.0$
drawer-open	41.7	**60.0**	56.2	55.3	53.5
door-open	41.5	60.2	83.0	85.0	**87.7**
window-open	47.7	**55.5**	54.8	54.3	49.2
window-close	24.5	49.0	42.7	52.3	**52.5**
push	35.0	43.7	**56.5**	48.8	40.3
button-press-topdown	76.8	83.3	**89.3**	86.3	88.8
Average	44.5	58.6	**63.8**	63.7	62.0

This discrepancy could potentially come from the fact that only the second stage was evaluated to tune ρ in this thesis. With preloaded positive trajectories, more data is available for finetuning, making the reward function more accurate, which could mean that the agent can handle a larger VLM reward. Another potential cause could be that fixed goals were used in this thesis, while the FuRL paper [8] tuned ρ with random goal tasks. The increased randomness from random goals could have contributed to instability in the VLM reward, making the optimal ρ lower.

These results also indicate that the optimal VLM reward size may be task-dependent, as a very high positional reward seems to be best for e.g. door-open, while a much smaller value is better for e.g. drawer-open. For a more qualitative analysis, the running average success rate for each of these tasks can be seen in Figures 5.6 and 5.7, which demonstrate a clear task-dependency in the optimal VLM reward size.

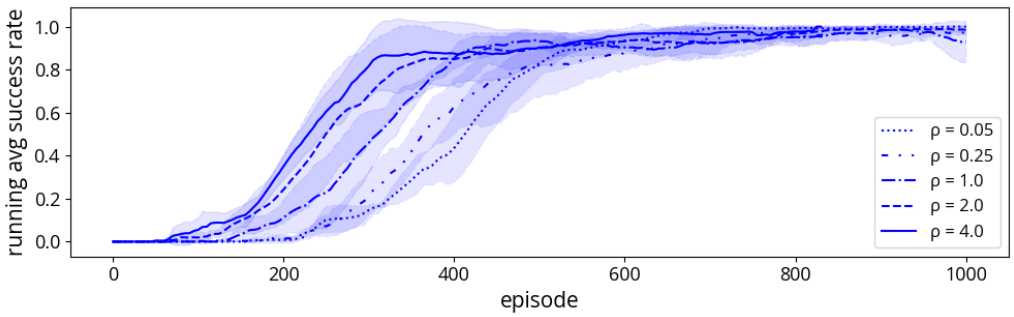


Figure 5.6: Running average success rate for original FuRL positional reward over 4 runs, with standard deviation, for the door-open task, with varying ρ.

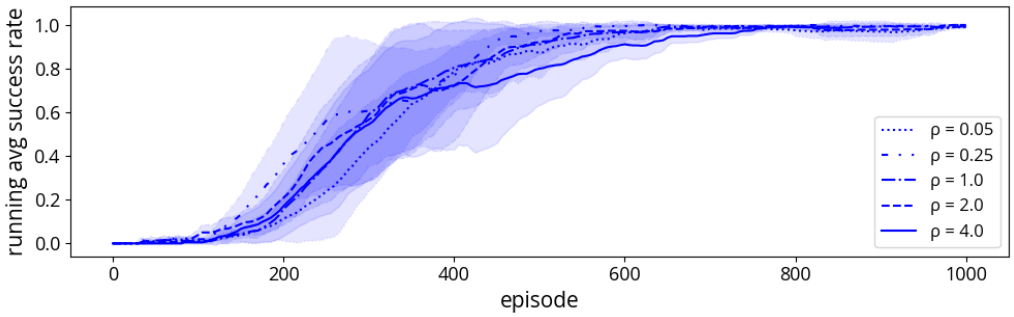


Figure 5.7: Running average success rate for original FuRL positional reward over 4 runs, with standard deviation, for the drawer-open task, with varying ρ.

5.3.2 The Directional Reward

Out of r_{D1}^{VLM} and r_{D2}^{VLM}, only r_{D2}^{VLM} was evaluated. This was because the optimal configuration for both delta features and goal-baseline regularization in positional rewards included computing the subtraction or regularization after the MLP heads, which also applies to r_{D2}^{VLM} but not r_{D1}^{VLM}.

Again following the configuration used in the FuRL paper [8], the directional rewards were evaluated with ρ=0.05, where the positional reward was standard FuRL (no temporal information). The RAS scores for r_{D2}^{VLM} can be found in Table 5.7. Relatively high λ values were necessary to compensate for the low ρ.

Table 5.7: Average RAS scores (in %) for r_{D2}^{VLM} with ρ=0.05 and varying λ over 4 runs (4 different seeds) per reward/task pair. Best performance is illustrated by bold font. SAC (no reward shaping) and FuRL with optimal ρ=1.0 were included for comparison. Grey background indicates baseline comparison; white background indicates contribution made by this thesis.

Task	SAC	FuRL	$\lambda = 30$	$\lambda = 80$	$\lambda = 150$
drawer-open	53.5	56.2	30.0	51.0	**62.7**
door-open	35.2	83.0	**90.3**	84.2	80.7
window-open	39.7	54.8	**68.5**	65.5	61.0
window-close	32.8	42.7	**59.3**	58.8	54.8
push	28.0	**56.5**	39.2	49.7	19.3
button-press-topdown	56.2	**89.3**	80.2	45.7	55.8
Average	40.9	**63.8**	61.2	59.1	55.7

This shows that while the directional reward computed only with the direction is better than using no reward shaping, positional shaping still achieves a better RAS score. However, it should be noted that drawer-open is a major outlier in this case. The running average success rate for drawer-open can be seen in Figure 5.8, which can be compared to the door-open success rate in Figure 5.9.

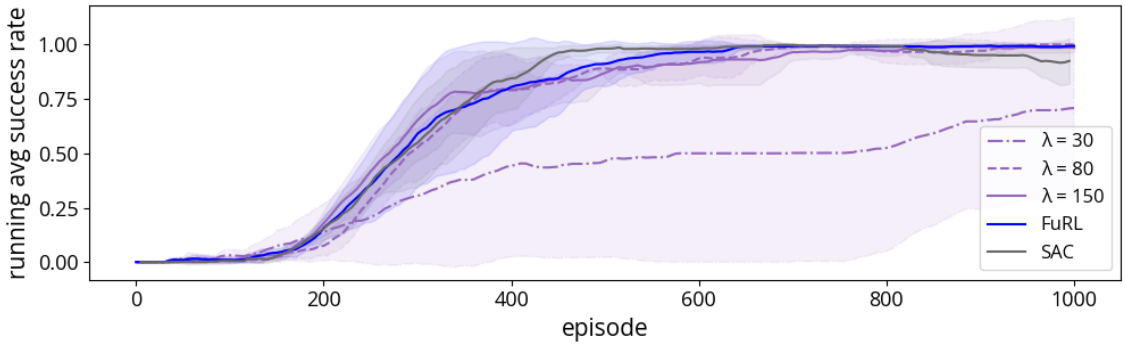


Figure 5.8: Running average success rate for r_{D2}^{VLM} over 4 runs (4 different seeds) for the drawer-open task, with ρ=0.05 and varying λ. Baselines SAC (no reward shaping) and FuRL with optimal ρ=1.0 were also included.

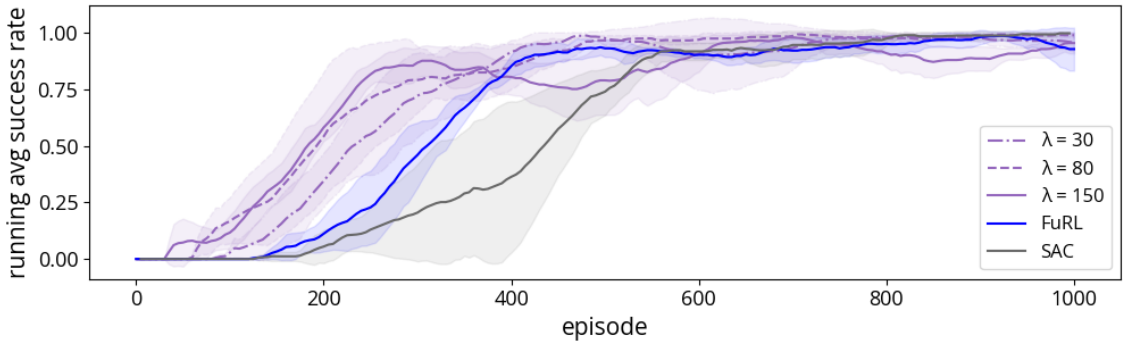


Figure 5.9: Running average success rate for r_{D2}^{VLM} over 4 runs (4 different seeds) for the door-open task, with ρ=0.05 and varying λ. Baselines SAC (no reward shaping) and FuRL with optimal ρ=1.0 were also included.

As can be seen in Figure 5.8, λ=30 in drawer-open is a significant outlier, as some of the runs achieve similar performance to other λ, while others fail completely, and the same pattern is not observed for the other λ. However, it is curious that VLM reward shaping actually does not seem to be beneficial in drawer-open in general, as SAC achieves essentially the same performance as even the optimal FuRL. Figure 5.9 is more aligned with the expected behavior, as SAC performs the worst, then FuRL, and then finally using both FuRL and the r_{D2}^{VLM} directional reward is the most efficient. This again suggests a task-dependency in the effectiveness of VLM reward shaping.

Figure 5.9 also shows that higher λ tend to experience faster convergence, but are more unstable once convergence is reached. This makes sense, because a higher λ means that the the agent will focus more on maximizing the noisy r_{D}^{VLM}, and risk ignoring r^{task}.

Moving onto directional rewards computed with both direction and position, the results with position s_t (i.e. r_{D3}^{VLM}) can be found in Table 5.8.

Table 5.8: Average RAS scores (in %) for r_{D3}^{VLM} with ρ=0.05 and different λ over 4 runs (4 different seeds) per reward/task pair. Best performance is illustrated by bold font. SAC (no reward shaping) and FuRL with optimal ρ=1.0 were included for comparison. Grey background indicates baseline comparison; white background indicates contribution made by this thesis.

Task	SAC	FuRL	$\lambda = 30$	$\lambda = 80$	$\lambda = 150$
drawer-open	53.5	56.2	**97.8**	87.0	88.5
door-open	35.2	83.0	93.2	**96.2**	81.8
window-open	39.7	54.8	76.0	73.0	**77.5**
window-close	32.8	42.7	**77.7**	76.2	65.3
push	28.0	**56.5**	43.5	46.7	51.2
button-press-topdown	56.2	89.3	76.5	**90.3**	78.2
Average	40.9	63.8	77.4	**78.2**	73.8

Here a clear improvement can be seen over both FuRL and r_{D2}^{VLM}. And when the position is included in the directional reward computation, the previous pattern of poor performance in drawer-open is no longer observed, as can be seen in Figure 5.10. This suggests that drawer-open is not appropriate for positional shaping, but may be more appropriate for directional shaping. This suggests that it is not only VLM reward shaping in general that is task-dependent, but that the effectiveness of positional vs. directional shaping is also task-dependent.

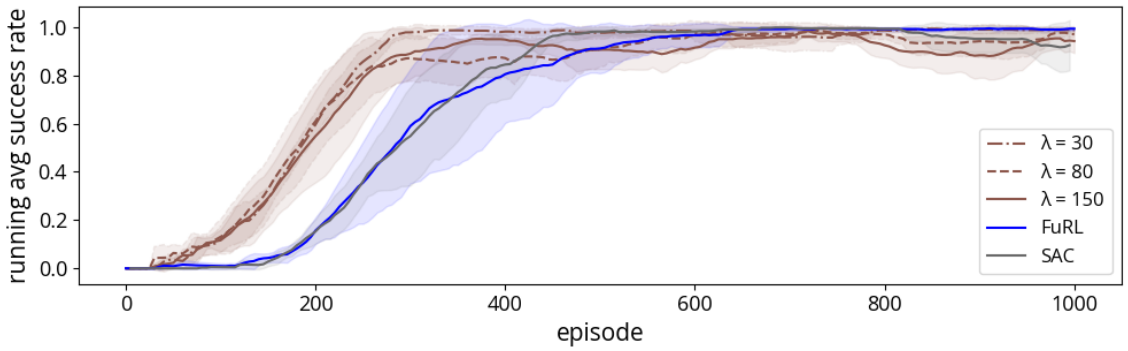


Figure 5.10: Running average success rate for r_{D3}^{VLM} over 4 runs (4 different seeds) for the drawer-open task, with ρ=0.05 and varying λ. Baselines SAC (no reward shaping) and FuRL with optimal ρ=1.0 were also included.

The task-dependency of positional vs. directional shaping is further demonstrated by Figure 5.11, where the success rate for the button-press-topdown task is plotted. For this task, positional shaping with FuRL beats SAC, but the difference between positional and directional shaping is not as pronounced as in Figure 5.10.

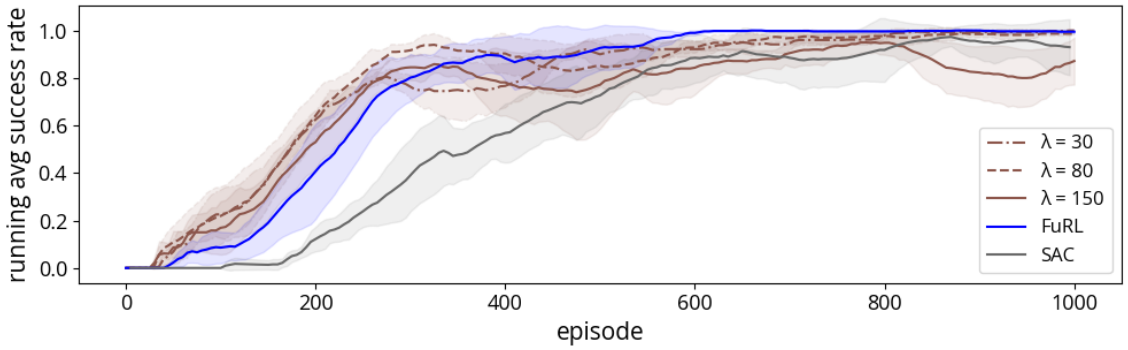


Figure 5.11: Running average success rate for r_{D3}^{VLM} over 4 runs (4 different seeds) for the button-press-topdown task, with ρ=0.05 and varying λ. Baselines SAC (no reward shaping) and FuRL with optimal ρ=1.0 were also included.

The position s_t in the concatenation can be replaced by $s_t - s_0$ or $\alpha \text{proj}_L s_i + (1-\alpha)s_i$ for r_{D4}^{VLM} and r_{D5}^{VLM}. The RAS scores for r_{D4}^{VLM} can be seen in Table 5.9, and the RAS scores for r_{D5}^{VLM} can be seen in Table 5.10.

Table 5.9: Average RAS scores (in %) for r_{D4}^{VLM} with ρ=0.05 and different λs over 4 runs (4 different seeds) per reward/task pair. Best performance is illustrated by bold font. SAC (no reward shaping) and FuRL with optimal ρ=1.0 were included for comparison. Grey background indicates baseline comparison; white background indicates contribution made by this thesis.

Task	SAC	FuRL	$\lambda = 30$	$\lambda = 80$	$\lambda = 150$
drawer-open	53.5	56.2	**89.7**	82.5	75.7
door-open	35.2	83.0	82.5	**88.7**	82.7
window-open	39.7	54.8	**82.5**	81.3	74.5
window-close	32.8	42.7	**78.7**	70.7	77.7
push	28.0	**56.5**	51.2	44.5	44.0
button-press-topdown	56.2	**89.3**	76.0	85.5	84.8
Average	40.9	63.8	**76.8**	75.5	73.2

Table 5.10: Average RAS scores (in %) for r_{D5}^{VLM} with ρ=0.05 and different λs over 4 runs (4 different seeds) per reward/task pair. Best performance is illustrated by bold font. For goal-baseline regularization, α=0.4. SAC (no reward shaping) and FuRL with optimal ρ=1.0 were included for comparison. Grey background indicates baseline comparison; white background indicates contribution made by this thesis.

Task	SAC	FuRL	$\lambda = 30$	$\lambda = 80$	$\lambda = 150$
drawer-open	53.5	56.2	87.0	77.7	**87.8**
door-open	35.2	83.0	**95.2**	90.8	84.2
window-open	39.7	54.8	**76.8**	75.0	75.5
window-close	32.8	42.7	75.0	**80.2**	79.0
push	28.0	56.5	43.2	**61.2**	44.8
button-press-topdown	56.2	89.3	**94.0**	84.3	72.2
Average	40.9	63.8	**78.5**	78.2	73.9

Again, directional shaping consistently outperforms positional shaping, with a goal-baseline regularized position beating FuRL with optimal ρ=1.0 in every single task. Goal-baseline regularization achieves a better RAS score than when the position is s_t, but it is not significantly better, so it is likely better to just go with s_t to avoid the extra computation.

5.3.2.1 Directional Baseline

As with the positional reward, the effect of the baseline on the directional reward can be investigated. Again, two baselines were compared, but only for $\lambda = 80$ for each algorithm. Only a single λ was evaluated because the effect of the baseline should be the same regardless of the size of the directional reward. The results of using a generic versus tailored baseline can be seen in Table 5.11. No clear pattern can be seen for these results, which suggests that the baseline is less important for the directional reward.

Table 5.11: Average RAS scores (in %) for r_{D3}^{VLM}, r_{D4}^{VLM} and r_{D5}^{VLM} with ρ=0.05 and λ=80 over 4 runs (4 different seeds) per reward/task pair. Baseline is either generic b_g or tailored b_t. Previous results in Section 5.3.2 used baseline b_t. Best performance (for the same reward function but different baseline) is illustrated by bold font. SAC (no reward shaping) and FuRL with optimal ρ=1.0 were included for comparison. Grey background indicates baseline comparison; white background indicates contribution made by this thesis.

Task	r_{D3}^{VLM}, b_g	r_{D3}^{VLM}, b_t	r_{D4}^{VLM}, b_g	r_{D4}^{VLM}, b_t	r_{D5}^{VLM}, b_g	r_{D5}^{VLM}, b_t
drawer-open	**89.1**%	87.0	74.8%	**82.5**	**89.0**%	77.7
door-open	94.2%	**96.2**	**91.7**%	88.7	82.7%	**90.8**
window-open	**78.2**%	73.0	74.2%	**81.3**	**75.8**%	75.0
window-close	71.8%	**76.2**	**77.2**%	70.7	75.8%	**80.2**
push	**60.3**%	46.7	**48.7**%	44.5	54.2%	**61.2**
button-press-topdown	81.3%	**90.3**	**86.2**%	85.5	83.7%	**84.3**
Average	**79.2**%	78.2	75.4%	**75.5**	76.9%	**78.2**

Choosing a generic or tailored baseline does not have much of an effect, but the greatest performance so far can be seen with r_{D3}^{VLM} and a generic baseline b_g.

5.3.3 Combining Positional and Directional Rewards

Table 5.12: Average RAS scores (in %) for fixed directional r_{D3}^{VLM} reward magnitude ($\rho \cdot \lambda$=4.0) and varying positional reward magnitude over 4 runs (4 different seeds) per reward/task pair. Baseline used is generic b_g. Best performance is illustrated by bold font. SAC (no reward shaping) and FuRL with optimal ρ=1.0 were included for comparison. Grey background indicates baseline comparison; white background indicates contribution made by this thesis.

Task	SAC	FuRL	w/o r_{P}^{VLM}	$\rho = 0.05$	$\rho = 1.0$	$\rho = 2.0$
drawer-open	53.5	56.2	77.7	89.1	**91.8**	83.7
door-open	35.2	83.0	96.0	94.2	**96.2**	95.0
window-open	39.7	54.8	**81.2**	78.2	74.5	78.7
window-close	32.8	42.7	76.5	71.8	**78.5**	76.2
push	28.0	56.5	51.3	**60.3**	54.2	48.0
button-press-topdown	56.2	**89.3**	85.7	81.3	84.7	86.2
Average	40.9	63.8	78.1	79.2	**80.0**	77.9

The previous evaluation of the directional rewards included a very small positional reward. But with ρ=0.05 and λ values 30, 80 and 150, it likely does not have a big effect on performance, since the size of the positional reward is much smaller

than the directional reward. Table 5.12 shows RAS scores for fixed directional r_{D3}^{VLM} reward magnitude $\rho \cdot \lambda$=0.05 · 80=4.0 but with varying positional reward magnitude (standard FuRL). Evaluated configurations were (ρ=0.05, λ=80.0) without r_{P}^{VLM}, (ρ=0.05, λ=80.0) with r_{P}^{VLM} (from previous results), (ρ=1.0, λ=4.0) and (ρ=2.0, λ=2.0).

These results show that optimal performance using only directional rewards (no positional signal) is far better than optimal performance using only positional rewards (FuRL vs. w/o r_{P}^{VLM}). However, adding a small positional reward alongside the larger directional reward (ρ=1.0, λ=4) gives a small uptick in RAS of 1.9 percentage points to 80.0%, which is the best score achieved by any method this thesis.

5.3.4 Reward Curve Shapes

Once the positional and directional heads have been trained such that the success rate has converged, the shape of the reward curve along a trajectory can be examined. The positional and directional rewards along a positive trajectory (blue) and a negative trajectory (orange) from the door-open environment can be seen in Figure 5.12.

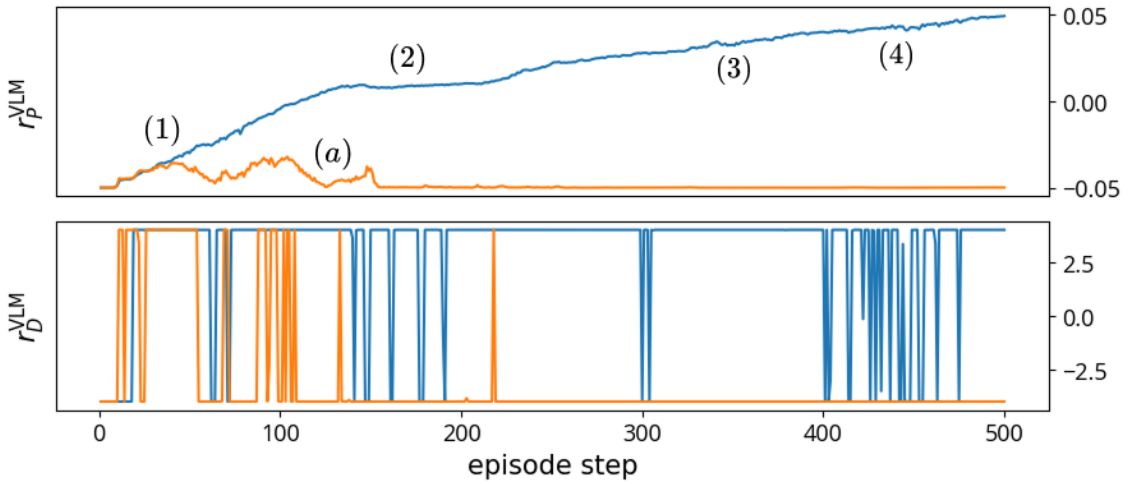


Figure 5.12: Positive (blue) and negative (orange) trajectory reward curves, for original FuRL positional reward and r_{D3}^{VLM} directional reward, with ρ=0.05 and λ=80.0. Numerical markers indicate events in positive trajectory, alphabetical markers in negative trajectory. (1) handle is grasped and door begins opening; (2) brief struggle (reflected in directional signal); (3) success; (4) slight backward shift; (a) agent reaches for and touches the handle, but then leaves and never returns.

The positional reward is monotone (strictly increasing) in the positive trajectory, which is desired. The directional reward behaves like an indicator function, producing only the maximum and minimum possible reward. This behavior can be interpreted as a yes/no signal of whether the observed change is beneficial. From the markers in Figure 5.12, the positional reward is useful as a global indicator of

progress, while the directional reward is useful for assessing local change, in specific moments in time. Non-beneficial change in the otherwise positive trajectory is reflected by a negative directional reward, while such changes are harder to interpret using only the positional reward signal, as they lead to a plateau at best, and no change at worst.

6

Conclusion

To conclude this thesis, the results from Chapter 5 will be used to provide an to answer the research questions posed in Section 3.2, and potential directions for future research will be suggested.

This chapter will begin with the third and fourth research questions, and then conclude with the first and second.

6.1 Reward Evaluation

How can the performance of these new VLM RL methods be evaluated?

The main difficulty in evaluating these rewards was how to address the randomness present in each run. This was partially addressed by assessing the first and the second stage separately, so that the randomness present in the first stage will not affect the evaluation of the second stage. One downside of this method is that it does not measure the effect that the first stage has on convergence. This could be assessed by measuring the convergence time from the start of the second stage, plotted over TTFS, to see if a longer first stage makes the convergence faster. If it does, then that would indicate that the finetuning in the first stage is effective for the purpose of solving the task.

For the first stage evaluation, the results shown in Section 5.2 are not particularly informative, as the amount of randomness reduces the confidence in the evaluated performance of the given reward. With a TTFS standard deviation often in the 100's, only 4 runs per reward and task is not enough for a proper evaluation of reward's effect on first stage exploration. Future research could therefore do a more thorough investigation into the effectiveness of temporal methods in first stage exploration, for example by simply running it more times for each task and method.

Evaluating the new methods with random goal tasks is also something that should be explored in the future. This will make the convergence slower, but the success rate does not need to converge to near 100% for the method to be properly evaluated. And if the goal is not fixed, then a specific iteration of the model parameters can be evaluated with multiple episodes, as initial positions will be randomized when the environment is reset. This would be a more interpretable evaluation than a running average success rate metric like RAS.

Additionally, since the success rate converges to near 100% for every algorithm, it becomes difficult to estimate what the algorithm is *capable of* rather than *how fast* it can do it. By making the environment more complex, such as by making the goal variable rather than fixed, it could be that it actually becomes impossible for some of the methods to consistently solve the task.

For instance, it could be that directional rewards speed up the learning in simple tasks, because the directional reward can be used to easily retrace previous trajectories. But if the environment is more complex, for example if some vital part of the state is obscured in the image observations, then simply retracing previous (image embedding) trajectories would not work unless the agent utilizes some other method to make inferences about the hidden state. However, this is a general problem with all temporally local rewards (i.e. evaluated at a specific moment in time), so it would a problem for positional shaping as well.

Finally, the evaluation can also be complemented with more qualitative methods, such as an evaluation of how the reward curves are shaped over time. With a dataset of positive trajectories, the rewards for each of their states can be computed at set intervals as the reward functions are finetuned over time. This would show how fast the algorithm learns to produce desirable reward curves, and can be used to evaluate the supervised reward finetuning in both stages of the algorithm.

6.2 Reward Performance

Given this evaluation, how do the new methods compare to existing methods?

6.2.1 The First Stage

To evaluate each algorithm in its first stage, the number of episodes spent in the first stage were counted (TTFS), along with how many episodes it took to collect 50 positive trajectories (TT50S). From the results in Section 5.2, using goal-baseline regularization seems to be slightly better than the rest at exploring and achieving the first success, for both of the metrics used.

However, there is quite a lot of randomness in these results, as the standard deviation for 4 runs is often above 100 episodes for both TTFS and TT50S. With only 4 samples for each task and reward, it is therefore hard to make a determination as to which reward is better with any degree of confidence. This could be amended by simply running it more times, but this will be left to future research, as the focus of this thesis was more on the second stage.

6.2.2 The Second Stage: Positional Rewards

For the positional reward algorithms, the best performance in terms of sample efficiency was achieved with delta features, where the subtraction was computed after $f^P_{W_I}$ and $f^P_{W_L}$ (i.e. r^{VLM}_{P2}). With the same ρ=0.05, r^{VLM}_{P2} achieved **18.4%** higher RAS over FuRL (**52.7%** vs. **44.5%**), and a **28.9%** improvement over standard

SAC with no reward shaping (**52.7%** vs. **40.9%**). Similarly small improvements were observed for the other fixed temporal methods. And for goal-baseline regularization, it again seems to be preferable to feed the embeddings directly into $f^P_{W_I}$ and $f^P_{W_L}$ before performing regularization, as in r^{VLM}_{P3}.

A potential explanation for this pattern is that delta features and goal-baseline regularization are fundamentally destructive operations, in that they remove information from the inputs, either through subtraction or through projection. By running the embeddings through $f^P_{W_I}$ and $f^P_{W_L}$ first, these heads can transform the embeddings such that only unnecessary information is destroyed. However, it should be noted that this pattern is not repeated when using a tailored baseline (Section 5.3.1.1). With a tailored baseline, like "closed door" in the open-door task instead of the generic "robot arm beside table", the opposite pattern can be observed, where r_{P1} and r_{P4} perform better than their "after heads" counterparts. It is therefore possible that the previous pattern was observed only by chance.

With a baseline tailored to the specific task, the RAS scores for delta features show smaller improvements, and goal-baseline regularization even performed worse than standard FuRL. This suggests that the performance is dependent on the chosen text baseline. It is curious, however, that these results are not very intuitive. Rather than performing better when the baseline is defined specifically for the given task, the performance is actually consistently better when the baseline is just a general description of the environment, ignoring the task-specific elements.

Since these results are unintuitive, it is possible that the generic baseline is actually also not optimal, and that there might exist a different type of baseline that could perform even better. But assuming that a generic baseline is better, then something that could be explored is using a general baseline that is more descriptive than just "robot arm beside table".

But with all of these results in mind, the conclusion made in this thesis is that fixed temporal information, provided through delta features or goal-baseline regularization, has little to no benefit in terms of sample efficiency. However, it is important to recognize the assumptions that this conclusion is made under.

One assumption is that this behavior would remain the same across all VLMs and across all pretraining objectives. The VLM used in this thesis is a LIV (Language-Image Value) VLM [13], which was trained specifically for the purpose of producing dense rewards in VLM RL using a language or image goal. In Chapter 4, two interpretations of $g-b$ were laid out. The first was that it represents the information in the baseline subtracted from the goal, thus creating a goal embedding of higher quality, and the second was that $g - b$ represented the generally desired direction, moving from the baseline to the goal. What these interpretations assume, however, is that they are valid under the representations learned using LIV. Future research could therefore examine the effect that the VLM architecture or pretraining objective could have on the performance of these methods.

6.2.3 The Second Stage: Directional Rewards

Instead of making alterations to the positional reward, adding a directional reward seems to be far more beneficial. Since the directional reward size λ was tuned for optimal RAS, the positional reward size ρ in the FuRL baseline was also tuned, to make the comparison fair. With the updated FuRL baseline, directional shaping outperformed it by **23.0%** (**78.5%** vs. **63.8%**) with r_{D5}^{VLM}, where the visual component in cosine similarity is Δs_t concatenated with a goal-baseline regularized s_t. Similar performance was observed for the other concatenated inputs (r_{D3}^{VLM} and r_{D4}^{VLM}), but with purely directional input Δs_t (r_{D2}^{VLM}) the RAS score decreased by **4.1%** compared to the FuRL baseline with optimal ρ (**61.2%** vs. **63.8%**).

However, the performance of r_{D2}^{VLM} is still surprisingly good, especially if the performance in each task is examined more closely. This is surprising because as was explained in Section 4.3.2 with a simplified 2D environment, the optimal direction should be dependent on the current state/position of the environment. In the example 2D environment, the directional reward with only directional input should only be able to give a kind of "global" directional reward, that is higher whenever the agent moves in the average direction observed in positive trajectories.

But what this explanation does not account for is that the embeddings produced by the VLM are far more complex than they would be for a simple 2D navigation problem (e.g. higher dimensionality). With more complex embeddings, it is possible that some of the changes/directions observed are *only* possible in certain states, which means that the directional reward function can infer the position from the given direction. This inference is likely not possible in all states however, so it would be better if a positional input is provided explicitly through concatenation, as demonstrated by the performance of r_{D3}^{VLM}, r_{D4}^{VLM} and r_{D5}^{VLM}.

For the textual baseline, the pattern observed for the positional rewards where the performance degrades with a tailored baseline is not seen again, as a generic baseline is slightly better for r_{D3}^{VLM}, while a tailored baseline is slightly better for r_{D4}^{VLM} and r_{D5}^{VLM}. No clear pattern can be discerned from these results, so the conclusion made in this thesis is that the choice of textual baseline has little to no effect for directional rewards.

But these results make it clear that directional shaping is preferable to purely positional shaping for sample efficiency. A big reason for why this might the case is that directional rewards directly model the value of the change induced by the action, which is beneficial because the purpose of RL is to optimize the policy, i.e. the actions chosen by the agent.

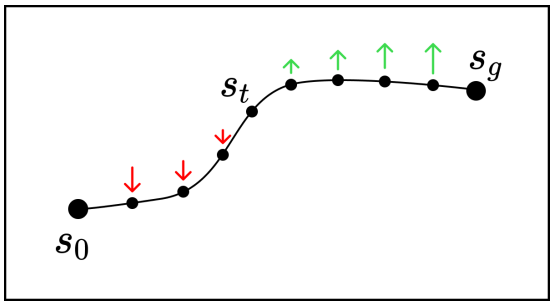


Figure 6.1: Positional reward optimization for a positive trajectory. To determine $r_P^{VLM}(s_t)$, the optimization makes r_P^{VLM} gradually decrease (red arrows) for states preceding s_t in the trajectory, and gradually increase (green arrows) for subsequent states.

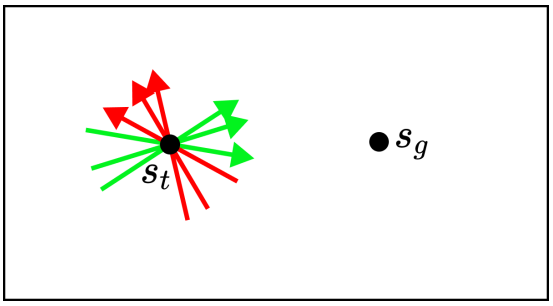


Figure 6.2: Directional reward optimization in s_t. To evaluate a direction observed in s_t, the reward function is trained with positive/negative (green/red) directions previously observed in s_t.

The improved performance could also stem from how the positional and directional reward functions are trained. This difference is illustrated in Figure 6.1 and Figure 6.2. For the positional value of s_t, the finetuning algorithm needs to account for the rewards along the entire trajectory. This makes training slower, because each step in the positional finetuning only tries to make the reward for s_t larger relative to s_{t-n}. This optimization only takes two states into consideration, but as is shown in Figure 6.1, the ultimate goal is to make the reward monotone over the entire trajectory. This means that multiple passes are necessary before the positional rewards have converged across the entire trajectory.

But to determine the value of a direction observed in s_t, the directional reward function only needs directions previously observed in s_t, as shown in Figure 6.2. This makes the optimization simpler, because there is no relative reward specification that one direction must be e.g. δ higher/lower than some other direction, as with positional rewards. And the directional rewards can converge locally in s_t, since the optimization only needs directions previously observed in s_t, while the positional finetuning needs to account for rewards along entire positive trajectories.

While directional shaping is better than purely positional shaping, it seems like a combination of the two is preferable to maximize RAS, as demonstrated by the results in Section 5.3.3. By increasing the positional reward such that ρ=1.0 and λ=4.0, the best performance was observed with a RAS score of **80.0%**, which is a **25.4%** improvement over the optimal FuRL baseline (**80.0%** vs. **63.8%**), and a **95.6%** improvement over SAC with no reward shaping (**80.0%** vs. **40.9%**).

But the contribution of the positional reward is still relatively small in the presence of a directional signal, as its exclusion only degrades the performance to a RAS score of **78.1%**. This suggests that the information encoded in the positional reward is almost entirely captured by the directional reward already (or at least, the information that is actually useful). Purely directional shaping is therefore preferable to avoid the extra computation, unless data collection is very expensive.

6.3 Defining Temporal Information

How is temporal information defined, and how can it be represented for efficient reward shaping in a sparse reward RL context?

In this thesis, temporal information was categorized as either fixed or adaptive. Both types describes the progression of the system through time, but fixed temporal information can only use inputs that do not change over the course of each episode. This allows the rewards to model the value of the state s_t from a new (fixed) perspective, but it is limiting because the reward function cannot account for how the system is *currently* changing, only how it has changed in relation to fixed points. Adaptive temporal information is less limited because it can use states *relative* to s_t, like s_{t-k}. This can be used to assess not just the current state of the system, but also the current change.

The utility of the current change in the context of RL can be explained if viewed through the lens of the agent's actions. In any given state, the agent has a wide range of actions that it can choose from, each inducing some change in the system, resulting in a new state. From this perspective, the new state is *downstream* from the change. In practice, modeling the value of the change should therefore be better than the value of the state, as it would provide a more direct action-consequence feedback to the agent.

This interpretation explains the results seen in Chapter 5. Directional reward functions receive adaptive temporal information, which means that they can model the value of the change directly. In doing so, directional shaping outperforms positional shaping, since the positional reward functions can only observe fixed temporal information. This also explains the relatively small improvements observed when incorporating fixed temporal information, since all positional rewards (new methods and FuRL baseline) model the value of the downstream state rather than the value of the change.

But the upside is that it is not a choice between using either fixed *or* adaptive temporal information, as they are each implemented separately in the positional and directional reward functions. This means that if a significant improvement can be found for how fixed temporal information can be incorporated into positional rewards, then that improvement can be used in combination with an additional directional reward, getting the best of both worlds.

For exploiting adaptive temporal information, a potential improvement could be to take more states into consideration than just s_t and s_{t-k}. This would be especially helpful for more complex tasks, where the agent needs to perform a specific sequence of actions one after the other. For this, temporal information could be encoded with a short-horizon sequence encoder, like a Transformer.

Finally, as explained in Section 6.2.2, using different baselines, VLM architectures, and pretraining objectives can be evaluated to determine whether they would be better suited for exploiting either fixed and adaptive temporal information.

6.4 Reward Curves and Finetuning

How can a VLM reward function be modeled such that it takes advantage of temporal inputs, and what should its training procedure look like?

In Section 5.3.1.2, the directional reward appears to behave as an indicator function. A possible interpretation of this is that $r_{\mathrm{D}}^{VLM}(s_t)$ is a signal of whether the observed change at s_t is beneficial or not in the eyes of the directional reward function. This stands in contrast to the positional reward function, which produces values between its minimum and maximum.

This can be explained by the difference in finetuning objective. For r_{P}^{VLM}, minimizing the ranking loss makes the reward for s_t larger/smaller *relative* to s_{t-k}/s_{t+k}. The directional finetuning does not have this relative reward specification. Instead, the directional reward in s_t is simply maximized for some directions and minimized for others. The result is that the directional reward is always either the minimum or the maximum, while the positional reward needs to produce values between the minimum and maximum, to ensure that $\ldots < r_{\mathrm{P}}^{VLM}(s_{t-k}) < r_{\mathrm{P}}^{VLM}(s_t) < r_{\mathrm{P}}^{VLM}(s_{t+k}) < \ldots$ holds along entire positive trajectories.

A relevant question here is whether this shape of the directional reward is desirable. Judging purely by performance, the current directional reward is clearly beneficial. However, it is possible that a different shape could improve performance even further.

To create a different shape for the directional reward function, the finetuning algorithm would have to be changed such that it is not just provided inputs for which it should minimize/maximize the reward. A potential change could be to provide the finetuning algorithm with a target, and train the reward function to produce the target by e.g. minimizing the squared error $(r-t)^2$, where r is the reward and t the target.

This target could represent the certainty that the given direction is beneficial or not, which could be computed using its prevalence in previous positive trajectories, either through some algorithm or with a separate neural network. The target would be positive if this algorithm/network is certain that the direction is beneficial, negative if it is certain that the direction is not beneficial, and close to 0 if it is uncertain.

Using certainty to determine the directional reward size could also be implemented outside of the finetuning algorithm. The directional reward function would then be left as a step function, but the output is multiplied by a factor determined by the certainty.

Alternatively, the directional reward function could produce something other than an embedding for computing the cosine similarity with the textual embedding. For instance, with the position s_t as input, it could output a desired direction $\widetilde{\Delta} s_t$. The $\widetilde{\Delta} s_t$ direction would then be treated as an oracle, and the reward would be computed using the angle between the oracle direction $\widetilde{\Delta} s_t$ and the direction Δs_t that was actually observed, e.g. through $\mathrm{csim}(\Delta s_t, \widetilde{\Delta} s_t)$.

Since $\widetilde{\Delta}s_t$ is a direction, the output could also be modeled using concepts from directional statistics. For instance, instead of the outputting a direction, the output could instead be the parameters for a von Mises-Fisher distribution, which is a simple probability distribution over directions. By outputting a probability distribution, this method would inherently model the uncertainty in the predicted direction, which can be used to determine the magnitude of the directional reward. One problem with these direction prediction methods is that they do not take textual information into account. However, this could be accounted for by including it in the input for the direction predictor, possibly through concatenation or cross-attention.

Bibliography

[1] Poorva Agrawal et al. "Advancements in Computer Vision: A Comprehensive Review". In: *2024 International Conference on Innovations and Challenges in Emerging Technologies (ICICET)*. 2024, pp. 1–6. DOI: 10.1109/ICICET59348.2024.10616321.

[2] Zhen Jia et al. "Human Image Generation: A Comprehensive Survey". In: *ACM Comput. Surv.* 56.11 (June 2024). ISSN: 0360-0300. DOI: 10.1145/3665869. URL: https://doi.org/10.1145/3665869.

[3] Bonan Min et al. "Recent Advances in Natural Language Processing via Large Pre-trained Language Models: A Survey". In: *ACM Comput. Surv.* 56.2 (Sept. 2023). ISSN: 0360-0300. DOI: 10.1145/3605943. URL: https://doi.org/10.1145/3605943.

[4] Florian Bordes et al. *An Introduction to Vision-Language Modeling*. 2024. arXiv: 2405.17247 [cs.LG]. URL: https://arxiv.org/abs/2405.17247.

[5] Kate Baumli et al. *Vision-Language Models as a Source of Rewards*. 2024. arXiv: 2312.09187 [cs.LG]. URL: https://arxiv.org/abs/2312.09187.

[6] Yuchen Cui et al. "Can foundation models perform zero-shot task specification for robot manipulation?" In: *Learning for dynamics and control conference*. PMLR. 2022, pp. 893–905.

[7] Juan Rocamonde et al. "Vision-Language Models are Zero-Shot Reward Models for Reinforcement Learning". In: *The Twelfth International Conference on Learning Representations*. 2024. URL: https://openreview.net/forum?id=N0I2RtD8je.

[8] Yuwei Fu et al. *FuRL: Visual-Language Models as Fuzzy Rewards for Reinforcement Learning*. 2024. eprint: arXiv:2406.00645.

[9] Richard S Sutton. "Reinforcement learning: An introduction". In: *A Bradford Book* (2018).

[10] Tuomas Haarnoja et al. "Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor". In: *Proceedings of the 35th International Conference on Machine Learning*. Ed. by Jennifer Dy and Andreas Krause. Vol. 80. Proceedings of Machine Learning Research. PMLR, July 2018, pp. 1861–1870. URL: https://proceedings.mlr.press/v80/haarnoja18b.html.

[11] Majid Ghasemi, Amir Hossein Moosavi, and Dariush Ebrahimi. "A Comprehensive Survey of Reinforcement Learning: From Algorithms to Practical Chal-

lenges". In: 2024. URL: https://api.semanticscholar.org/CorpusID:274422589.

[12] Babak Badnava et al. "A new potential-based reward shaping for reinforcement learning agent". In: *2023 IEEE 13th Annual Computing and Communication Workshop and Conference (CCWC)*. IEEE. 2023, pp. 01–06.

[13] Yecheng Jason Ma et al. "LIV: language-image representations and rewards for robotic control". In: *Proceedings of the 40th International Conference on Machine Learning*. ICML'23. Honolulu, Hawaii, USA: JMLR.org, 2023.

[14] Karl Weiss, Taghi M. Khoshgoftaar, and DingDing Wang. "A survey of transfer learning". In: *Journal of Big Data* 3.1 (May 2016). ISSN: 2196-1115. DOI: 10.1186/s40537-016-0043-6. URL: http://dx.doi.org/10.1186/s40537-016-0043-6.

[15] Tianhe Yu et al. "Meta-world: A benchmark and evaluation for multi-task and meta reinforcement learning". In: *Conference on robot learning*. PMLR. 2020, pp. 1094–1100.

A

Appendix 1

A.1 Positional Rewards

A.1.1 Generic Baseline

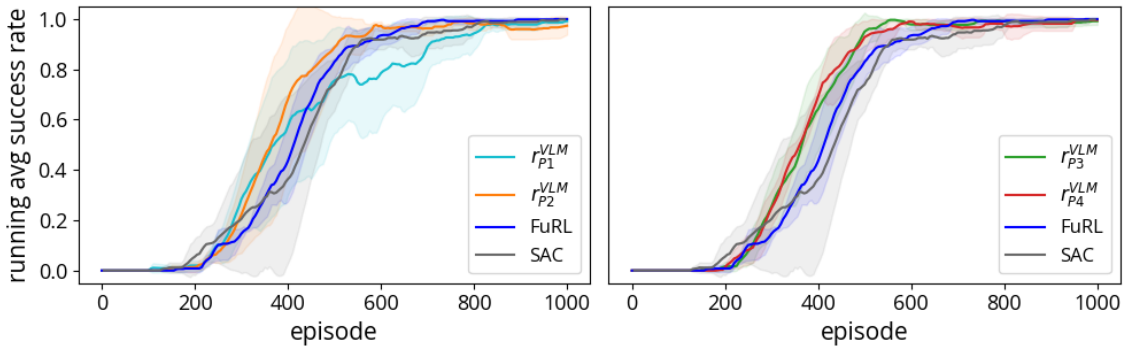


Figure A.1: Running average success rate and standard deviation over 150 training episodes with different positional rewards trained to perform the door-open task. In goal-baseline regularization, α=0.4.

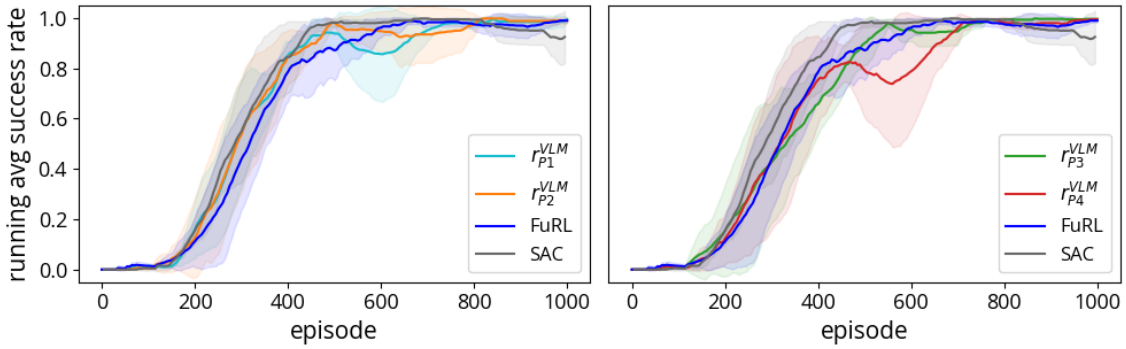


Figure A.2: Same as in A.1 but in the drawer-open environment.

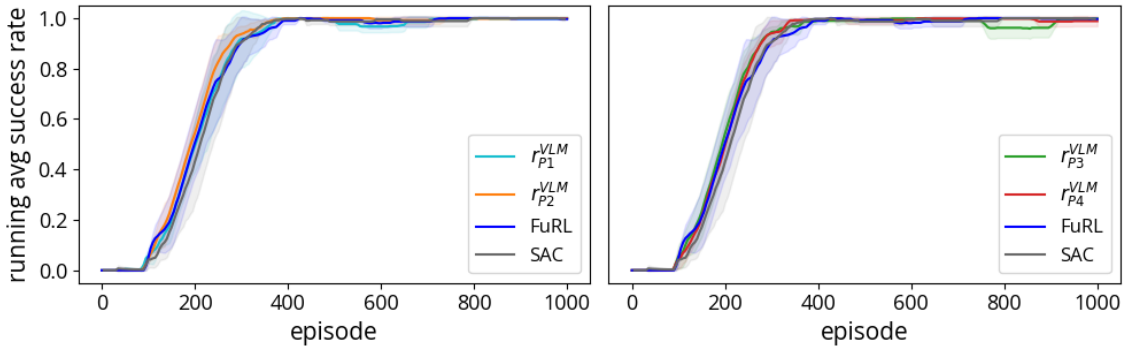


Figure A.3: Same as in A.1 but in the window-open environment.

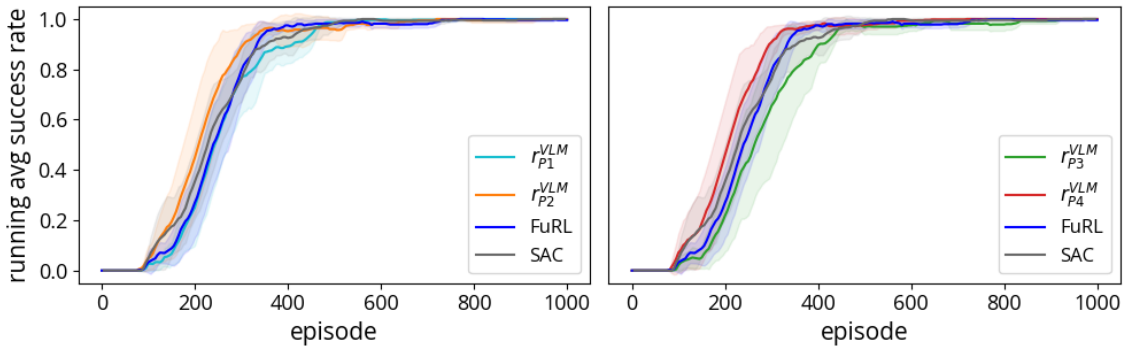


Figure A.4: Same as in A.1 but in the window-close environment.

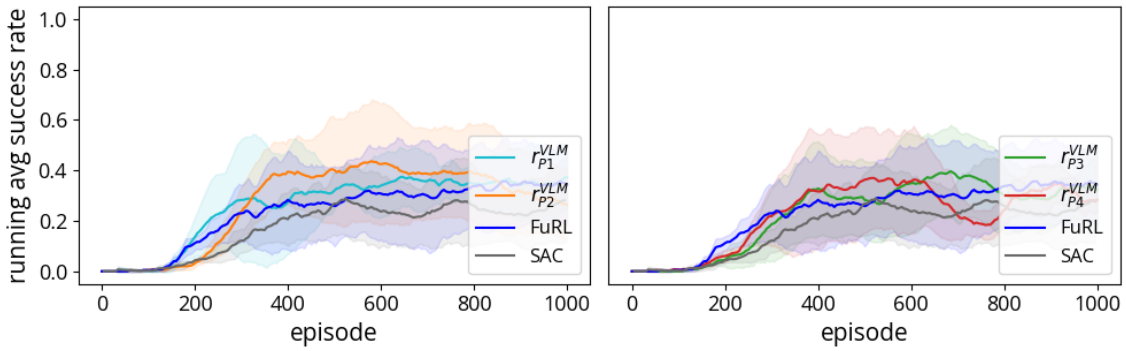


Figure A.5: Same as in A.1 but in the push environment.

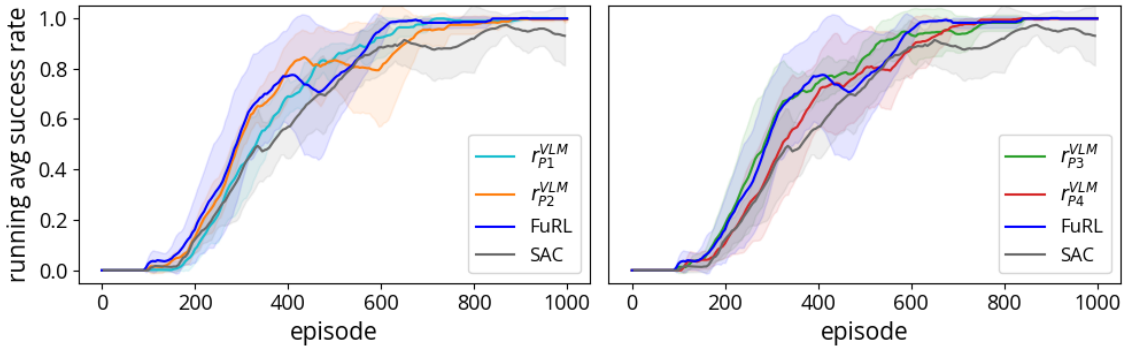


Figure A.6: Same as in A.1 but in the button-press-topdown environment.

A.1.2 Tailored Baseline

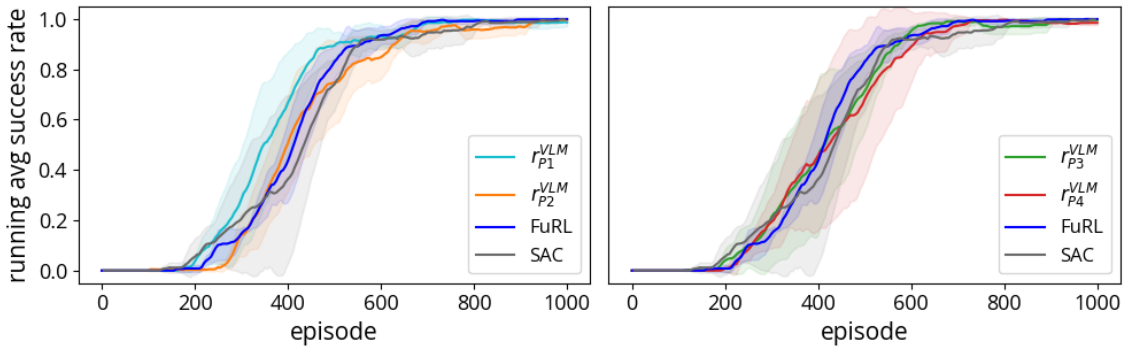


Figure A.7: Same setup as in Figure A.1, but with a baseline that is tailored to the given task.

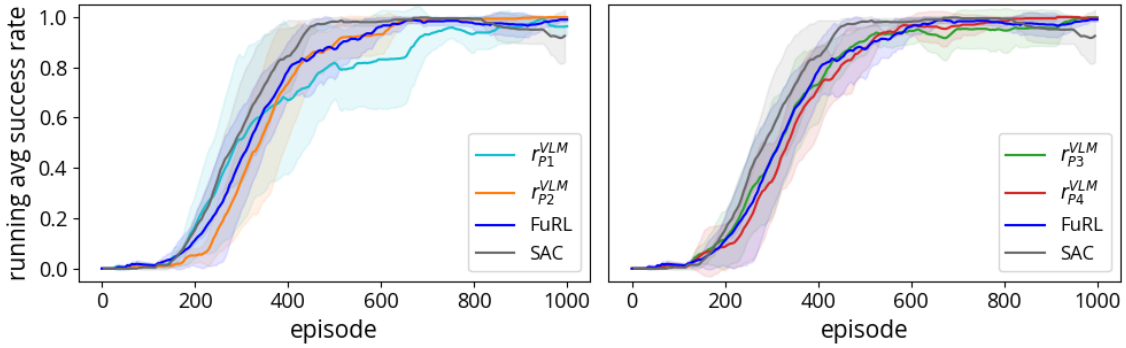


Figure A.8: Same as in A.7 but in the drawer-open environment.

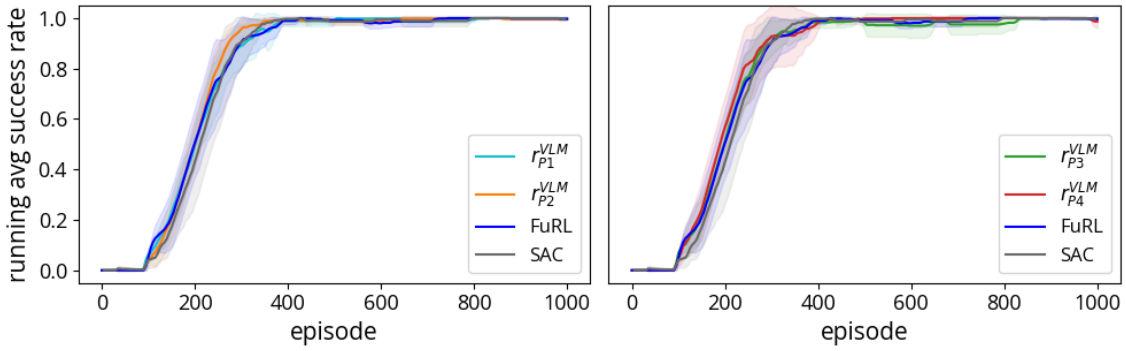


Figure A.9: Same as in A.7 but in the window-open environment.

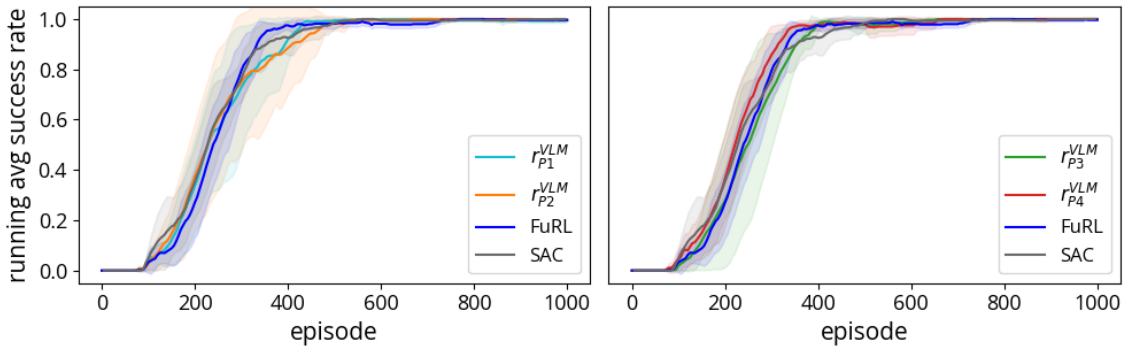


Figure A.10: Same as in A.7 but in the window-close environment.

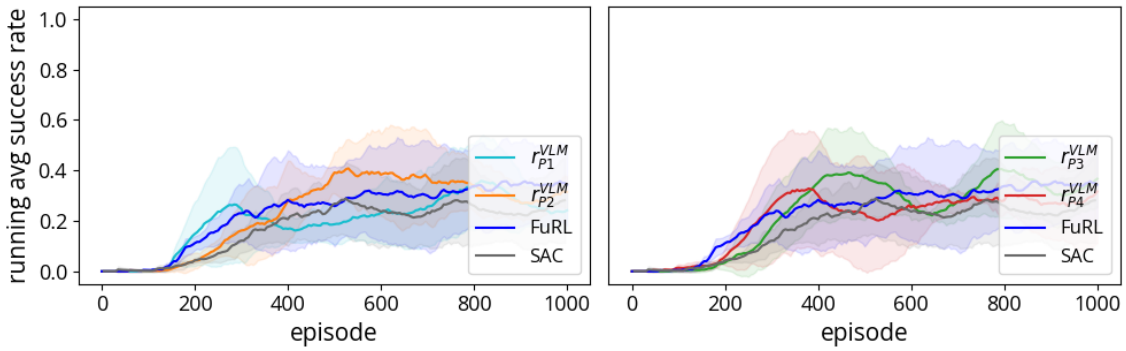


Figure A.11: Same as in A.7 but in the push environment.

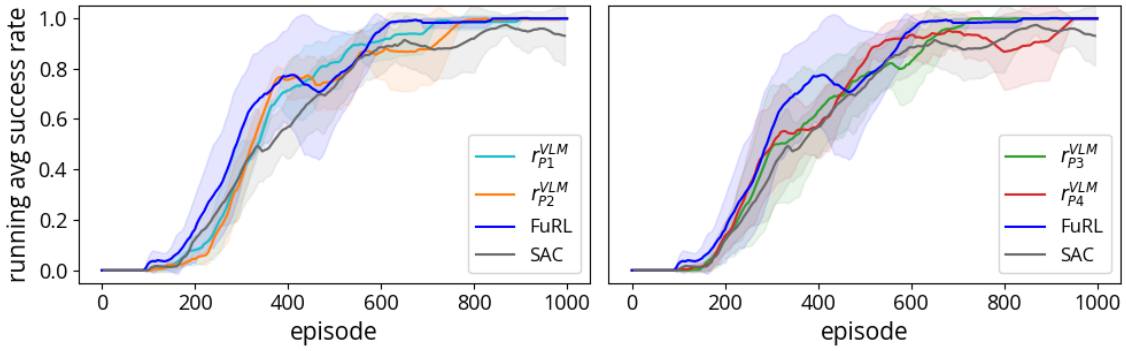


Figure A.12: Same as in A.7 but in the button-press-topdown environment.

A.2 Directional Rewards

A.2.1 Generic Baseline

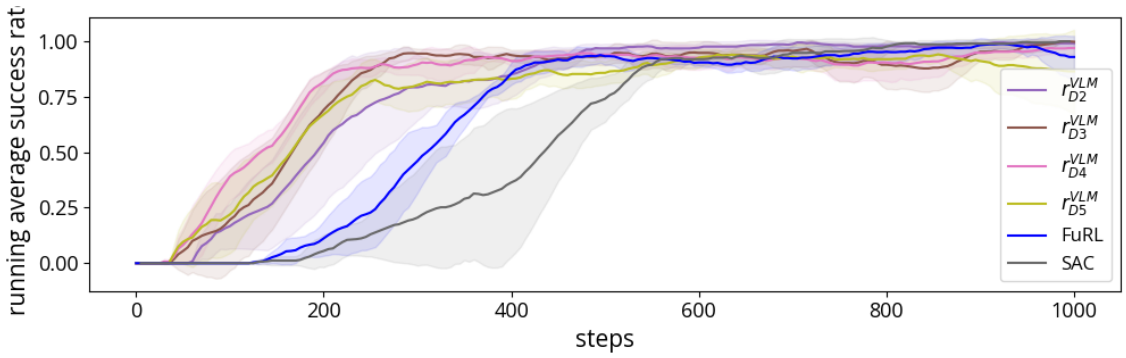


Figure A.13: Running average success rate over 150 episodes for r_{D2}^{VLM}, r_{D3}^{VLM}, r_{D4}^{VLM} and r_{D5}^{VLM}, with a generic baseline, ρ=0.05 and λ=80. In goal-baseline regularization, α=0.4.

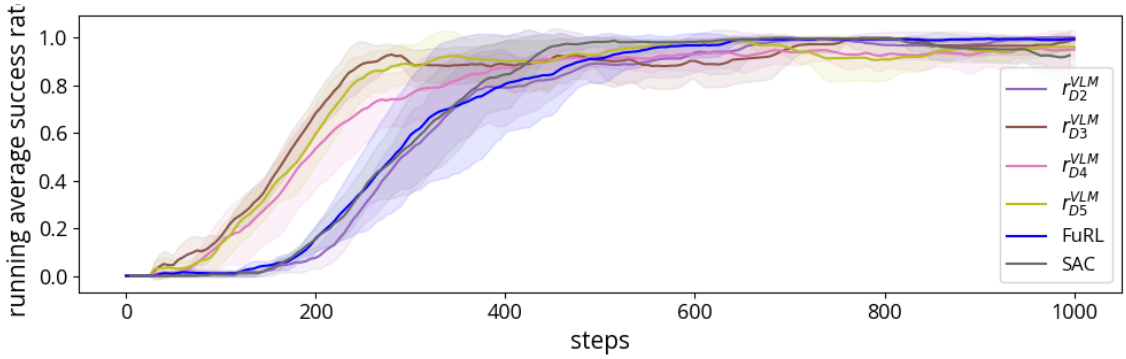


Figure A.14: Same as in A.13 but in the drawer-open environment.

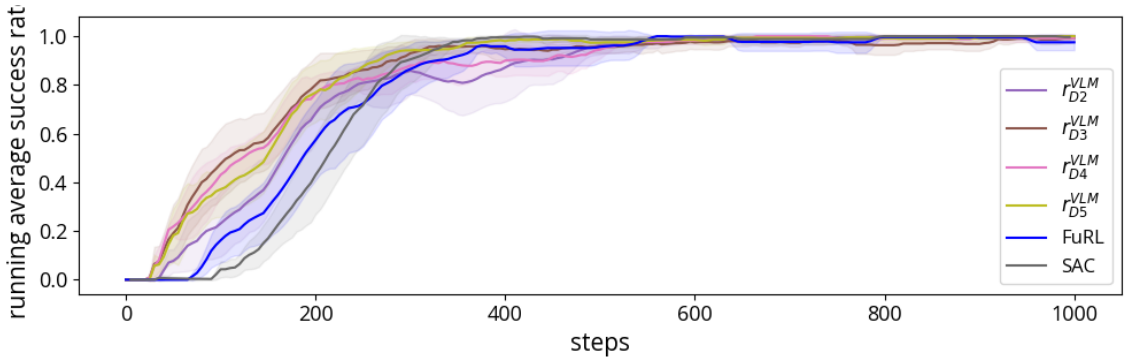


Figure A.15: Same as in A.13 but in the window-open environment.

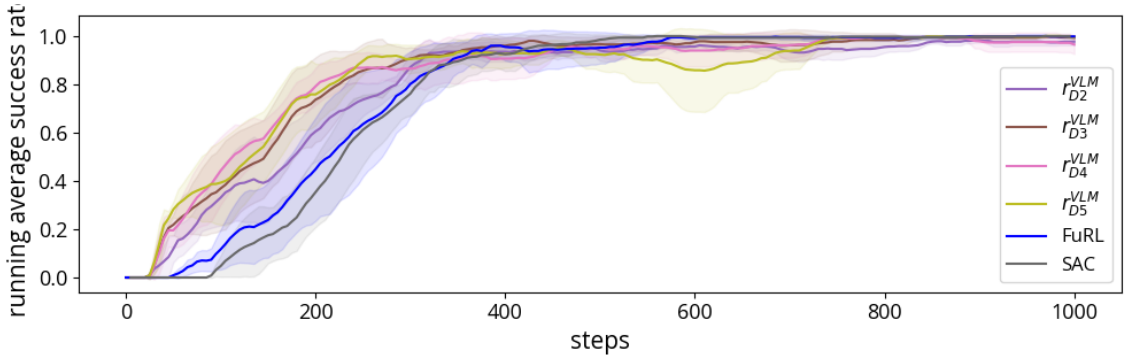


Figure A.16: Same as in A.13 but in the window-close environment.

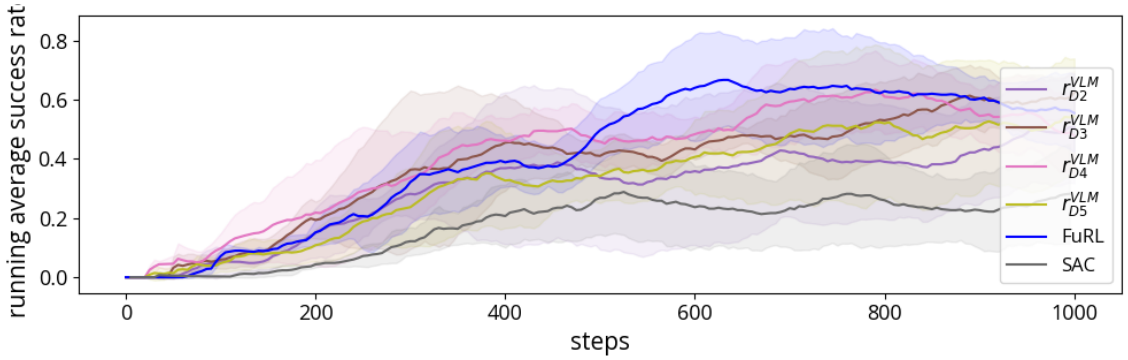


Figure A.17: Same as in A.13 but in the push environment.

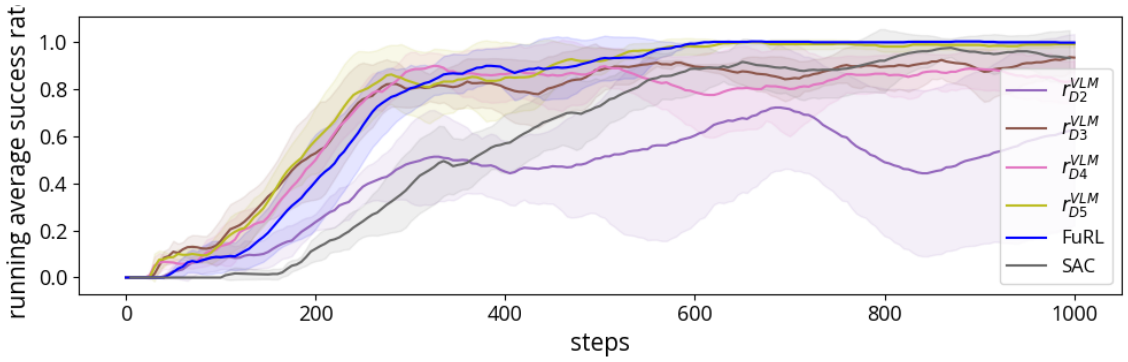


Figure A.18: Same as in A.13 but in the button-press-topdown environment.

A.2.2 Tailored Baseline

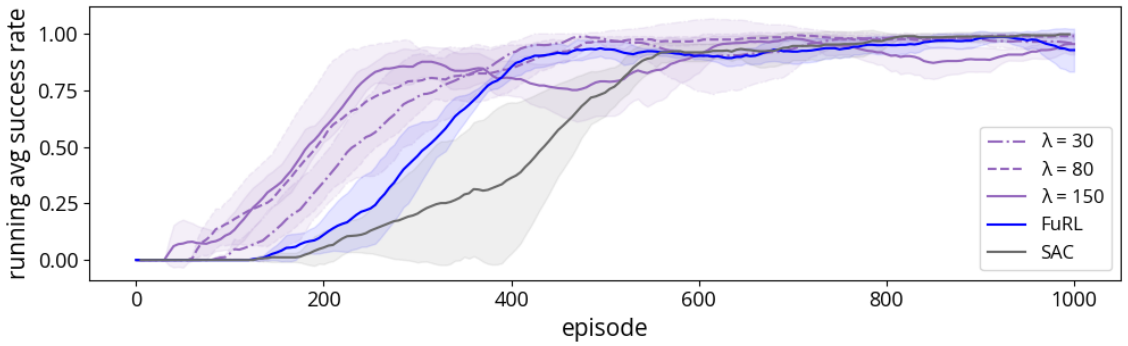


Figure A.19: Running average success rate in the door-open environment over 150 episodes for r_{D2}^{VLM}. Evaluated with 4 runs each for λ=30, λ=80 and λ=150.

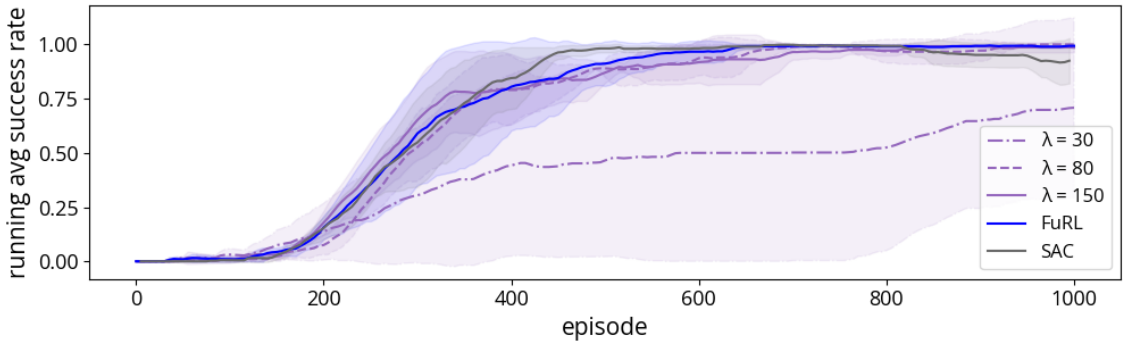


Figure A.20: Same as in A.19 but in the drawer-open environment.

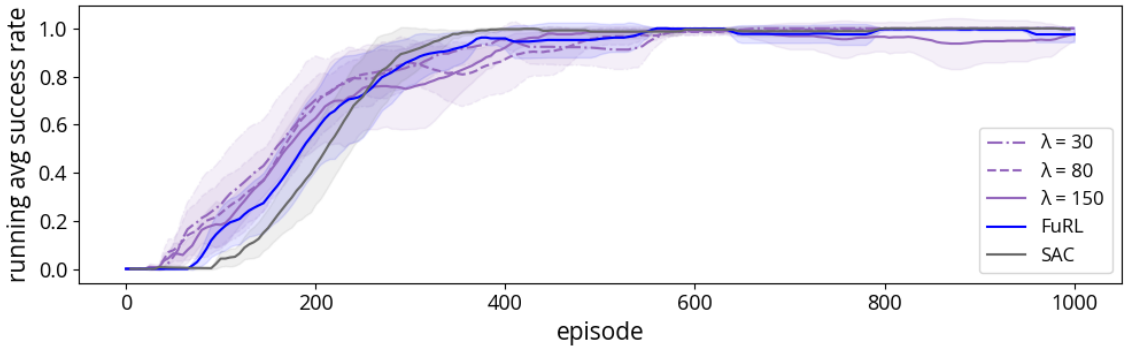


Figure A.21: Same as in A.19 but in the window-open environment.

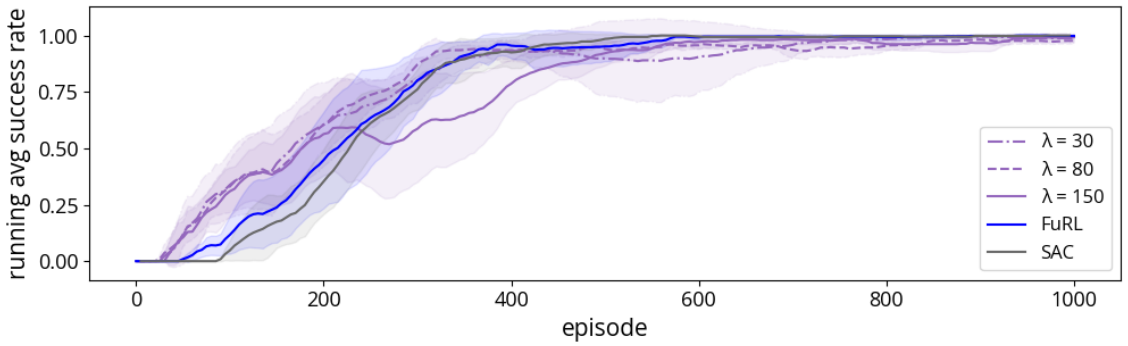


Figure A.22: Same as in A.19 but in the window-close environment.

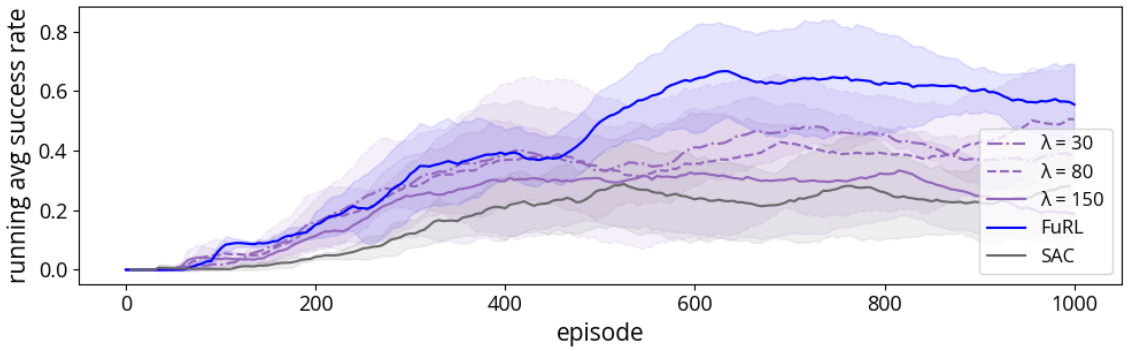


Figure A.23: Same as in A.19 but in the push environment.

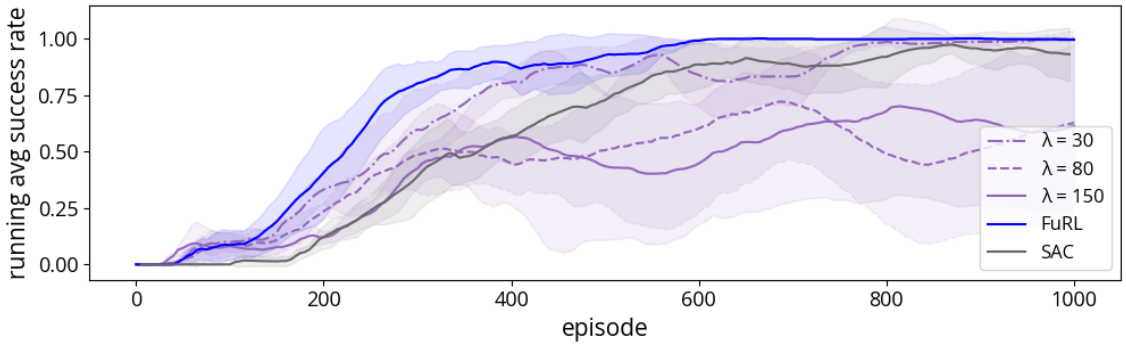


Figure A.24: Same as in A.19 but in the button-press-topdown environment.

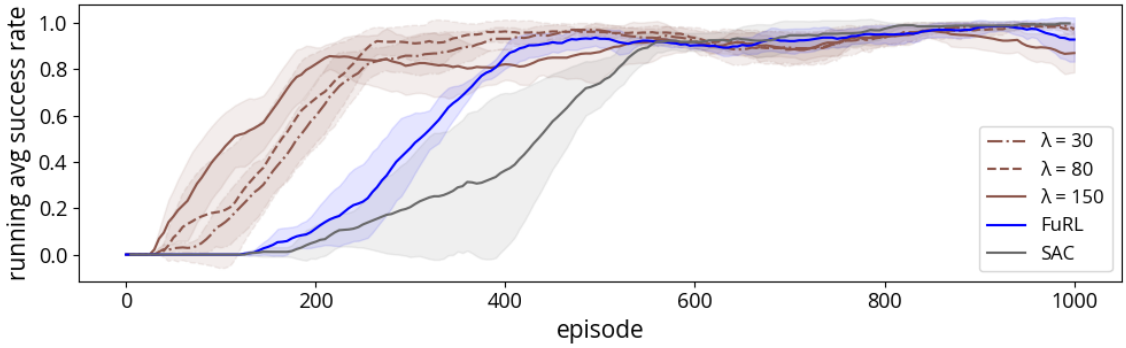


Figure A.25: Running average success rate in the door-open environment over 150 episodes for r_{D3}^{VLM}. Evaluated with 4 runs each for λ=30, λ=80 and λ=150.

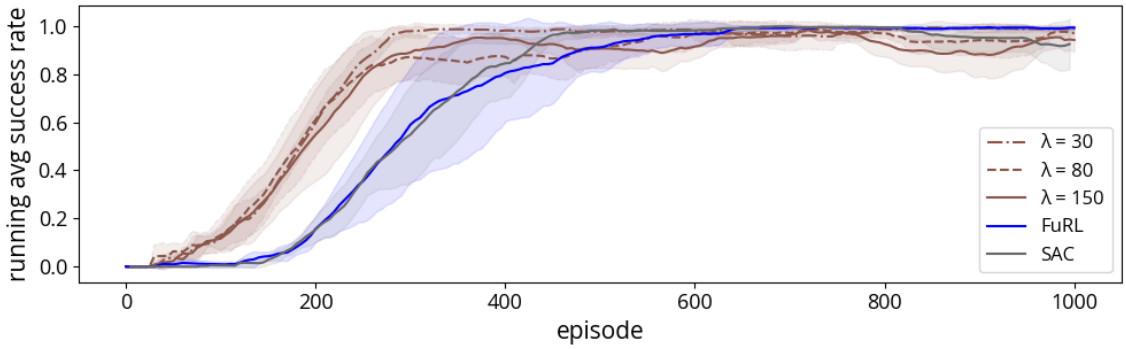


Figure A.26: Same as in A.25 but in the drawer-open environment.

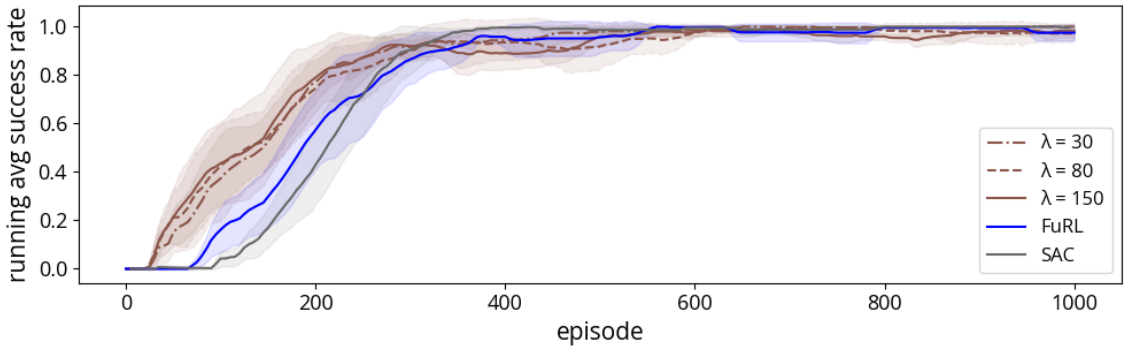


Figure A.27: Same as in A.25 but in the window-open environment.

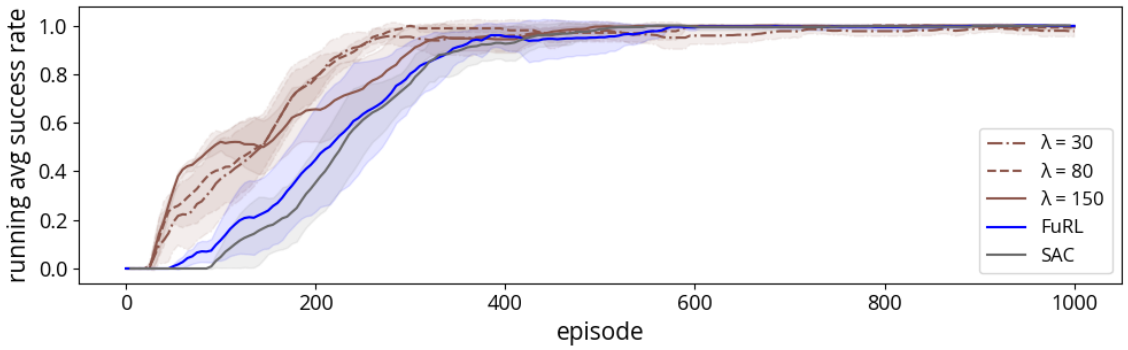


Figure A.28: Same as in A.25 but in the drawer-open environment.

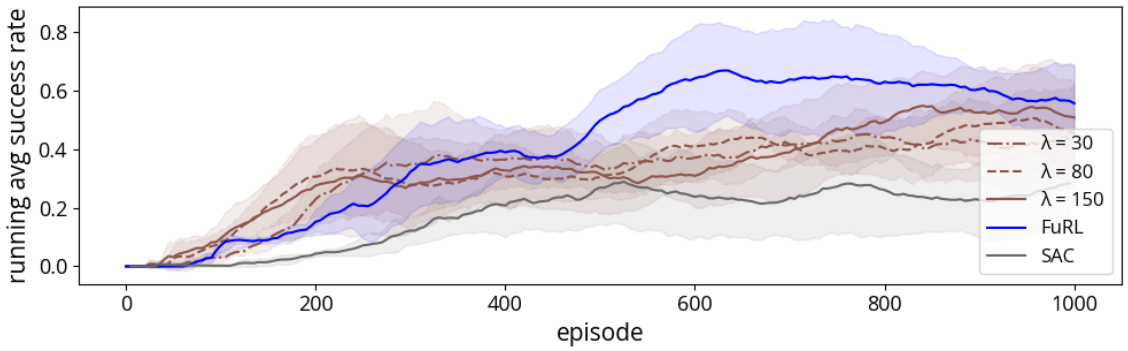


Figure A.29: Same as in A.25 but in the push environment.

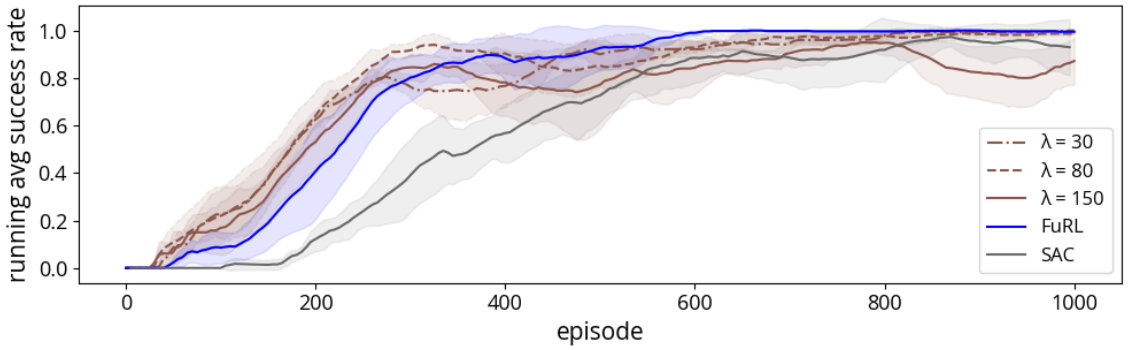


Figure A.30: Same as in A.25 but in the button-press-topdown environment.

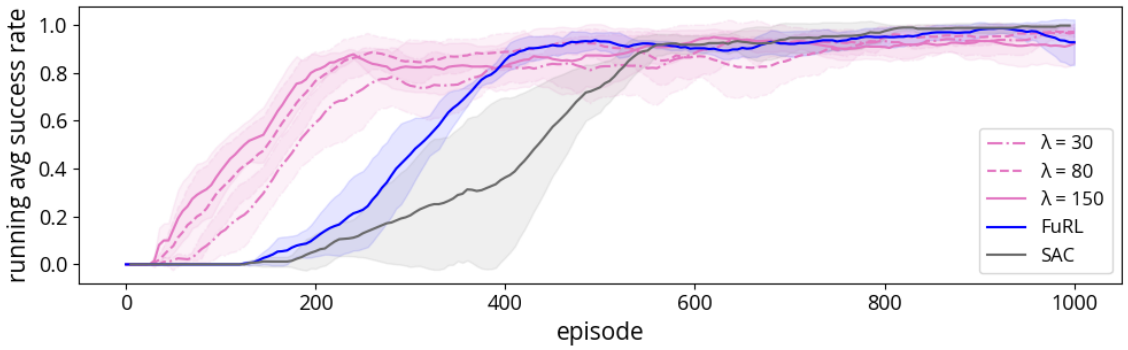


Figure A.31: Running average success rate in the door-open environment over 150 episodes for r_{D4}^{VLM}. Evaluated with 4 runs each for λ=30, λ=80 and λ=150.

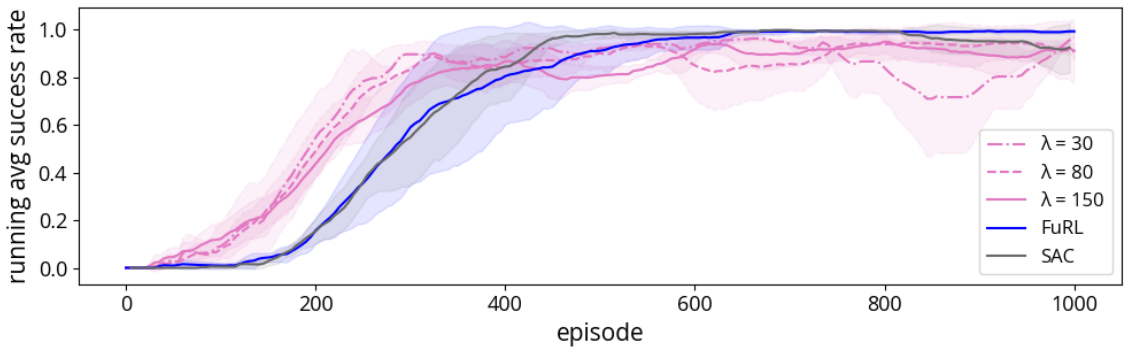


Figure A.32: Same as in A.31 but in the drawer-open environment.

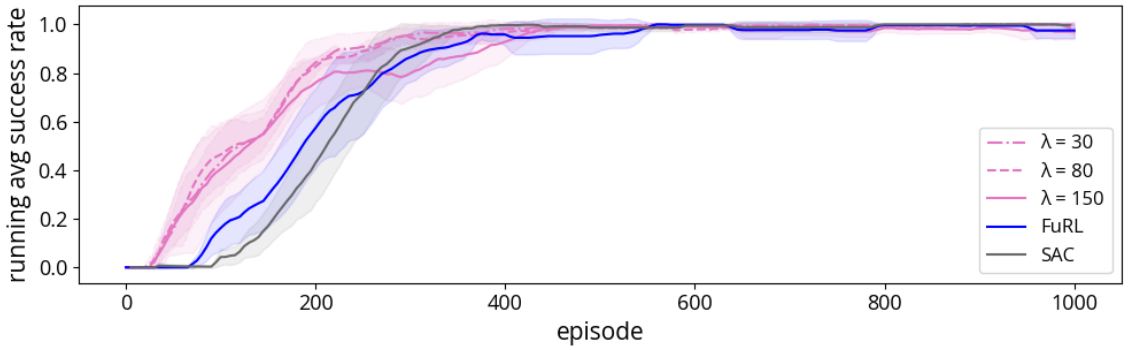


Figure A.33: Same as in A.31 but in the window-open environment.

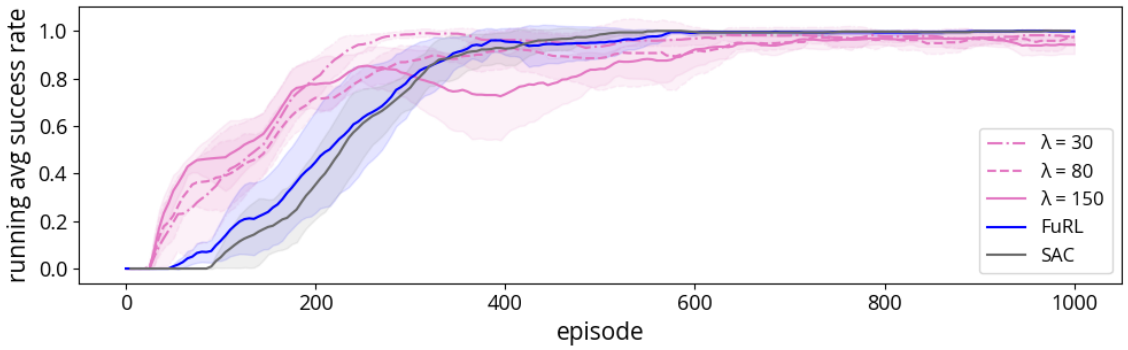


Figure A.34: Same as in A.31 but in the drawer-open environment.

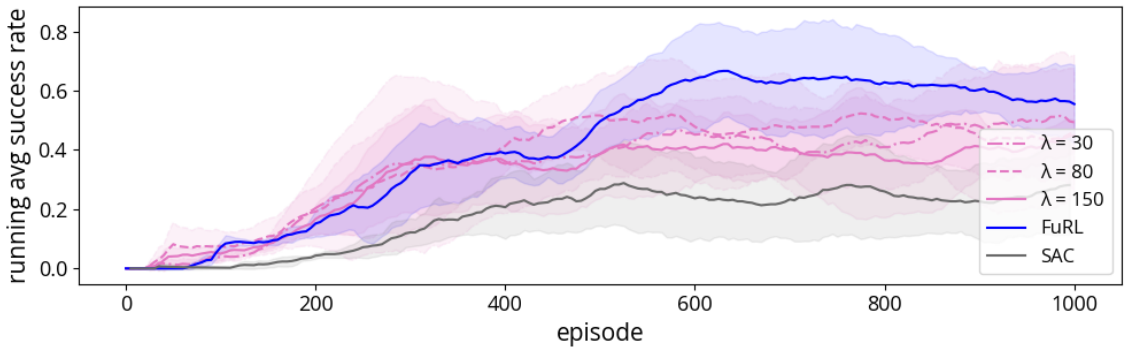


Figure A.35: Same as in A.31 but in the push environment.

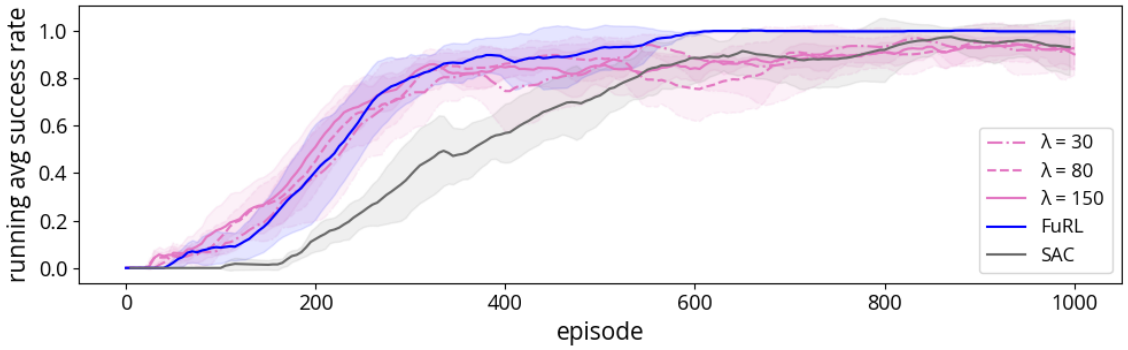


Figure A.36: Same as in A.31 but in the button-press-topdown environment.

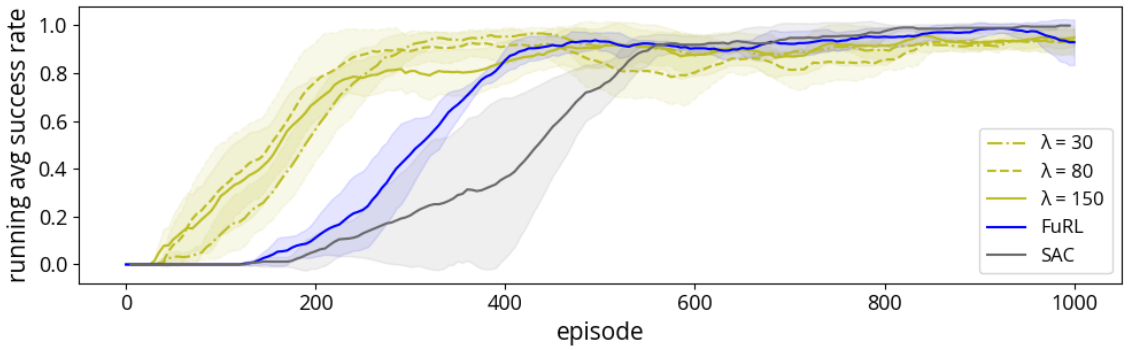


Figure A.37: Running average success rate in the door-open environment over 150 episodes for r_{D5}^{VLM}. Evaluated with 4 runs each for λ=30, λ=80 and λ=150. In goal-baseline regularization, α=0.4.

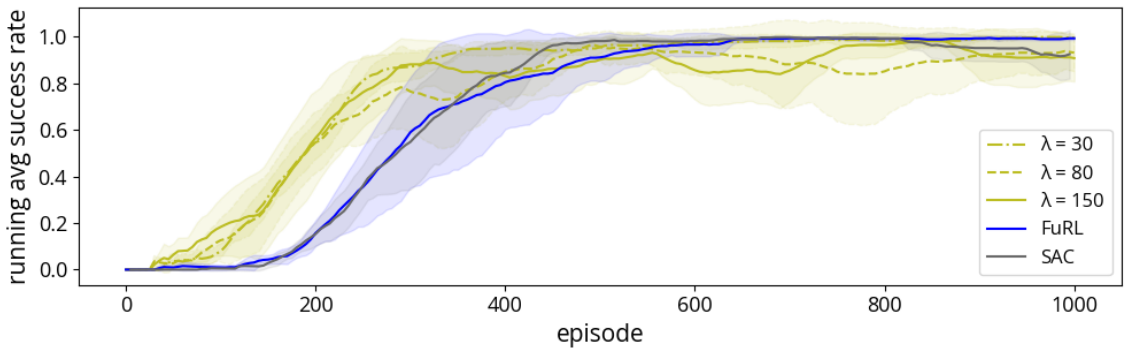


Figure A.38: Same as in A.37 but in the drawer-open environment.

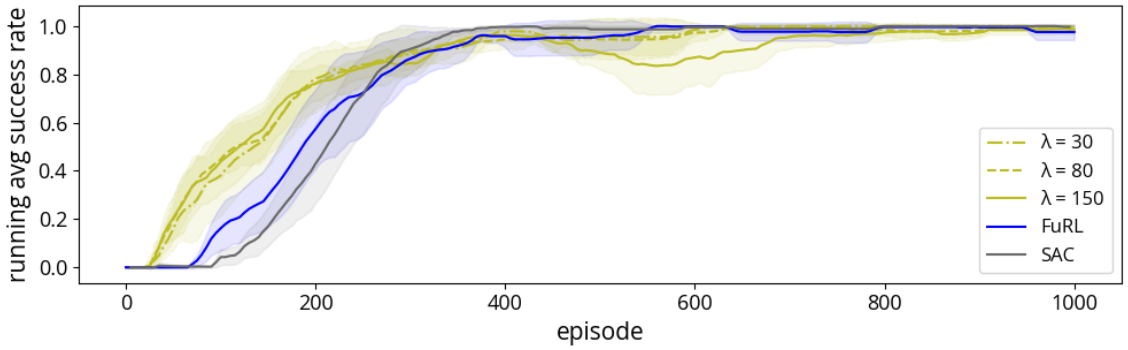


Figure A.39: Same as in A.37 but in the window-open environment.

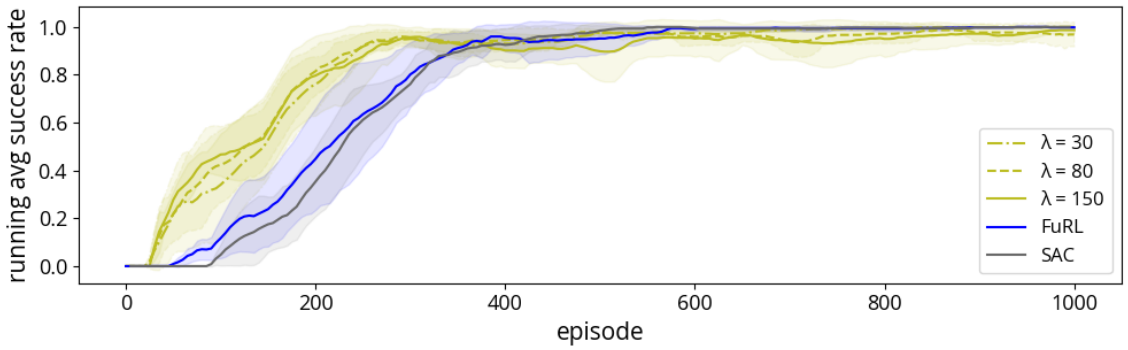


Figure A.40: Same as in A.37 but in the drawer-open environment.

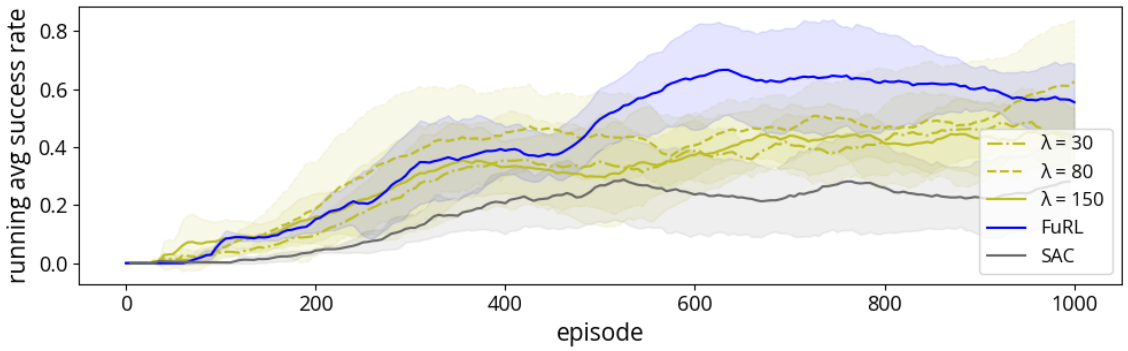


Figure A.41: Same as in A.37 but in the push environment.

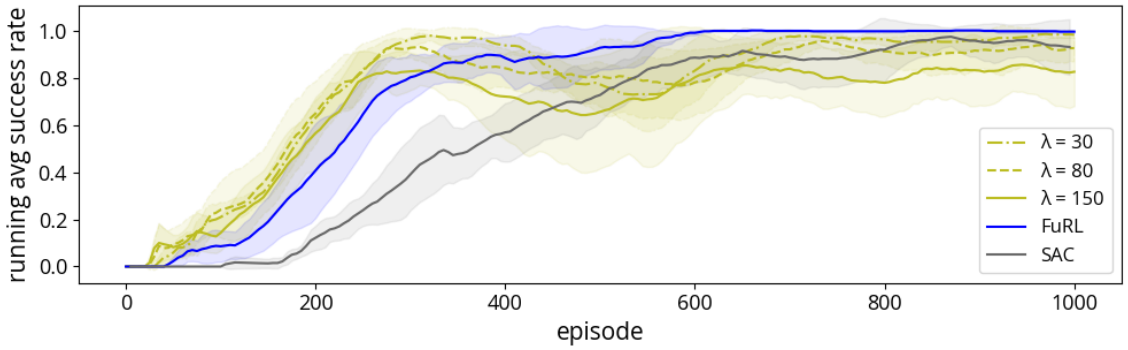


Figure A.42: Same as in A.37 but in the button-press-topdown environment.

DEPARTMENT OF SOME SUBJECT OR TECHNOLOGY
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden
www.chalmers.se

CHALMERS
UNIVERSITY OF TECHNOLOGY