university of groningen / faculty of science and engineering

Avoiding Brownian motion by utilizing consequential action selection in reinforcement learning for robotics

Bachelor's Project Thesis

Ivo Brink, s3981665, i.e.brink@student.rug.nl,
Supervisors: Prof. Dr. L. Schomaker & Sha Luo

Abstract: Robotics solutions often require high maintenance and possess low adaptability in the workspace and are therefore not as widely adopted in the industry as envisioned. Reinforcement learning is believed to be able to alleviate the disadvantages mentioned above. Regardless, progress has been slow, and utilized performance benchmarks are simplified representations of realistic working conditions. Furthermore, trajectories of seemingly random small state changes are observed in exploration phases that seem redundant for learning. This research uses more versatile benchmarks to test a different approach to action selection in reinforcement learning for robotics. A heuristic is developed that selects more meaningful actions instead of the actions that result in the so-called Brownian motion trajectories. These consequential actions are efforts that are estimated to provide more significant state differences in the utility landscape, with an anticipated increase in exploration and faster rewards. A state-of-the-art Soft Actor-Critic algorithm has been developed to train an UR5e robot in three different tasks. Results provided evidence of the existence of Brownian motion in the initial training phases over all three tasks. Selecting consequential actions provided more effective exploration in the early phases and, as a result, achieve an average increase in success rate of 10%. Consequential action selection proved less effective in later stages of learning, causing model convergence to be 25% slower overall.

1 Introduction

The field of robotics has not yet taken as much advantage of the latest advances in machine learning as expected. The majority of small and medium enterprises do not yet incorporate the usage of robotics, although it would greatly increase their efficiency and reduce labour costs (Ballestar et al., 2021). The impediment for these enterprises is the maintainability of a robotics solution since they require costly reprogramming for dynamically changing environments. (Pedersen et al., 2016). Realistic industrial working conditions require a high variation in the types of movement that are required to successfully perform a task, especially given the increasing demand for customization of products (Pedersen et al., 2016). Therefore, a cheap maintainable robotics solution is an untapped market. This argument is even stronger for home-based service robotics, where task variability is even much larger. It is hypothesized that machine learning can aid in reprogramming a robot by allowing it to learn from its environment using reinforcement learning (RL), decreasing changeover times and allowing for cheaper maintenance costs.

Traditional industrial robot behaviour is predominantly hard coded, and the means of observing the surroundings are limited. This makes industrial robots sensitive to prior calibration in the workspace. Their task is a fixed policy that does not allow for any improvement or adaptation. So, if an object within the production line strays by only a fraction of a centimeter, the process may need to be aborted. Corrective action and even a manual recalibration by a human operator are needed. This may be an acceptable solution in a strictly fixed environment, e.g., specially

adapted conveyor belts. However, the industrial production process may change, and objects' dimensions, weight, and expected trajectories may vary. Reprogramming a robot arm can be costly and time-consuming, and testing new software on the production line comes at high costs. Both online and offline programming to change robot behaviour are either time-consuming or cost expensive (Ajaykumar et al., 2021). Reinforcement learning could offer the maintainable solution necessary to offer great adaptability all around (Kober et al., 2013). Note that even when a great trainable RL solution is offered, fast training times are essential in robotics. That is because simulations are often sped up, and inaccurate representations of the working environment and real-life training require human supervision to reset the environment as well as the robot, not to mention that the robot will wear down, causing it to perform differently each time it resets.

Service robots could also reap the benefits from reinforcement learning. Consider a robot that has to clean a surface that has to gently run along with the shape of the surface in order to clean it properly. Applying too much force will damage the surface. Respectively, applying not enough force will not clean the surface at all. This robot requires a smooth and generalizable type of movement to clean any surface. A robot that has to squat a fly on a wall must be fast and quick to succeed in its task. One might see the problem that arises; the robotics task is not only about moving an end-effector from point A to B. An end-effector's speed, path, and momentum need to be considered to get results that will lead to a generalizable learning problem.

Progress in reinforcement learning for robotics is slower than anticipated. Especially the initial phases of reinforcement learning are very similar to the physics process of Brownian motion, where a particle is randomly disturbed from its current position in space with tiny bumps only (Karatzas & Shreve, 1998). This phenomenon is similar to motor babbling, where random short motor commands are used to explore and learn an environment. Nonetheless, motor babbling can be used to help learn the inverse kinematics of a robot (Kase et al., 2021). Only after a lengthy RL process the system might discover a fruitful uphill direction in the utility landscape. This so-called Brownian motion results in lengthy training times and inconsistent reward landscapes. It should be noted that the biological motor babbling in the fetal stage and infants is not 'Brownian' but impulse-like, i.e., with large excursions of the end effector.

This project proposes that a different approach to selecting actions in reinforcement learning can lead to a quicker and more natural way of learning movements. The idea is to distinguish between actions with a small reward effect and *consequential actions*, i.e., actions that lead to clear and reliable upward jumps in the utility landscape.

In robotics reinforcement learning, small changes in joint angles or torque around the joints of an arm are usually presented as actions (a_t) in the RL problem. Respectively, the joint angles of the arm are the state (s_t). Given a task such as moving an end-effector from point A to point B in the task space, the system effectively should learn to solve the inverse kinematics problem*. However, modern robots are already equipped with an inverse kinematics solver such as MoveIT (Chitta, 2016), so the challenges for the RL problem shift to higher-level aspects of the task. So, instead of the usual motor babbling that allows RL to move uphill in the utility landscape slowly, it is suggested that higher-level elements can be used to steer RL algorithms in the proper direction more effectively. However, this change to higher-level aspects has not been adopted yet. A more common approach to escape from slow reinforcement learning is to use imitation learning (Zhu et al., 2018), however, this is a costly approach because the exact target behaviour needs to be manually provided.

Other solutions include curriculum learning and transfer learning. Nonetheless, these solutions require a form of manual supervision as well, e.g., a human teacher. Ideally, the teacher is inherently present in the algorithm, so learning becomes an independent task. Therefore, a means to tackle the problem of slow learning in RL is to shift to

*The inverse kinematics problem is the problem of finding a vector of joint variables which produce the desired end-effector location in the 3D task space.

a different form of action selection, leaning more towards higher-level aspects, such as choosing parameters of trajectory planning methods instead of joint angles. This does not require any form of manual intervention that imitation, transfer, and curriculum learning have. However, current reinforcement learning in robotics still utilizes actions that have a negligible effect on the state, e.g., the torque of an electric motor or the change of angles inside joints.

This often means the environment is not adequately explored, which can cause long training times. As a result, the aforementioned Brownian motion can be observed. This study aims to provide more analyses into the phenomenon of Brownian motion in the current reinforcement learning paradigm and if it can be prevented. In the case that the prevention of this motion does not lead to better performance, it might be suggested that a shift towards the higher level actions is a solution to slow reinforcement learning in robotics and the seemingly unnecessary Brownian motion.

2 Related work

2.1 Human learning

The proposed research aims to determine the effect of using consequential action selection instead of the so-called Brownian movement in a reinforcement learning robotics environment. Instead of actions that provide limited exploration due to a small $(\Delta y, \Delta x, \Delta z)$ in the workspace, we need additional analysis of the action and reward space in order to sample more extensive excursions which may increase the probability of a larger effect on the reward. The idea is constituted by the fact that human motion is rewarded on the basis of large-effect actions. For example, catching a ball not quickly enough depends on the whole movement required to perform such a task. In current reinforcement learning, rather small-effect actions are rewarded (the first slight movement that initiates the movement, for example), not allowing a complete movement to be learned naturally. Instead, many consecutive small-effect actions are learned in a traditional reinforcement learning problem. A human would experience a reward for the complete movement, starting from its initial state all the way to reaching the goal. Whereas an RL algorithm rewards itself using the Bellman equation, back-propagating the reward through the sequence of small-effect actions performed, starting from the first action and using a discounted reward to identify the value of subsequent actions. This mapping of planned actions and motor commands in the brain is called an inverse model (Wolpert & Kawato, 1998). It is suggested that some representation of body dynamics and environment must also exist to allow the brain to coordinate behaviours that have not yet been encountered (Mussa-Ivaldi, 1999). This idea is similar to the problem of inverse kinematics, where the actuators are effectively modelled for ease of coordination, but also to reinforcement learning, where representations of agent state (body dynamics) are considered to construct a policy to achieve a goal.

The combination of both ideas requires a higher-level approach to learning movements in reinforcement learning. A method that considers actions with a high effect. This noticeably shifts the problem from a standard reaching task to learning different types of movements using larger effect actions. That is also why different benchmarks will be used in this research instead of a traditional reaching task. The so-called Brownian motion is hypothesised to actively prevent larger excursions which lead to the way of learning movement that is present in infants. Note that the presence of Brownian motion might also be a consequence of a dysfunctional method of action-selection as a whole, requiring a complete remodelling of the way actions are currently represented. Nonetheless, for this research, the focus is on encouraging larger-effect actions within the current paradigm.

2.2 Reinforcement learning

2.2.1 Preliminaries

Reinforcement learning problems are considered as Markov decision processes (MDPs) (Sutton & Barto, 2018). An MDP is defined as the tuple (S, A, p, r), where:

- S represents all possible states, so $s_t \in S$ is the state the agent is in.

- A represents all possible actions, so $a_t \in S$ is the action the agent performs.
- $p : S \times S \times A \to [0, \infty)$ is the state transition probability, which means the probability of the next state given s_t and a_t.
- $r : S \times A$ is the reward on each transition.

Reinforcement learning is a method to accommodate a learnable solution that aims to solve the MDP effectively. An RL algorithm learns to achieve an optimal policy, referred to as π. A policy is a method in which actions are chosen given the current state. A perfect policy always chooses the actions leading to the biggest sum of rewards. Note that in a robotics environment, the state and action space are often continuous, meaning there are infinite state transitions.

2.2.2 Robotics

Robotics environments are ultimately challenging reinforcement learning problems. Sparse reward landscapes make it hard to use policy gradient solutions (Nachum et al., 2016), and value functions have issues with the continuous action space. The actions, the angles of the joints or the torque presented by an electric motor consist of continuous values just as the states can be presented in joint angle space or Cartesian coordinates. This results in endless variations of action state combinations. Depending on the task at hand, rewards are often sparse, meaning that a reward is only obtained when a specific sequence of actions has occurred and led to a desirable outcome. Ideally, a reinforcement learning algorithm wishes to predict the reward gained by taking any action from any state. This becomes extraordinarily hard in continuous state and action spaces. Especially when rewards are very rarely obtained using random initial exploration.

2.2.3 Type of algorithms

There exist two types of reinforcement learning algorithms. On one side, there are on-policy algorithms such as TRPO and PPO (Schulman et al., 2017, 2015). On-policy algorithms require a high number of samples to discover the gradient of a policy which requires longer training times. This is because they only use the experiences generated by the current policy to update it. Nevertheless, they are stable (relatively insensitive to hyperparameter tuning). The Proximal Policy Optimization (PPO) algorithm is widely used in robotics environments and works quite effectively. On the other side, there are off-policy algorithms such as DDPG-HER and TD3 (Andrychowicz et al., 2017; Dankwa & Zheng, 2019). However, these algorithms are very unstable (sensitive to hyper-parameter tuning), yet they have better sample efficiency as they learn from all past experiences. However, there is a best of both worlds algorithm called Soft Actor-Critic (SAC). It is a model-free off-policy entropy-regularized reinforcement learning algorithm that achieves the best performance for most continuous state and actions spaces (Haarnoja et al., 2018). The off-policy characteristic allows the algorithm to learn from past experiences not generated by the current policy, increasing sample efficiency.

Nonetheless, SAC is far more stable than the other off-policy alternatives. The regularized entropy enables the algorithm to behave stochastically, encouraging exploration so that the algorithm is not likely to get stuck in local optima. The algorithm is considered state-of-the-art in current reinforcement learning and will be used throughout this study to solve different robotics tasks.

2.2.4 Soft Actor-Critic

The term "Soft" in Soft Actor-Critic means the entropy is regularized. The algorithm favours maximal entropy due to a regularization term in the loss function. A traditional reinforcement learning algorithm recognizes the best policy (π^*) from obtaining the largest sum of the reward ($r(s_t, a_t)$) over all time steps (t), see equation 2.1. The difference with SAC is that an entropy term is also maximized in the policy, as depicted in equation 2.2. In contrast, traditional Actor-Critic algorithms use entropy as an external exploration method (Mnih et al., 2016). The α is a hyperparameter that allows us to regulate the entropy term as desired.

Equation 2.1 Traditional reinforcement learning policy equation.

$$\pi^* = argmax_\pi \sum_{t=0}^{T} \mathbb{E}_{(s_t,a_t)\sim\rho_\pi}[r(s_t,a_t)] \tag{2.1}$$

Equation 2.2 SAC stochastic policy equation.

$$\pi^* = argmax_\pi \sum_{t=0}^{T} \mathbb{E}_{(s_t,a_t)\sim\rho_\pi}[r(s_t,a_t)+\alpha\mathcal{H}(\pi(\cdot|s_t))] \tag{2.2}$$

The entropy term $\mathcal{H}(\pi(\cdot|s_t))]$ is described in equation 2.3. Here the negative logarithm of the probability of an action being chosen given the current state in the current policy is used to facilitate stochastic behaviour by including it in the policy equation. This means that a high probability of an action being chosen by the current policy is disadvantageous to the update, e.g., the loss function encourages stochastic (non-deterministic) behaviour.

Equation 2.3 Definition of entropy term used in SAC.

$$\mathcal{H}(\pi(\cdot|s_t) = \mathbb{E}_{a\sim\pi(\cdot|s))}[-\log(\pi(a|s))] \tag{2.3}$$

The actor-critic architecture uses two neural networks to get the best policy as follows. The first neural network, called the critic estimates a Q-value ($Q(s_t, a_t)$) for each state and action combination (in reality, it consists of two networks approximating two Q-values to mitigate positive bias by taking the minimum every time (Haarnoja et al., 2018)). The network parameters of the critic are learned by minimizing equation 2.4, often referred to as the temporal difference error. Note that θ refers to the network parameters of the critic. Equation 2.4 shows a relatively standard loss function inside reinforcement learning. However, the $V_\theta(s_{t+1})$ is different than normal, as it includes the entropy term $-\alpha\log(\pi_\gamma(a|s_{t+1})$ (equation 2.5)

Equation 2.4 Critic loss to be minimized.

$$J_Q(\theta) = \mathbb{E}[(Q_\theta(s_t,a_t) - (r(s_t,a_t) + \gamma V_\theta(s_{t+1})))^2] \tag{2.4}$$

Equation 2.5 Value of next state for critic update.

$$V_\theta(s_{t+1}) = \mathbb{E}[Q_\theta(s_{t+1},a) - \alpha\log(\pi_\phi(a|s_{t+1}))] \tag{2.5}$$

The second network, called the actor, approximates the policy as depicted in equation 2.6. Note that the parameters of the actor-network are referred to as ϕ. This ϕ is mostly used in the entropy term to refer to the policy of which the probability of an action being taken given the state is used to punish deterministic behaviour. Equation 2.6 is to be minimized, so a high Q-value and low probability of actions being selected are desirable for the policy in SAC.

Equation 2.6 Actor loss to be minimized.

$$J_\pi(\phi) = \mathbb{E}_{s\sim\mathcal{D}}[\mathbb{E}_{a\sim\pi_\phi(\cdot|s)}[\alpha\log\pi_\phi(a|s) - Q_\theta(s,a)]] \tag{2.6}$$

3 Methodology

3.1 Environment

The environment is introduced first as a means to better understand the study described. A Universal Robot model 5e (UR5e) was set up using Gazebo and ROS (Quigley et al., 2009; Qian et al., 2014), as a means to analyse the current problems with RL in robotics and to test different methods of solving them. An openAI gym-like environment was provided by Sha Luo from her Self-Imitation Learning by Planning research (Luo et al., 2020). This code base offered the necessary controls to use a reinforcement learning algorithm to steer the arm using the proper drivers as provided by the robot manufacturer (Universal Robots). The arm is then controlled using Python scripts based on kinematics, ROS, and neural network libraries. The robot arm consists of six joints, of which three

(base, shoulder, and elbow) will be used to control the arm in this study. The three wrist joints will be disabled since their functionality is unnecessary for the tasks presented here.

The state s_t is described in equation 3.1, here $\theta_1, \theta_2, \theta_3$ are the angles in radians of the three operational joints respectively, and the x_s, y_s, z_s are the three-dimensional coordinates of the end-effector location (denoted by the s). In equation 3.2, an action (a_t) is the change in angles for the three joints, thus $\Delta\theta_1, \Delta\theta_2, \Delta\theta_3$. Equation 3.3 shows the three-dimensional coordinates x_g, y_g, z_g of the goal location (denoted by the g). All the values are illustrated in an image of the UR5e robot in Figure D.2.

Equation 3.3 State and action spaces

$$s_t = \{\theta_1, \theta_2, \theta_3, x_s, y_s, z_s\} \tag{3.1}$$

$$a_t = \{\Delta\theta_1, \Delta\theta_2, \Delta\theta_3\} \tag{3.2}$$

$$g = \{x_g, y_g, z_g\} \tag{3.3}$$

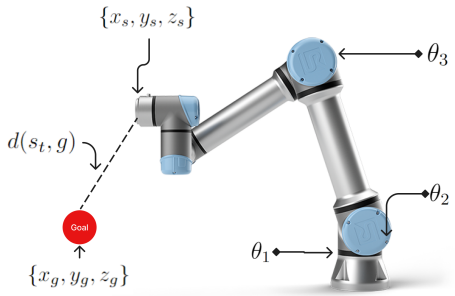

Figure 3.1: UR5e robot with illustration of the values.

The study is separated into two modules. One is designing a robotic reinforcement learning environment to create different tasks with corresponding reward signals and necessary observations. The other includes the design of selecting consequential actions in the Soft Actor-Critic algorithm.

3.2 Soft Actor-Critic

The implementation details of the Soft Actor-Critic algorithm can be found in Appendix B. Note that the actor uses two output layers so that a distribution with a mean (μ) and a standard deviation (σ) can be used to sample actions stochastically.

3.3 Robotic tasks

In order to elaborately test any model on its performance, we need to consider its applicability throughout different tasks. Since learning applications demand high generalizability, different tasks need to be designed to quantify this generalizability. Traditionally, the reaching task is the most appropriate benchmark for a robotic arm. The arm is initialized in some random position and needs to figure out the inverse kinematics to get to a given randomly positioned goal. In this study, two additional benchmarks will be used. A via-point task that requires the robot to coordinate a movement to a goal from a specific direction and a velocity task that requires the arm to move rapidly to the single-point destination.

3.3.1 Task 1: Reaching

The reaching task is the most straightforward robotics task. It will be referred to as the first task or task 1. It illustrates the exact problem of inverse kinematics by positioning the end effector to a predefined target. The UR5e robot will sample random joint angles within the predefined workspace so that the robot's initial pose differs at the start of each epoch. The goal is generated similarly, by sampling random joint angles possible end effector locations can be calculated using forward kinematics. A selected goal is declared *valid* when the sampled joint angles adhere to restrictions that manage the robot so that it does not collide with itself. These angles can be found in Appendix A.

In order to reward and punish desired and undesired behaviour, the environment needs a carefully hand-crafted reward signal. Note that this process is prone to calibration errors that may

affect performance. Nonetheless, the reaching task allowed for a simplistic reward signal approach. A reward of one is obtained when the robot arm reaches its goal within a defined criteria threshold, depicted in the reward equations. At all other times, the reward equals the negative Euclidean distance between the goal and end effector. The following is depicted in equation 3.4, 3.5, and 3.6.

Equation 3.4 Euclidean distance from one point to another.

$$d(a,b) = \sqrt{(x_a - x_b)^2 + (y_a - y_b)^2 + (z_a - z_b)^2} \tag{3.4}$$

Equation 3.5 Error used to check whether goal is reached.

$$e(s_t, g) = max(|x_s - x_g|, |y_s - y_g|, |z_s - z_g|) \tag{3.5}$$

Equation 3.6 Reward signal task 1.

$$r(s_t, g) = \begin{cases} 1 & \text{if } e(s_t, g) < 0.1, \\ -d(s_t, g) & \text{otherwise} \end{cases} \tag{3.6}$$

This punishment is designed to steer the arm in the right direction effectively. The alternative presents a sparse reward environment with prolonged sporadic learning experiences.

3.3.2 Task 2: reaching, via point

The second task, the so-called via-point task, describes a more defined movement of the end effector. This task is referred to as the second task (task 2). It imposes more constraints on the arm by requiring it to move in the vicinity of a second point, the via-point. This method inherently requires the movement of the arm to come from a predefined angle. The via-point needs to be reached before the final goal can be reached. This task requires a more complicated reward signal. The reward is similarly constructed as depicted in the original reaching task. The punishment equals the negative Euclidean distance between the via-point and the end effector as long as the via-point is not reached. A threshold determines when the via point is reached, and a reward of 1 is obtained when this threshold is crossed. The reward signal switches so that the distance between the final goal and the end effector becomes the penalty after the via-point is reached. A new more stringent threshold determines when the final goal is reached. A reward of 10 is awarded upon completion. The reward for the final goal is increased drastically to indicate that reaching the final goal is the highest priority.

The distance criteria for the via-point, e.g., the threshold that considers the end effector close enough, is half as strict as the threshold of the final goal. As can be observed in equation 3.7 and 3.8. The via-point is determined as follows, random joint angles are sampled until a possible end-effector location (determined using forward kinematics) falls within the Euclidean distance of $[0.2, 0.4]$ from the original goal. The original goal is created as discussed in the reaching task.

Equations 3.7 and 3.8 describe the reward landscape behaviour as follows, where g_1 is the final goal and g_2 is the via-point. Equation 3.7 is used until the via-point reward is obtained, and the environment switches to the new signal described in equation 3.8.

Equation 3.7 Reward signal task 2 via point.

$$r(s_t, g_2) = \begin{cases} 1 & \text{if } e(s_t, g_2) < 0.2, \\ -d(s_t, g_2) & \text{otherwise} \end{cases} \tag{3.7}$$

Equation 3.8 Reward signal task 2 final goal.

$$r(s_t, g_1) = \begin{cases} 10 & \text{if } e(s_t, g_1) < 0.1, \\ -d(s_t, g_1) & \text{otherwise} \end{cases} \tag{3.8}$$

Due to the addition of a second goal, the goal space increases by three dimensions as follows

(equation 3.9). As the goal space is part of the full state representation in the algorithm, the dimensionality of the problem increases by three.

Equation 3.9 Goal space for task 2, as a part of the state space

$$g = \{x_{g_1}, y_{g_1}, z_{g_1}, x_{g_2}, y_{g_2}, z_{g_2}\} \tag{3.9}$$

The rest of the state, and action space remain the same throughout this specific task.

3.3.3 Task 3: Reaching with desired velocity

The velocity task is effectively a reaching task with the addition of a velocity requirement when the end effector reaches the final goal. This is the third task (task 3). Simulation constraints make it difficult to determine the velocity of the end-point of the arm accurately. Analyses of the simulation showed that observations were mostly made with equal time intervals allowing a simplified representation of velocity, namely the distance travelled per action. As illustrated in equation 3.10, where v is velocity in m/s, Δs is delta distance in metres, Δt is delta time in seconds. It becomes apparent that when Δt stays constant, the only variable that influences the velocity is the distance travelled per observation.

Equation 3.10 Traditional formula for calculating velocity.

$$v = \frac{\Delta s}{\Delta t} \tag{3.10}$$

In simulations, it is often challenging to estimate the simulation time elapsed due to speed-up calculations and varying computational load on the simulation, causing lag or other unforeseen time effects. Therefore, a more simplistic approach that considers the distance travelled per action taken in the environment instead of actually measuring system time is the best alternative.

Consequently, the distance travelled, which will now be referred to as the velocity, will be the constraint to be checked in the reward signal for this task. The reward signal is described in equation 3.11. In which we can see that if the goal is reached with enough velocity, a reward of ten is obtained, and if the goal is reached without enough velocity, a reward of one is obtained instead. The punishment remained unchanged. Furthermore, the state space increased with one dimension to allow for a velocity parameter, as depicted in equation 3.12.

Equation 3.11 Reward signal task 3 final goal.

$$R(s_t, g) = \begin{cases} 10 & \text{if } e(s_t, g_1) < 0.1 \text{ and} \\ & v(s_t, s_{t-1}) > 0.07 \\ 1 & \text{if } e(s_t, g_1) < 0.1 \text{ and} \\ & v(s_t, s_{t-1}) < 0.07 \\ -d(s_t, g_1) & \text{otherwise} \end{cases} \tag{3.11}$$

Equation 3.12 State space for task 3.

$$s_t = \{\theta_1, \theta_2, \theta_3, x_s, y_s, z_s, v\} \tag{3.12}$$

3.4 Consequential action selection

Consequential actions are by definition actions that lead to quicker and better reward jumps in the utility landscape, especially in initial training phases when the environment remains undiscovered. Therefore, the actions should result in better information regarding the environment of the agent, this means they will result in more effective exploration in the early phases of training. Selecting only consequential actions is difficult, as it requires a formal definition that can be calculated, e.g., leading to something that can categorize actions into consequential and non-consequential. Note that predicting consequential actions will always remain an approximation and can not yet be determined as easily as described. Different approaches have been studied to see how we can analyze the actions leading to better exploration and rewards. As a means to quantify the meaningfulness of actions, we consider the difference in reward Δr and the difference in state Δs. Different approaches to formalizing consequential actions have been used in action selection, and their methods are described below.

3.4.1 Definitions

Different definitions have been examined to determine what fits a consequential action best in order to select them later during training. The main goal of this study is to get rid of Brownian motion. Brownian motion is defined as actions resulting in small state differences, e.g., small $(\Delta x, \Delta y, \Delta z)$ which can be measured in Euclidean distance. Therefore, one approach is to define a consequential action as a non-Brownian action, meaning that the requirement is to determine the distance travelled as a result of taking an action and using a threshold to categorise it as Brownian and non-Brownian. Nonetheless, state differences only realise a better method for exploration, but not necessarily for higher rewards. Therefore, the distance between the goal and the end effector can also be utilised to determine the meaningfulness of an action. Equation 3.13 illustrates an approach where only the state differences are considered for classification. Note that in this non-Brownian definition of a consequential action, the threshold remains constant. Let $C(s_t, a_t)$ be a boolean value that shows if an action is consequential or not. Assume that $f(s_t, a_t)$ is a state approximator that can determine the next state (s_{t+1}) from the current state (s_t) and the action (a_t).

Equation 3.13 Definition 1: non-Brownian actions.

$$C(s_t, a_t) = \begin{cases} 1 & \text{if } d(s_t, f(s_t, a_t)) > 0.05 \\ 0 & \text{otherwise} \end{cases} \tag{3.13}$$

The second definition also considers the difference in reward as a result of a given action. The reward mostly resembles the distance from the end effector to the goal, so the delta reward (Δr) effectively models if the end effector moves in the goal's direction. Note that this delta reward is different from the actual reward as it does not provide rewards of 1 or 10 when a task has been completed successfully, it remains the distance from the end effector to the goal. The absolute value of the delta reward was taken so that actions moving further away from the goal are also considered consequential since moving farther away from the goal gives meaningful information about the goal's location too. This is described in equation 3.14, here it is also assumed that $f(s_t, a_t)$ is a state approximator that can determine the next state (s_{t+1}) from the current state (s_t) and the action (a_t).

Equation 3.14 Calculating the delta reward of a given action.

$$|\Delta r| = |d(s_t, g) - d(f(s_t, a_t), g)| \tag{3.14}$$

From the equation that describes the delta reward, we can derive a new definition for a consequential action, depicted in equation 3.15. Whereas ω is a linearly decreasing threshold dependent on the number of epochs, as illustrated in equation 3.16, where n is the current epoch. This threshold is constructed this way because the constraint is too big for the model to converge, so it is only used to stimulate exploration in the early phases.

Equation 3.15 Definition 2: consequential actions based on goal location

$$C(s_t, a_t) = \begin{cases} 1 & \text{if } d(s_t, f(s_t, a_t)) + |\Delta r| > \omega \\ 0 & \text{otherwise} \end{cases} \tag{3.15}$$

Equation 3.16 Omega threshold linear decay, dependent on epoch.

$$\omega = 0.15 - (\frac{0.15}{1000} * n) \tag{3.16}$$

The goal of this study is to select the consequential actions during training, to examine whether a heuristic that prioritizes bigger movements aids learning. Nonetheless, for using any of these definitions, we need to be able to predict the consequences of an action on the state. In other words, predicting the exact next state from a given state and action. Thus, a form of state approximation is necessary, e.g., a function for $f(s_t, a_t)$. This is described in the next subsection.

3.4.2 State approximation

Here we describe two methods for approximating the next state given the current state and an action. The first method uses a Multilayer Perceptron (MLP) that learns to predict the next state from past experiences. The MLP consists of 5 layers, one input layer, three hidden layers, and an output layer. A dropout is used between the hidden layers. The Adam algorithm was used to backpropagate the loss (Kingma & Ba, 2014) and all hidden layers used the ReLu activation. More information regarding the details of this network is described in Appendix C.

The second model uses forward kinematics, which is essentially a mathematical formula for calculating the position of the end-effector using the angles of the joints and arm lengths. Forward kinematics is the opposite of inverse kinematics, in which we need to calculate the angles of the joints in order to achieve a given end effector position. In forward kinematics, we do the opposite; we calculate the position of the end effector with the given joint angles and arm lengths.

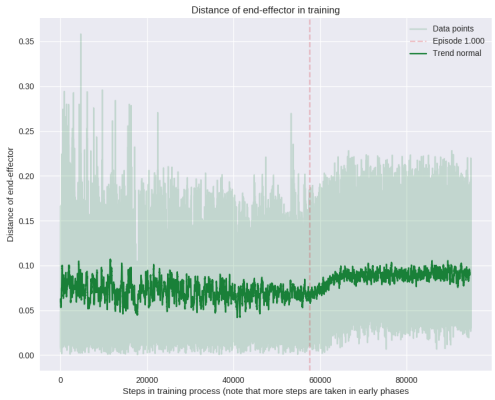

Figure 4.1: Euclidean distance of consecutive end effector states in a line plot. Trend is determined by a rolling average of 300. Red dotted line is exact moment of model convergence.

4 Results

4.1 Brownian motion

Learning trajectories of the traditional Soft Actor-Critic algorithm have been extensively analysed with the aim of defining the Brownian motion that can be observed during learning phases. The distance observed between states presents a method to investigate actions and their effects on the state space. Figure 4.1 shows the Euclidean distance measured between end-effector positions at each time step of the first task with the traditional SAC algorithm. A clear jump in the distance in the rolling average of 300 can be observed when we look at the time when the model starts to converge (from epoch 1000). Additionally, the lower bound of the distance seems to be elevated, meaning that a functioning model does not prefer actions that lead to minimal state distances. It is suggested that these actions result in the Brownian motion present in the early learning phases. Figure 4.2 shows the same graph but is illustrated as a density plot. The first 500 epochs of training (blue) and the last 500 epochs of training (green) are compared. Figure 4.2 shows a clear difference in the distance the end effector travels per action taken. The Brownian motion, state deltas measured in the distance, can be observed by the presence of actions resulting in distances smaller than 0.5.

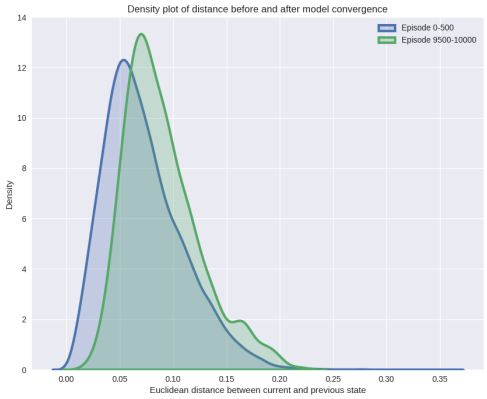

Figure 4.2: Euclidean distance of consecutive end effector states in a density plot. Green equals distribution of distances in the last 500 epochs of training. Blue equals distribution of distances in the first 500 epochs of training.

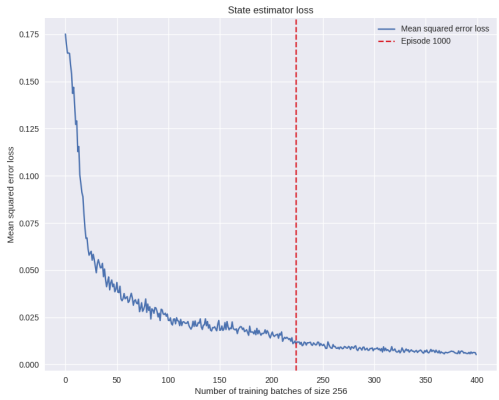

Figure 4.3: MSE loss of MLP for state estimation for 400 batches of size 256. The red dotted line shows the time of SAC convergence for comparison.

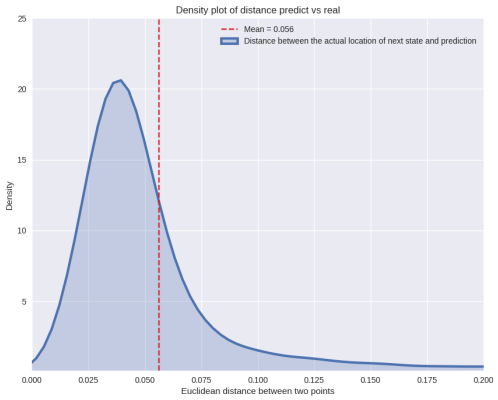

Figure 4.4: Density plot of Euclidean distance between the prediction of the MLP and the actual next state. The red dotted line shows the distance mean.

4.2 Multilayer perceptron

The Multilayer Perceptron (MLP) used to estimate the next state performed as depicted in the figures below. The mean squared error (MSE) loss is plotted in Figure 4.3.

Notice that the state approximator is still learning when the SAC model converges (1000 epochs). The testing performance measured in Euclidean distance is presented in Figure 4.4. This shows a mean of 0.0564 in terms of distance between the actual location of the end effector and the prediction provided current location and action. A few outliers, mostly wildly inaccurate predictions, cause the mean to be a bit higher than the peak of density. One can see from the density plot that most distances are within a range of 0.01 and 0.06. In order to compare the order of magnitude of this result, Brownian motion is considered to result in state differences smaller than 0.05.

4.3 Model performance

The traditional Soft Actor-Critic algorithm was not as successful as anticipated. The success rate can be observed in Figure 4.5 and Table 4.1 for all of the three tasks. The first task (reaching task) is one of the easiest problems within the reinforcement learning environment, as suggested by its quick convergence (1000 epochs) to a 100% success rate. Nonetheless, the performance on the other tasks is not as high as expected for a current state-of-the-art algorithm such as SAC. The second task (via-point task) converged to a success rate of 97% after approximately 1000 epochs. The third task (velocity task) was the most difficult for the algorithm. It converged to a 93% success rate after about 4000 epochs.

Consequential action-selection within the Soft Actor-Critic algorithm provided the results depicted in Figure 4.6 and Table 4.2. Forward kinematics was used to determine the effect of an action on the current state. The first definition was used to select consequential actions (equation 3.13). Results show that a 100% success rate is still reached, however, after 1250 epochs. A 250 epoch difference (25%) for the altered action

Table 4.1: Success rate and epochs before model convergence of all three tasks with traditional SAC algorithm.

	Task 1	Task 2	Task 3
Success rate	100%	97%	93%
Epochs converge	1.000	1.000	4.000

Figure 4.5: Success rate of all three tasks using the Soft Actor-Critic algorithm. Green is the first task, blue the second task, and red the third task.

	Conseq. model	Normal model
Success rate	100%	100%
Epochs	1.250	1.000

Table 4.2: Success rate and epochs before model convergence of consequential action selection and normal action selection, using the SAC algorithm.

selection. This difference is better illustrated in the difference graph in Figure 4.7. This clearly shows advantages for consequential action selection in the early training phases. Nonetheless, a clear disadvantage in the later stages of training. The results of definition 2 are shown in Appendix D.

5 Discussion

5.1 State approximation

The method in which this study tested state approximation using a traditional Multilayer Perceptron (MLP) was not accurate enough to be used in consequential action selection. Figure 4.4 illustrates that the distance between the prediction and the actual state had a mean of 0.056. For comparison, Brownian actions are defined as actions resulting in state differences smaller than 0.05. The accuracy of the MLP is just an order of magni-

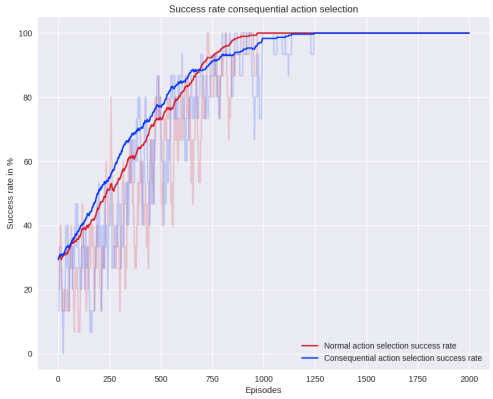

Figure 4.6: Success rate of traditional SAC versus consequential action-selection model (definition 1). Blue is consequential action selection, red is traditional action selection.

Figure 4.7: Success rate difference of traditional SAC versus consequential action-selection (definition 1).

tude too large. This means that the MLP state approximator is not precise enough to distinguish between Brownian, non-Brownian actions, but also consequential and non-consequential actions. Explanations for the under-performing MLP solution might be a non-complete state representation, from which it is difficult to estimate the exact next state. Such as not including rotations and arm lengths of the robot. However, the MLP was trained on 10.000 epochs which equals around 100.000 training examples for predicting the next state. Unfortunately, traditional machine learning could not precisely recognise the patterns in that data. This emphasises the unforgivably complex robotics environment once again.
Predicting states is a bit more straightforward when they can be calculated. Given an action (a_t) and a state (s_t), we can apply the change in angles on the joints and using the arm lengths we can calculate the location of the end-effector quite easily. This process is referred to as forward kinematics. Nonetheless, this will increase computational load if performed often and repeatedly, especially in a sped-up simulation. Note that this approach will lead to 100% accuracy in approximating the next state. This approach offers no generalisable solution that can be used throughout multiple reinforcement learning problems since it utilises the specific methods of this strict environment in order to determine the next state. Imagine a different robot with more joints, different arm lengths, or even a different reinforcement learning problem, and the whole heuristic becomes useless. Therefore, a manually hand-crafted solution is far from ideal, but in this study, it is merely used to analyse its effect when utilised in consequential action selection. A learning solution is desired when it comes to solving problems in the field of reinforcement learning and not just in a specific environment. The motivation behind using machine learning solutions is often the ability to adapt to different environments, and offering unique solutions is, therefore, counter-intuitive. Nonetheless, in this paradigm, it is challenging to create a well-functioning machine learning solution as we are talking about an instance of simultaneous learning, whereas the state approximator should outperform the SAC algorithm to improve its decision-making to select meaningful actions. Since the SAC algorithm is relatively sophisticated, it becomes difficult to construct a better learning solution in a part of the task, e.g., approximating the next state. Since the accuracy of the MLP was insufficient and learning required ten times as much data as the SAC required to solve the whole task, it was decided that constructing an acceptable machine learning solution was not realistic within the scope of this study.

5.2 Brownian motion

Brownian motion can be observed in the robotics environment using reinforcement learning. It is visually apparent from the small movements that the robot often coordinates in the early phases of learning. Figure 4.2 and 4.1 show that small effect actions are indeed present up until episode 1000 (time of convergence). These actions result in state differences, measured in Euclidean distance, smaller than 0.05. The notion of Brownian motion can thus be confirmed in a robotics reinforcement learning paradigm. It is hypothesized that more reinforcement learning problems will show similar behaviour, which can be confirmed with further research.

Using forward kinematics for approximating the next state, we can use a form of consequential action selection to prevent Brownian motion from happening, using definition one illustrated in equation 3.13. Figure 4.6 shows the success rate over the episodes of the SAC algorithm with and without consequential action selection. The algorithm still converges a bit later; however, this illustrates that Brownian motion is not necessary for learning a task successfully.

5.3 Consequential actions

Consequential actions are ultimately actions that encourage exploration and obtain higher rewards as a result. One might argue that an optimal policy would only pick consequential actions instead of actions that would, for example, lead to the Brownian motion. This is true. However, an optimal policy might only arise after a while of training, and consequential actions might be just very informative in the early phases of training. So, selecting consequential actions in early phases is hypothesised to increase training efficiency. Note that a consequential action can have different

definitions, as illustrated in Section 3.4.1. This study aims to provide a heuristic that can define consequential actions properly as a means to investigate whether or not selecting them improves early performance.

In Figure 4.7, we can see a reliable difference in the early training phases by selecting actions that will result in more considerable state differences. The consequential action selection model has a higher success rate in the initial learning phase. However, it drops drastically when the model starts converging to a 100% success rate. As a result, the standard action selection model converges quicker. Note that the form of action selection considered in this unique example only utilises the state differences (definition 1). This means that the type of consequential action is non-Brownian, as described in equation 3.13.

Results from definition 2 (equation 3.15) can be found in Appendix D. This definition showed similar behaviour, concluding that a delta reward does not contribute to a more efficient method of defining the meaningfulness of an action. Better definitions for consequential actions might still exist, ones that also improve the convergence times. Nonetheless, this study has not encountered these definitions, but this proposal incentivises further research.

5.4 Problems with RL

In reinforcement learning for robotics, many problems remain. States often consist of inaccurate representations, training takes too long (even in sped-up simulations), and the current state-of-the-art algorithm (SAC) is not even able to thoroughly learn tasks that extend from the traditional reaching task. Literature in RL is primarily optimistic about its ability to solve many tasks. However, in practice, these solutions often require time-consuming fine-tuning of reward signals, hyperparameters, observation settings, and other environmental changes. This hand-crafting until the reinforcement learning algorithm performs well does not offer a generalizable learning solution that is required to replace industrial and service robots to learn new types of movement. A solution that is able to do more than one task is necessary without making any changes to the algorithm every time. To summarize, reinforcement learning does not perform well enough yet to be effectively used in real-world applications, such as in the field of robotics, which could reap the benefits.

5.5 Future research

The problems with RL result in the incentive for many future research opportunities. This study concluded that Brownian motion within a reinforcement learning robotics paradigm exists and prevents the current state-of-the-art algorithm from achieving better exploration in the early phases of training. Nonetheless, the constraint of not selecting actions that result in Brownian motion is a disadvantage for later stages of training resulting in later convergence. As mentioned earlier, the definition of a consequential action selection is open for further research. There might be methods to achieve even better performance by picking actions with different properties not presented in this study.

Furthermore, it might be exciting to see what an entirely different kind of action selection in reinforcement learning would lead to. For example, in a robotics environment, representing the actions by parameters of an inverse kinematics solver such as MoveIT—shifting the whole reinforcement learning paradigm to higher-level aspects. This is likely another viable alternative to preventing Brownian motion that is now proven to slow learning in initial phases.

References

Ajaykumar, G., Steele, M., & Huang, C.-M. (2021). A survey on end-user robot programming. *ACM Computing Surveys (CSUR)*, *54*(8), 1–36.

Andrychowicz, M., Wolski, F., Ray, A., Schneider, J., Fong, R., Welinder, P., ... Zaremba, W. (2017). Hindsight experience replay. *Advances in neural information processing systems*, *30*.

Ballestar, M. T., Díaz-Chao, Á., Sainz, J., & Torrent-Sellens, J. (2021). Impact of robotics on manufacturing: A longitudinal machine learning

perspective. *Technological Forecasting and Social Change*, *162*, 120348.

Chitta, S. (2016). Moveit!: an introduction. In *Robot operating system (ros)* (pp. 3–27). Springer.

Dankwa, S., & Zheng, W. (2019). Twin-delayed ddpg: A deep reinforcement learning technique to model a continuous movement of an intelligent robot agent. In *Proceedings of the 3rd international conference on vision, image and signal processing* (pp. 1–5).

Haarnoja, T., Zhou, A., Abbeel, P., & Levine, S. (2018). Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International conference on machine learning* (pp. 1861–1870).

Karatzas, I., & Shreve, S. E. (1998). Brownian motion. In *Brownian motion and stochastic calculus* (pp. 47–127). Springer.

Kase, K., Matsumoto, N., , & Ogata, T. (2021). Leveraging motor babbling for efficient robot learning. *Journal of Robotics and Mechatronics*, *33*(5), 1063-1074. doi: 10.20965/jrm.2021.p1063

Kingma, D. P., & Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Kober, J., Bagnell, J. A., & Peters, J. (2013). Reinforcement learning in robotics: A survey. *The International Journal of Robotics Research*, *32*(11), 1238–1274.

Luo, S., Kasaei, H., & Schomaker, L. (2020). Accelerating reinforcement learning for reaching using continuous curriculum learning. In *2020 international joint conference on neural networks (IJCNN)* (pp. 1–8).

Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., ... Kavukcuoglu, K. (2016). Asynchronous methods for deep reinforcement learning. In *International conference on machine learning* (pp. 1928–1937).

Mussa-Ivaldi, F. A. (1999). Modular features of motor control and learning. *Current opinion in neurobiology*, *9*(6), 713–717.

Nachum, O., Norouzi, M., & Schuurmans, D. (2016). Improving policy gradient by exploring under-appreciated rewards. *arXiv preprint arXiv:1611.09321*.

Pedersen, M. R., Nalpantidis, L., Andersen, R. S., Schou, C., Bøgh, S., Krüger, V., & Madsen, O. (2016). Robot skills for manufacturing: From concept to industrial deployment. *Robotics and Computer-Integrated Manufacturing*, *37*, 282–291.

Qian, W., Xia, Z., Xiong, J., Gan, Y., Guo, Y., Weng, S., ... Zhang, J. (2014). Manipulation task simulation using ros and gazebo. In *2014 ieee international conference on robotics and biomimetics (robio 2014)* (pp. 2594–2598).

Quigley, M., Conley, K., Gerkey, B., Faust, J., Foote, T., Leibs, J., ... others (2009). Ros: an open-source robot operating system. In *Icra workshop on open source software* (Vol. 3, p. 5).

Schulman, J., Levine, S., Abbeel, P., Jordan, M., & Moritz, P. (2015). Trust region policy optimization. In *International conference on machine learning* (pp. 1889–1897).

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., & Klimov, O. (2017). Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*.

Sutton, R. S., & Barto, A. G. (2018). *Reinforcement learning: An introduction*. MIT press.

Wolpert, D. M., & Kawato, M. (1998). Multiple paired forward and inverse models for motor control. *Neural networks*, *11*(7-8), 1317–1329.

Zhu, Y., Wang, Z., Merel, J., Rusu, A., Erez, T., Cabi, S., ... others (2018). Reinforcement and imitation learning for diverse visuomotor skills. *arXiv preprint arXiv:1802.09564*.

A Appendix: Workspace constraints

End effector workspace UR5e		
Coordinates of end effector	Min	Max
x	0.01	0.8
y	-0.3	0.8
z	0.242	0.842

Table A.1: Minimum and maximum position of end effector, tested so that the arm can reach all points within this space.

Joints UR5e		
Joints	Min	Max
1	-0.8	1.3
2	-2.5	-0.2
3	0	2.5

Table A.2: Minimum and maximum position of joint angles in radians, tested so that the arm can not collide with itself.

B Appendix: Soft Actior-Critic implementation details

Hyper-parameters for both actor and critic	
α	0.2
γ	0.99
τ	0.01

Table B.1: The hyper-parameter values for both the actor and the critic.

Critic architecture		
Layers	Nr. of neurons	Activation
Input layer	state + goal Task 1: 9 Task 2: 12 Task3: 10	N/A
Hidden layer 1	512	ReLu
Hidden layer 2	256	ReLu
Hidden layer 3	64	ReLu
Output layer	1 (state value)	N/A

Table B.2: Critic architecture, note that two critics are made in the algorithm to mitigate positive bias in the Q-value. The minimum of both is taken and used in the loss function of the actor network.

Critic details	
Optimizer	Adam
Batch size	256
Loss	Equation 2.4

Table B.3: Critic further details.

Actor architecture		
Layers	Nr. of neurons	Activation
Input layer	state + goal Task 1: 9 Task 2: 12 Task3: 10	N/A
Hidden layer 1	512	ReLu
Hidden layer 2	256	ReLu
Hidden layer 3	64	ReLu
Output layer 1	3 (Mean: action size)	N/A
Output layer 2	3 (Log std: action size)	N/A

Table B.4: Actor architecture, note that hidden layer 3 is connected to output layer 1 and output layer 2, to get the mean separately from the logarithm of the standard deviation. This is done so that a distribution can be made from which can be sampled as a means to increase stochasticity.

Critic details	
Optimizer	Adam
Batch size	256
Loss	Equation 2.6

Table B.5: Actor further details.

C Appendix: MLP implementation details

MLP architecture			
Layers	Nr. of neurons	Activation	Dropout
Input layer	9 (state+action)	ReLu	N/A
Hidden layer 1	50	ReLu	N/A
Hidden layer 2	25	ReLu	0.1
Hidden layer 3	5	ReLu	0.1
Output layer	3	ReLu	N/A

Table C.1: Architecture details of MLP state estimator

MLP details	
Optimizer	Adam
Batch size	256
Number of training batches	400
Number of test samples	10.000
Dropout layers	2
Loss	MSE

Table C.2: Further details of MLP

D Appendix: Results of consequential action selection definition 2

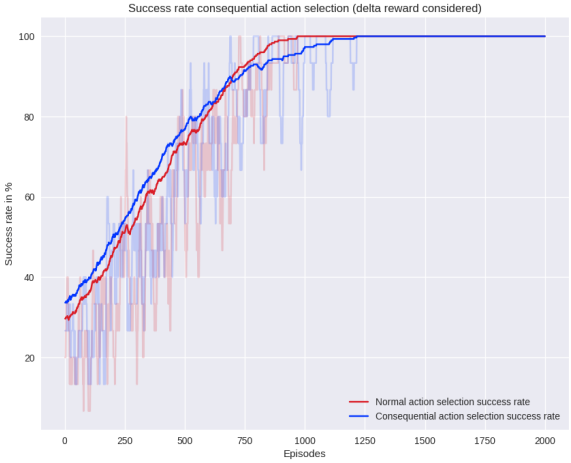

Figure D.1: Success rate of traditional SAC versus consequential action-selection model (definition 2). Blue is consequential action selection, red is traditional action selection.

Figure D.2: Success rate difference of traditional SAC versus consequential action-selection (definition 2).