Technical University of Munich

Department of Informatics

Bachelor's Thesis in Informatics

# Comparison of Reinforcement Learning for Direct and Indirect Locomotion Control in Target Tracking with Snake-like Robots

Julian Schmitz

Technical University of Munich

Department of Informatics

Bachelor's Thesis in Informatics

# Comparison of Reinforcement Learning for Direct and Indirect Locomotion Control in Target Tracking with Snake-like Robots

# Vergleich von bestärkendem Lernen für direkte und indirekte Fortbewegungssteuerung bei Zielverfolgung mit schlangenartigen Robotern

| | |
|---|---|
| **Author:** | Julian Schmitz |
| **Supervisor:** | Prof. Dr.-Ing. habil. Alois Knoll |
| **Advisor:** | Zhenshan Bing, M.Sc. |
| **Submission:** | Oktober 8, 2018 |

I confirm that this bachelor's thesis is my own work and I have documented all sources and material used.

Munich, Oktober 8, 2018

*(Julian Schmitz)*

# Abstract

Snake-like robots have long and thin, often limbless bodies that enable them to swim and to move through rubble as well as to overcome obstacles. In future, such robots could provide important help in scenarios where human presence is either adverse due to dangerous environments or impossible as for example in tight spaces. However, the complexity of their movement and the high degrees of freedom make finding versatile and robust locomotion control a difficult task. Training Reinforcement Learning (RL) agents to control snake-like robots in target tracking scenarios could help to improve their practical operation in various challenges.

To explore techniques to control snake-like robots by an RL agent, a target tracking scenario with multiple target trajectories was implemented in this work. The planar modular snake-like robot can perceive the target by a vision sensor in its first module and laterally rotate the joints between its modules to move. The goal to achieve robust target tracking was approached by two different concepts. Firstly, the RL agent was trained to indirectly control the robot by steering the direction of a human-crafted slithering gait. Secondly, the RL agent was trained to track the target by directly controlling the joints of the snake-like robot.

Both approaches successfully tracked the target on the training trajectory. However, on the evaluation trajectory with a sharp 90° turn, the indirect control agent only performed a mean of 54.32% successful tracking episodes, whereas the direct control agent has proven to be more robust and achieved a tracking accuracy of 93.84%. It remains to be studied if dynamically changing the parameters that characterize the slithering gait such as the frequency and amplitude of its swings would improve the indirect control agent. In the long term, an important goal is to apply the same concepts to snake-like robots with three-dimensional movement.

## Zusammenfassung

Schlangenartige Roboter verfügen über lange, flexible Körper, die es ihnen sowohl ermöglichen zu schwimmen, als auch, sich durch Schutt und über Hindernisse zu bewegen. In Zukunft könnten solche Roboter daher eine wichtige Hilfe in Krisenszenarien leisten, in denen Menschen aufgrund gefährlicher Umgebungen oder zu enger Räume nicht selbst eingreifen können. Sie bewegen sich mithilfe ihrer vielen Gelenke auf komplexe Weise fort, weshalb das Erarbeiten einer anpassungsfähigen und robusten Fortbewegungssteuerung eine schwierige Aufgabe darstellt. Die praktische Anwendung schlangenartiger Roboter könnte verbessert werden, indem Reinforcement Learning (RL) Agenten zur Steuerung dieser Roboter in Zielverfolgungsszenarien trainiert werden.

Um Techniken zur Steuerung der Roboter durch RL Agenten zu untersuchen, wurde in dieser Arbeit ein Zielverfolgungsszenario mit unterschiedlichen Pfaden implementiert. Der modulare schlangenartige Roboter sieht das Ziel mithilfe eines Vision-Sensors und rotiert die Gelenke zwischen seinen Modulen lateral um sich planar fortzubewegen. Das Ziel robuster Zielverfolgung wurde anhand von zwei verschiedenen Ansätzen verfolgt. Zunächst kontrollierte der RL Agent den Roboter indirekt, indem er die Richtung einer Formel für schlängelnde Fortbewegung steuerte. Im zweiten Ansatz verfolgte der RL Agent das Ziel über die direkte Kontrolle der Gelenke des schlangenartigen Roboters.

Auf dem Trainingspfad verfolgten beide Ansätze das Ziel erfolgreich. Jedoch erzielte der indirekte Agent lediglich eine Verfolgungsrate von 54.32% auf dem Evaluationspfad mit scharfer 90° Kurve. Der direkte Agent zeigte sich robuster und erreichte in diesem Szenario noch eine Genauigkeit von 93.84%. In Zukunft bleibt zu untersuchen, ob eine dynamische Anpassung der Parameter der Fortbewegungsformel des indirekten Agenten (wie beispielsweise die Frequenz und Amplitude der Schwünge) dessen Genauigkeit verbessern würde. Des Weiteren ist es ein wichtiges Ziel, dieselben Konzepte für schlangenartige Roboter mit dreidimensionaler Fortbewegung zu testen.

# Contents

# Glossary

| | |
|---|---|
| ACER | Actor-Critic with Experience Replay |
| ACM | Active Cord Mechanism |
| ANN | Artificial Neural Networks |
| CMU | Carnegie Mellon University |
| CNN | Convolutional Neural Network |
| CoRL | Conference on Robot Learning |
| EPFL | Ecole polytechnique fédérale de Lausanne |
| ILSVRC | ImageNet Large Scale Visual Recognition Competition |
| MDP | Marcov Decision Process |
| RL | Reinforcement Learning |

# List of Figures

# 1 Introduction

In recent years, Machine Learning (ML) algorithms have achieved promising breakthroughs in many fields, from playing games [1] to describing images [2]. Self-driving cars are testing to drive autonomously in complex urban environments without human intervention and already transport passengers in pilot programs [3, 4]. Specifically, deeply-layered *Artificial Neural Networks* (ANNs) have been successful in surpassing past achievements and even human-level performances in some isolated tasks such as image classification [5, 6]. These artificial neurons, that are inspired by biological brain cells, process information and pass it on to further neurons via weighted connections. New types of neurons, architectures and optimization methods were developed and significantly improved the performance. For instance, the development of *Convolutional Neural Networks* (CNNs) led to a rise in image recognition accuracy. In the *ImageNet Large Scale Visual Recognition Competition* (ILSVRC) the task is to classify pictures into one of 1000 categories. Human-level error rates of 5,1% at this data set have been surpassed since a CNN from Microsoft Research won the competition in 2015 with an error rate of 3.57% [7, 8].

Moreover, *Reinforcement Learning* (RL) researchers have achieved promising breakthroughs in problems that previously posed a great difficulty for Machine Learning algorithms and any way of automation in general. RL algorithms learn by trial and error and discovery. They optimize their action policy to maximize a feedback signal and learn similarly to advanced animals. In October 2015, Deepmind's reinforcement learning algorithm *AlphaGo* [9] was the first computer program to defeat a world champion at the Chinese board game of Go. Due to the high number of possible moves in this game, it was expected that it would take algorithms at least another decade to reach human levels as evaluating the feasible set of actions would require high amounts of computational power [9]. In their "Mastering" of the game of Go, Deepmind's AlphaGo used a combination of Monte Carlo tree search and reinforcement learning instead of traditional exhaustive search [9]. The algorithm started to learn by watching human players and later improved by playing against humans. It was of interest to the Go community not only because it defeated the world champion but as well because it even utilized moves that were priorly unpopular to the human player base and "overturned hundreds of years of received wisdom" [10]. The successor algorithm *AlphaGo Zero* defeated the original AlphaGo and moreover, was trained without any prior human knowledge, solely by self-play [11, 12]. In another highly complex environment, the RL algorithm *OpenAI Five* learned to play the computer game *Dota 2* above semi-professional level only utilizing the displayed pixels and performing actions like a human player [13]. This game is one of the most complex eSports games in the world with year-round training professionals and had a prize pool of $40M in 2018, the largest of all eSports games [1]. At the Dota 2 world championship, OpenAI Five played against two professional teams and lost both matches against highly strategic plays by the human teams after defeating semi-professional teams before [14, 13]. OpenAI Five consists of five ANNs that respectively cooperate with the 4 other Artificial Neural Networks in a team and apply highly strategic decisions. Moreover, the algorithm of OpenAI Five was trained exclusively via self-play like AlphaGo Zero [15, 13].

The ability to learn from self-play or discovery without prior human knowledge could potentially prove very useful in fields where human knowledge is either costly, hard or even impossible to obtain. Robotic locomotion poses one such field, where human knowledge in the form of locomotion control algorithms becomes increasingly hard to obtain due to their complexity and many degrees of freedom. Deepmind has achieved robust locomotion skills in simulations for a simple walker, a quadruped and a humanoid model using RL with simple rewards by training them in rich environments [16]. The models learn to overcome different obstacles in their environment altering their locomotion autonomously. They jump or crouch with different intensity and locomotion for easy to overcome obstacles than for ones that are hard to overcome [16]. By observing nature and animals that are close to the robotic models, researchers can craft locomotion algorithms that mimic biological locomotion. However, to achieve sufficient robustness for difficult terrain, obstructing obstacles, narrow passages or even things falling at the robot while moving is a complex task.

Snake-like robots are a field of robots inspired by the body shape and locomotion found in biological snakes and their closer relatives such as lizards and amphibians. Their complex movements that utilize the many internal degrees of freedom in their long and often limbless bodies are still studied. The distinct features of snake-like body models in addition to the capability of autonomous locomotion would significantly improve the application of such robots on tasks such disaster rescue, military surveillance, and factory maintenance [17, 18]. One pioneer in the development of snake-like robots is Shigeo Hirose who built his first *Active Cord Mechanism* (ACM) snake-like robot [19] in 1972 and still develops his robots further. His first model was restricted to planar movement whereas his later snake-like robots can also perform three-dimensional movement as well as swim through water [19]. When a Tsunami hit Fukushima's power plant in 2011, his later ACM-R5 snake-like robot was called the "only robot capable" of removing radioactive contaminants inside nuclear reactors and dismantling the nuclear reactors [20]. Furthermore, the snake-like robot of the Carnegie Mellon University (CMU) assisted the Mexican Red Cross when an earthquake hit Mexico City in 2017 and passed through the rubble of a collapsed building in order to find survivors [21].

These manifold tasks and environments respectively require specialized forms of locomotion. Many snake-like robots are controlled via scripted or parameterized gait equations. After changes in the environment, a scripted gait might not work and programming a new scripted gait might be necessary, whereas a parameterized script is able to move in more diverse environments as the operator is able to change the behavior dynamically [22]. This adaptability makes parameterized gaits desirable over static scripted gaits [22]. The objective of this work is to explore if, in a similar way, the application of RL algorithms could potentially improve autonomy and robustness of locomotion over the use of more static types of locomotion that are not able to adapt to their environment.

Therefore, in this work, a target tracking scenario for a planar snake-like robot was built in the robot simulator V-REP [23]. In order to facilitate the RL algorithm *Proximal Policy Optimization* (PPO) to perform locomotion control for a snake-like robot, two experiments were developed. In the indirect locomotion control experiment, the RL algorithm learned to indirectly steer the direction of movement of a slithering gait equation for locomotion control and thus rely on human-crafted locomotion. In contrast, for the direct locomotion control experiment, the RL algorithm learned to perform a similar locomotion by directly controlling the robot's joints to move and track the target without prior knowledge. The resulting agents are discussed and compared for target tracking accuracy on evaluation tracks.

The thesis is structured as follows: In chapter 2, a short introduction into finite Markov Decision Process (MDP) as well as reinforcement learning is given. An overview of some developed snake-like robots and of use of RL with combination with robotics is given in chapter 3. Chapter 4 describes the simulation environment as well as the different experiments and their evaluation methods. In chapter 5, the results are evaluated and compared. Chapter 6 summarizes and discusses the work and outlines potential future work.

# 2 Theoretical Background

This chapter will give a short introduction to the basis of the technological concepts used in this work. For an extensive in-depth explanation of the underlying concepts and application of Reinforcement Learning it is recommended to refer to Sutton and Barto [24] as well as to the second edition of Sutton and Barto [5].

## 2.1 Machine Learning

Most *Machine Learning* algorithms can be split into the main categories of *Supervised Learning*, *Unsupervised Learning*, *Semi-Supervised Learning* and *Reinforcement Learning.* These categories differ in the way that information about the data set is provided to the model and thus as well in their purpose and application. In Supervised Learning, each data point has a corresponding label and the algorithm tries to generalize from its training data. Typical tasks are regression or classification. In contrast, Unsupervised Learning works on data without labels and is typically used for clustering or association tasks. It is used to detect patterns and group similar data points together. Semi-Supervised Learning may be of use if small parts of the data are labeled and the majority of the data is unlabeled. Finally, reinforcement Learning algorithms obtain no labels, but a *reward* score on their performance. They interact with their environment and develop a behavior to maximize their reward. Reinforcement Learning is best applied if it is easy to assess the outcome of a task but hard to find the perfect way to solve it. This assessment can serve as the reward for a reinforcement Learning that will then try to discover the optimal method for maximizing this reward. In interactive problems, Supervised Learning alone is not adequate as it may not be feasible to label the desired behavior in a way that the data set is representative of all important situations that the agent has to act in [5]. If prior knowledge about the solution to a task is hard to obtain or not available, an agent must be able to learn from its own experience [5].

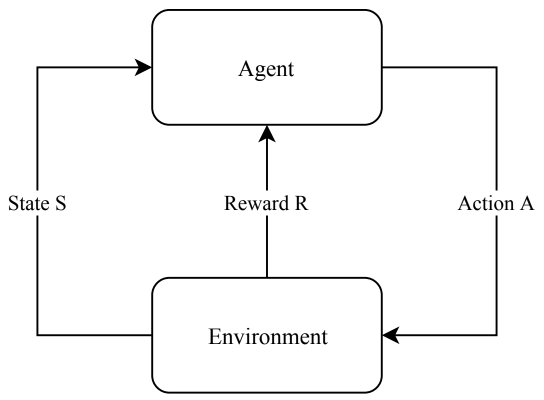

**Figure 2.1:** Interaction between agent and environment in an MDP.

## 2.2 Markov Decision Process

A *Markov Decision Process* (MDP) is a tuple of

- $S$: a set of states $s$
- $A$: a set of actions $a$
- $P_a(s, a, s')$: a probability that action $a$ will transition from state $s$ to state $s'$
- $R_a(s, a, s')$: the reward after transitioning from state $s$ to state $s'$ with action $a$

The concept of MDPs is a fundamental theoretical background for RL. It is ”meant to be a straightforward framing of the problem of learning from interaction to achieve a goal.” [5]. These interactions between the agent and its environment are shown in figure 2.1. The agent perceives the *state*, an observation of the environment, and selects an *action* based on this knowledge. One time step later, the environment returns the *reward* for the consequences of that action as well as the new resulting state. The states may only hold partial information about the environment and the actions might only influence a part of it. But both, the state and the reward, have to be connected to the reward in some way for the agent to be able to learn from this information. The agent discovers the respective rewards for different behaviors in certain states by trial and error. On the basis of this experience, the algorithm updates a policy for choosing actions in order to maximize the reward [5]. This cycle of acting, then perceiving states and obtaining a reward is repeated for every step during the training of an agent.
An important property of an MDP is the Markov Property. A state holds the Markov Property if it retains all relevant past information. An example for a state that has the Markov Property would be a position in checkers. The history of actions that led to this

specific position is lost. However, the history is irrelevant to the further game. The state is "independent of its path" [5]. For RL this means that if a state has the Markov Property, then it is independent of its past and the next state and reward solely depend on the current state and action at the time step $t$ [24].

## 2.3 Reinforcement Learning

Reinforcement Learning (RL) is one subcategory of Machine Learning in which the agent tries to optimize the reward obtained in a Markov Decision Process. The agent interacts with its environment via its actions and can observe the state of the environment. Additionally, the agent obtains feedback in the form of a reward signal which it's trying to maximize. As mentioned before, RL is particularly useful for tasks in which it is easy to assess if a goal was reached, but the best ways to reach it are hard to determine. In highly complex environments it may be unfeasible to label a correct option for every single input, but the final outcome might be easy to label as successful or unsuccessful.
When working with sparse rewards, not all actions result in a reward signal. In contrast to obtaining a reward signal for every state and action pair, the algorithm only gets a reward for finishing a complex task but not necessarily for each required subtask [5]. For example, the final match score of a football game may serve as a reward, but taking a step and shooting might not be immediately rewarded because it is difficult to assess how successful one these subtasks was performed but easy to obtain the final score. When an agent is only getting sparse rewards, it may take longer to succeed in a given task [25, 5]. Given the complex task to move 20 meters to the right, the agent that gets a small positive reward for each meter it progresses will learn a succeeding policy faster than the agent that only gets a reward once it finishes the task. The sparse reward will result in more discovery without a hint of progress. Before obtaining its first reward, the policy would have to be indifferent between moving 15 meters to the left and moving 15 meters to the right, although the latter is closer to the goal. On the other hand, rewards for intermediate steps have to rely on prior knowledge [25]. If, in the same task, there was a slide 3 meters to the left, which would lead to the goal position, this would be a faster way to solve task. That unsuspected method of solving the problem may only be discovered in the case of sparse rewards, as the algorithm in the case of abundant rewards might have decided in favor of exploitation of the obtainable rewards instead of exploration of other ways. In this way, prior knowledge can potentially impair learning the optimal behavior.

# 3 Related Work

In this chapter, an overview of works in development of snake-like robots as well as details of a slithering gait will be given. Furthermore, past works that combine Reinforcement Learning and Robotics are discussed.

## 3.1 Snake-like Robots

Their distinct body type and locomotion gives snakes the capability to move in manifold ways and overcome a lot of different obstacles. Utilizing their many internal degrees of freedom and their long body structure, they can traverse flat and uneven terrain alike, squeeze through tight spaces as well as swim in and under water. To achieve the same capabilities for robots could enable these robots to move through rubble, debris or ruins to find victims in search and rescue scenarios or to be of use in military surveillance and factory maintenance [17]. The development of snake-like robots holds the promising prospect to mimic the manifold movement capabilities of real snakes and assist in a variety of tasks.
Shigeo Hirose developed his snake-like Active Cord Mechanism (ACM) robots since 1972.

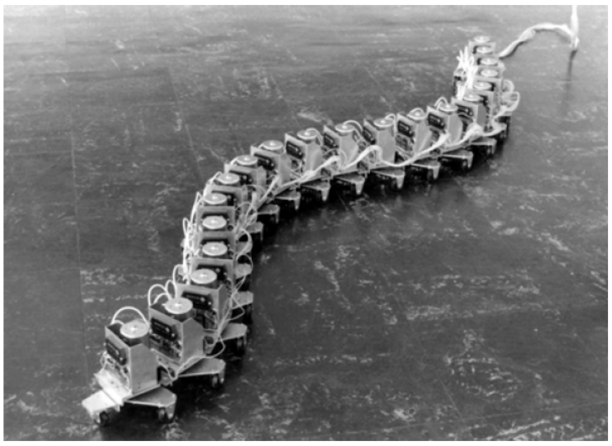

**Figure 3.1:** ACM prototype from 1972 [19].

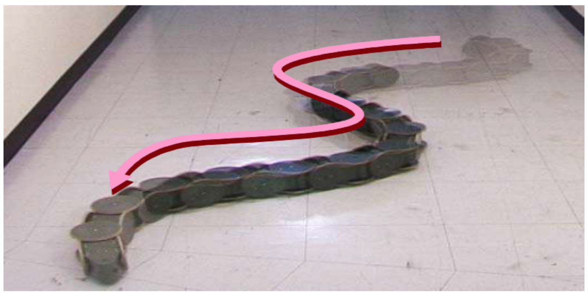

**Figure 3.2:** Serpentine movement [19].

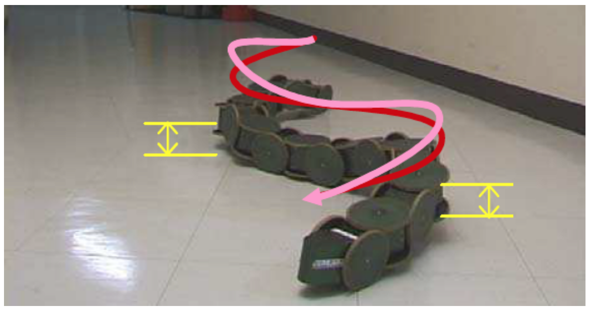

**Figure 3.3:** Sinus-lifting [19].

The prototype model of 1972 is displayed in figure 3.1. He says that the body of snakes can have the "function of an arm" when coiling as well as the "function of legs" when moving by creeping [19]. He created robots consisting of multiple modules with passive wheels and lateral rotatable joints to its neighboring modules. These robots are able to skate on their passive wheels in a two-dimensional movement that he calls *glide propulsion* or *serpentine movement* [19]. The planar serpentine movement is displayed in figure 3.2. The robots achieve this movement by turning their joints and using the friction of the wheels to the ground that works orthogonally to their rolling direction. Moreover, his later robots are able to perform a range of three-dimensional movements [19]. First of all, a motion that is similar to the planar serpentine movement with an additional lifting of certain body parts called *sinus-lifting*, which is closer to the three-dimensional motion of real snakes than serpentine movement and improves movement on slippery floor as well as on surfaces with high friction because of the fewer contact points with the ground [19]. Figure 3.3 illustrates the locomotion of sinus-lifting and the difference in contact points and motion to the serpentine movement. Furthermore, they can perform a caterpillar-like forward movement as well as sidewinding to move sideways and finally, some of his models like the HELIX can also perform swimming through water [19].
The applications for snake-like robots have grown with their capabilities and new types of snake-like robots have been developed that build upon different materials and concepts. In many different countries, researchers are developing distinct snake-like robots, as for example a Japanese snake-like robot for firefighting purposes that flies by means of a propulsion system emitting water [26]. Researchers from Stanford University developed a vine-like robot that moves by growth like fungi or plants via adaption of air pressure inside the robot utilizing a thin skin which is folded inside itself [27]. The use of growth instead of traditional movement brings benefits such as reduced vulnerability to friction and getting stuck. The tip can extend itself further and thus move whether another part of the robot is stuck or not [27]. In another work inspired by nature, researchers of the Ecole polytechnique fédérale de Lausanne (EPFL) have developed a snake-like robot

inspired by the amphibian salamanders. Their *Salamandra Robotica II* utilizes its limbs for locomotion on land and performs a snake-like movement without its limbs, that is inspired by primitive limbless fish, when swimming through water [28]. Furthermore, researchers of the CMU have developed a *Unified Modular Snake Robot* as well as a *Series Elastic Snake Robot* that is able to perform finer torque control than its predecessor [18, 29]. As mentioned in chapter 1, the snake-like robots of the CMU researchers already found use in the disaster rescue task to find survivors in the rubble of collapsed buildings following the earthquake that hit Mexico City in 2017 [21].

## 3.2 Slithering Gait

Their many degrees of freedom and their unique form of propulsion makes control of snake-like robots an "interesting challenge" [30]. Their often limbless bodies are able to perform a wide range of motion. *Slithering*, also called *lateral undulation* or *serpentine movement*, is a forward motion suitable for planar snake-like robots. Other types of movement like *sidewinding* are also common to real snakes, but are not suitable to perform planar movement with passive wheels [17].
The *Slithering Gait* achieves a planar locomotion for snake-like robots utilizing an equation to move joints into a movement inspired by real snakes. The original gait equation was proposed by Hirose [19]. Each joint angle is set by a sinus function with an offset for direction [22]. The outward-facing force of lateral ground friction that works orthogonally to the rolling direction of the passive wheels on the furthest edges propels the robot forward [17, 31]. Figure 3.2 shows Hirose's snake-like robot performing slithering. For the task of target tracking with a vision sensor in the head module, Bing et al. [32] added linear reduction to the body shape and a correction of the head module's angle to always face forwards. The linear reduction results in less swinging motion of the first modules of the robot as well as a more stable geometry [32]. The head correction reduces the swinging of the head module and thus the risk of losing the target out of sight [32].
The resulting locomotion can be modified by altering the dynamic parameters of the gait equation. Table 3.1 displays the parameters that characterize the slithering gait equation as well as their description and their values in this work. Information about the number and length of the modules of the modular snake-like robot is given via $K$ and $m$ respectively. The linear coefficients $y$ and $z$ configure the linear reduction that results in a smaller amplitude for the first joints. Values of $y = 0$ and $z = 1$ cancel the effect of linear reduction. The angular frequency $\omega$ controls the frequency of the sinus function. A higher value for $\omega$ results in more frequent swings of each joint and thus the whole

**Table 3.1:** Parameters that characterize the slithering gait equation and their values used in this work.

| Symbol | Description | Value |
|---|---|---|
| $K$ | Number of joints. | 8 |
| $m$ | Length of each module. | 0.35 |
| $y$, $z$ | Linear coefficients. | 0.5, 0.5 |
| $\delta$ | Phase shift. | $60\pi/180$ |
| $a$ | Amplitude. | $60\pi/180$ |
| $\omega$ | Time frequency. | $2.95\pi$ |
| $b$ | Turning radius offset. | Controlled by agent. |
| $p$ | Smoothness constant | $-0.3$ |

snake body. In a feasible range, a higher $\omega$, ceteris paribus, also results in a faster forward motion. The direction of this movement is set by the offset of bias $b$. The motion of the Slithering Gait follows the path of a circle. The bias $b$ sets the radius of this circle and the sign of $b$ sets the direction to clockwise for a negative $b$ and counter-clockwise for positive $b$. The resulting relation between $b$ and the direction of motion is that a small value such as 0.5 for $b$ leads to a sharp left turn and a value of $-0.5$ correspondingly leads to a sharp right turn. Greater absolute values respectively result in an increasingly straight forward motion. Hence, the importance of the sign diminishes with greater absolute values.
The $K$ joints are enumerated from $i = 0$ to $i = K - 1$. Let $i$ denote the subscript of the joint, $A_i$ denote the amplitude of joint $i$ after applying linear reduction and $t$ denote the time step.

$$A_i = a \cdot \frac{i \cdot y}{K + z} \tag{3.1}$$

$$\phi_i = b + A_i \cdot \cos(w \cdot t - \lambda \cdot i) \tag{3.2}$$

$$\theta_i = \phi_i \cdot (1 - e^{p \cdot t}) \tag{3.3}$$

The joints' angles $\theta_i$ are calculated for all $i \in K$. Equation (3.1) applies linear reduction to the amplitude to obtain the individual amplitude $A_i$ for joint $i$. Next, equation (3.2) the angle $\phi_i$ is calculated as a weighted cosinus with an offset. Finally, the angle $\phi_i$ gets reduced for smoothness in equation (3.3) and the joint's target angle is set to $\theta_i$. The constant $p$ causes the angles to decrease for the first time steps to begin the locomotion smoothly. This effect quickly diminishes in the first time steps. Finally, the head angle is set to the mean angle of all other angles $\theta$. This ensures that the head module constantly faces in the direction of the motion and improves observing the path via vision sensors. Comparable gait equations are also developed for three-dimensional snake-like locomotion, which shows the applicability of this approach [32, 33]. To explore the application of reinforcement learning for steering and indirect locomotion control together with such slithering gaits, this thesis focuses on the simpler planar movement.

## 3.3 Reinforcement Learning in Robotics

In 2017, the first Conference on Robot Learning (CoRL) took place at the Google campus in Mountain View. According to Google, the goal of CoRL was ”to bring machine learning and robotics experts together for the first time in a single-track conference, in order to foster new research avenues between the two disciplines” [34]. It was sponsored by Google, DeepMind, Siemens, Panasonic, NVIDIA, Amazon, Microsoft, Facebook and Volkswagen among others [35]. The various backgrounds of sponsors show that the potential use cases of reinforcement learning for robotic control are manifold and interesting to distinct applications.
One potential application of RL in robotics is the development of self-driving cars holds interest in many companies. The British Machine Learning company Wayve used reinforcement learning to train a car to autonomously drive in a rural environment via recorded real-world data of driving and ”dreaming” new similar training data [36]. In a later publication, they trained a RL algorithm without prior knowledge that performed all exploration and optimization on-vehicle without previous simulation [37].
While it is interesting to see that on-board learning works and may be feasible for some cases, simulations bring benefits such as less safety risks, less material wear and an abundant source of data [38]. However, transferring policies to reality can fail due to modelling errors and different physical properties. One attempt to solve this ”reality gap” is to randomize the dynamics of the simulator during training [38]. Thus, when ”learning Dexterity”, OpenAI performed the whole training of their algorithm in a simulated environment with randomized physical properties like friction coefficients and successfully transferred the policies to a physical robot [39]. They used reinforcement learning ”to learn dexterous in-hand manipulation policies which can perform vision-based object reorientation on a physical Shadow Dexterous Hand” [39]. The human-like hand learns without human demonstration or prior knowledge, but ”many behaviors found in human manipulation emerge naturally, including finger gaiting, multi-finger coordination, and the controlled use of gravity” [39]. The reinforcement learning algorithm for this work is PPO by Schulman et al. [40], the same as in OpenAI Five [39, 1]. It is the

"default reinforcement learning algorithm at OpenAI because of its ease of use and good performance" [41]. Schulman et al. [40] showed that PPO offers simplicity and good sample efficiency compared to similar state-of-the-art algorithms like Actor-Critic with Experience Replay (ACER) [42]. It will also be the reinforcement learning algorithm used in this thesis. For further information about PPO, it is recommended to refer to Schulman et al. [40].

# 4 Methodology

In this chapter, the simulation environment is discussed in terms of implementation details as well as in terms of the structure of the scene. Furthermore, details of observation and reward definition are given. Finally, the differences and characteristics of the indirect and direct target tracking experiments as well as their respective motivations are explained.

## 4.1 Simulation Environment

To build and simulate the scenes for training and evaluation of the experiments, the robotics simulator *V-REP* by Coppelia Robotics [23] is used as simulation software. It features a simulator as well as means for scene creation and gives the choice of different physics engines to choose from. A scene consists of different objects that can be controlled individually with the use of embedded Lua scripts. The script connected to an object is called a childscript. The snake-like robot is controlled via an embedded Lua childscript in V-REP. In addition to providing some control directly, the Lua script also performs message exchange with the reinforcement learning algorithm. The algorithm itself runs in Python and controls the robot via the V-REP *RemoteAPI* through which it also receives the environment's states.
The reinforcement learning algorithm is the PPO algorithm that was briefly discussed in chapter 3. Specifically, the PPO1 implementation of OpenAI Baselines that is optimized for CPU computation [43]. Utilizing the implementation of Baselines in favor of self-implementing the algorithm brings the benefit of quick prototyping and diminishes risks of introducing bugs and most importantly improving reproducibility of results [43, 44].
For each scene in V-REP, an environment file, written in Python, is needed, in order to control dynamic factors such as the target trajectory as well as to serve as an interface between the RL algorithm and the simulation software. The interfaces for the environments were implemented in Python and locally registered as environments in *OpenAI Gym* [45]. Gym serves as an interface and abstraction layer between the

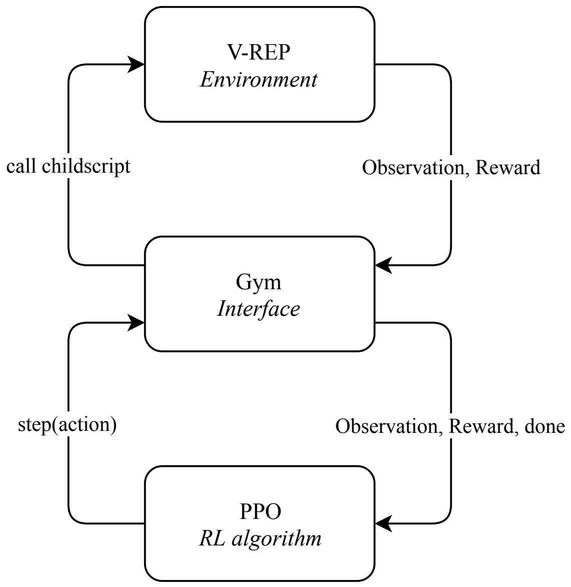

**Figure 4.1:** Overview of the communication with Gym as interface between the RL agent and the simulation program.

reinforcement learning agent and its simulation environment as shown in figure 4.1. The agent is able to call methods such as *step(action)* on a gym environment regardless of the implementation details of the environment. This way, no adjustments to the implementation of PPO had to be made and all specific logic such as the reward calculation and the interaction with V-REP was implemented in the respective Gym environments. As displayed in figure 4.1, the failure condition is assessed in the Gym environment and forwarded to the RL algorithm to inform him if the episode should be reset early, here via the parameter called *done*. In order to integrate V-REP with OpenAI Gym, these environments are inheriting from the superclass of vrep-env that provides helpful Python methods and improved usability for the use of the V-REP RemoteAPI together with Gym and Python [46].
The simulation runs for a number of episodes that respectively last for a maximum of 1024 time steps. An episode ends early if the fail condition is reached at a distance further than 15m to the target. After an episode is over, the scene resets and all objects, like the robot and the target, are reset to their original positions. In each time step, the reinforcement learning algorithm obtains the current observation and the reward for the last action from the environment as explained in chapters 4.1.3 and 4.1.4. Then, the policy decides upon an action for the current observation. This action is sent to the simulation software and the agent waits for the simulation to run for the duration of one time step, before sensing and acting in the next cycle. In order to train the reinforcement learning algorithm, a number of iterations are performed. After each iteration, the algorithm updates its policy

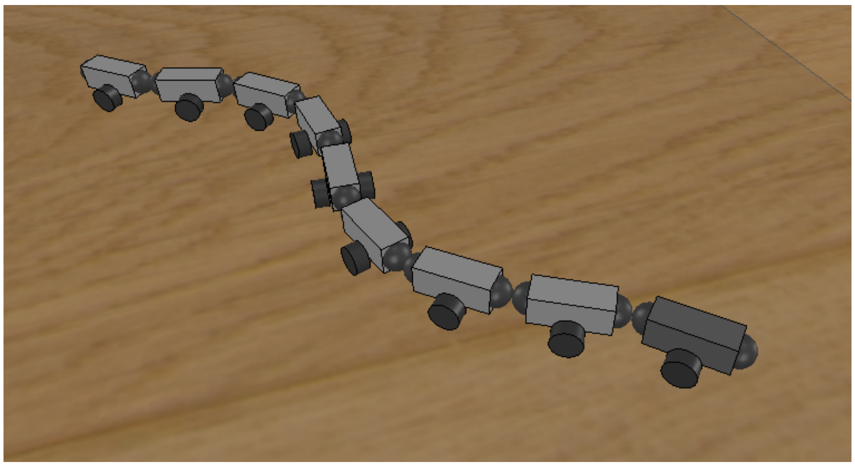

**Figure 4.2:** The model of the snake-like robot in the simulation scene.

based on the past batch of actions and their corresponding rewards in addition to its prior policy. The length of an iteration is 2048 time steps, which equals to the length of two whole episodes.

### 4.1.1 Snake-like Robot

The two agents share the same simulation scenarios for training, containing a snake-like robot and a target that the robot should track. The robot model is inspired by the early ACM versions from Hirose [19] and shortly discussed in chapter 3.1. It is a modular snake-like robot that consists of 8 modules interconnected in a row. Each module has two passive wheels and a joint to each of its neighboring modules. The joints are rotatable in a lateral direction, thus enabling only planar movement. By rotating its joints, the snake is able to use the ground friction and resulting outward facing force at the wheels to perform a slithering locomotion as explained in 3.2.
Figure 4.2 shows the robot in a sinus-like body shape. Each of the 8 modules is 35cm long, 20cm wide and 15cm high. Figure 4.3 shows the dimensions of the first two modules of the robot from a top view and figure 4.4 displays the dimensions of the first two modules from a side view. In its frontal module, the snake model features a vision sensor and a camera which are both facing forward. Only the first is being used as data source for the control algorithm. The latter is not supplying any data to the agent. The camera only provides a rendered image for a more human-friendly visualization during the simulation. The difference between the two sensor types in V-REP is that a vision sensor has a static resolution and is meant to extract image information from a simulation scene, while a camera adapts to the view size and is supposed to be used for scene visualization [23, 47].

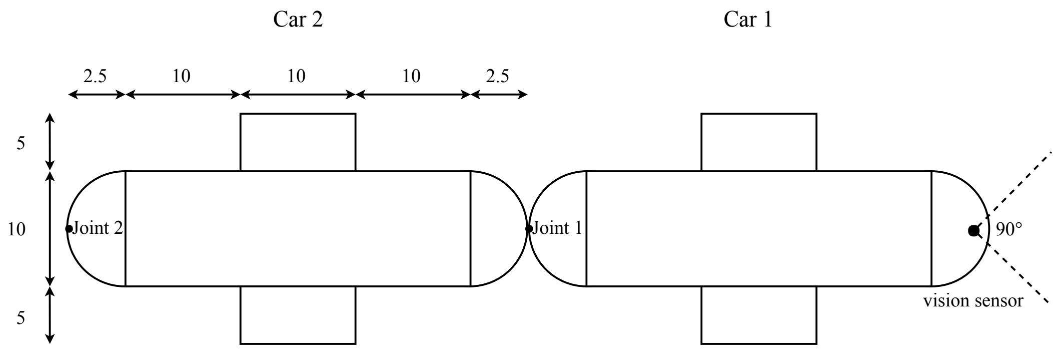

**Figure 4.3:** Top view of the first two modules of the snake model (in cm).

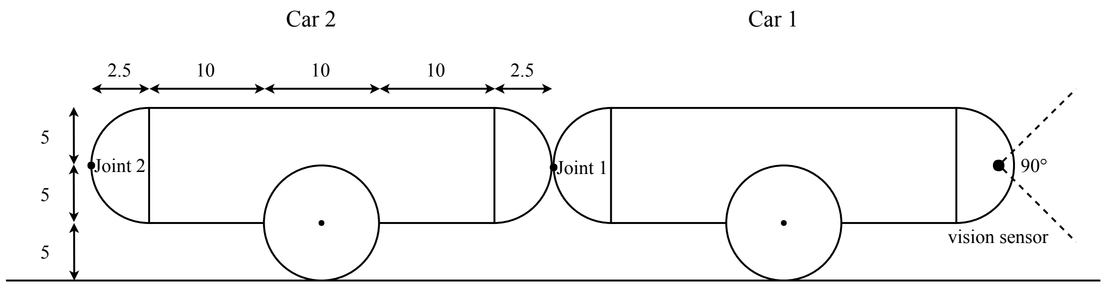

**Figure 4.4:** Side view of the first two modules of the snake model (in cm).

### 4.1.2 Target characteristics and behavior

The target is an object in the scene which is clearly recognizable and distinguishable by its color. It is a big, red sphere that moves in a trajectory away from the snake at a constant pace of 8cm per time step. Each time step equals 50ms of wall time. If the snake comes closer to the target than a minimum distance of 5 meters, the target will hold that minimum distance. The target's constant pace is slightly slower than the robot to ensure that it will only get lost when the agent fails the tracking task and not simply runs out of the vision sensor's range limit of 15 meters due to a higher movement speed. This range limit also marks the failure condition of a maximum distance between the snake and the target. Moving further than 15 meters away from the target will reset the episode. This behavior is chosen in order to evaluate in which episodes a successful target tracking was performed as well as to reduce the training time. When the robot is too far away to perceive the correct direction to the target, an episode is failed and the remaining time steps can safely be omitted.

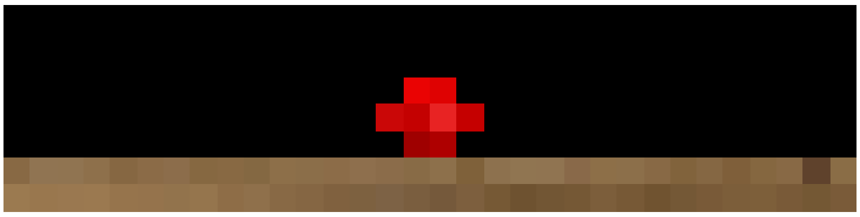

**Figure 4.5:** Exemplary extract of vision sensor data visualized in V-REP. For clarity, this image only shows 8 of 32 lines, 24 lines have been omitted and show either the ground or the "sky".

### 4.1.3 Observation structure

The observation consists of the image obtained from the vision sensor and the angles of the robot's joints as well as the target angles that each joint is currently rotating towards and the current speed of the head module. The vision sensor has an angle of 90° and a resolution of 32x32 pixels. Depth info is ignored for performance reasons and the far clipping plane is set to 15 meters. The image data obtained from the vision sensor is an array of 32x32x3 RGB-values.
To speed up training, some preprocessing is applied to the image data before feeding it to the reinforcement learning algorithm. Firstly, all lines except the first line above the ground are discarded, as they either hold only redundant information or no information at all. Neither the lines above the target nor the lines that solely show the ground hold any information important to the agent. Secondly, this line of 32 RGB-values is one hot encoded with the enumeration 0: ground or "sky" and 1: target. The encoding is done by setting the enumeration 1 to red pixels and the enumeration 0 to all pixels of a different color. This results in an array of 32 binary values. Finally, the joints' current angles and target angles as well as the head speed are added to this array. With the resolution of 32x32 and K=8, the final observation size is 49.

$$[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0] \tag{4.1}$$

After preprocessing, the exemplary vision sensor data in figure 4.5 results in the array displayed in (4.1). The image in figure 4.5 is only an extract of the most important 8 lines of the image. The other 24 lines contain redundant information and only show either the ground or the "sky".

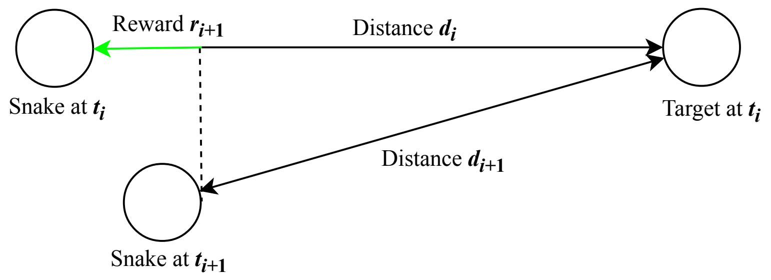

**Figure 4.6:** Schema of the reward definition. The feedback signal for each time step equals the change in distance to the last time step $r_{i+1} = d_i - d_{i+1}$ between the snake and the target in meters. Due to its slithering movement, the snake does not always move straight towards the target in the motion of one time step.

### 4.1.4 Reward definition

The reward function plays an important role in the reinforcement learning model. It serves as the feedback mechanism through which the agent learns if it reached the goal. Moreover, when not using sparse rewards, it signals how well the policy chose each action at the corresponding time step and observation state. Sparse rewards are obtained after a whole episode and not after every time step, thus only informing about the total performance and not containing specific information about each action's direct reward. A mistake in the reward function can result in the emerging of a completely different behavior of the agent. Would the sign of a reward function flip that rewards movement towards the right, the agent would now get rewarded for moving to the left and thus develop moving left as a policy even though this would be the opposite of the desired behavior.
It is also possible for unwanted consequences such as the *Cobra Effect* to arise. This effect describes the emerging of an unwanted result from benevolent actions. It dates back to the time of British colonialism in India where the authorities wanted to counteract the local cobra population and thus paid a prize for each delivered dead cobra. As a result, cobra farms emerged and bred the snakes to obtain more reward. The authorities first thought their campaign was a success until they discovered one of the cobra farms and shut the program down. Hence, the farmers released all their now worthless snakes into the wild and the region had even more of a plague than before. A similar problem can occur with reward functions in reinforcement learning. In fact, OpenAI discovered a case where their agent stacked up side rewards in a race game because it resulted in more reward than finishing the race. [48, 49]
In this work, the behavior that the reward function should incite in the agent is to move towards the target. Therefore, it compares the position of the snake robot before and after a time step. It rewards any movement into the right direction and penalizes movement into

the wrong direction. Figure 4.6 illustrates the reward definition graphically. Let $o_x$ and $o_y$ denote the position of the snake before the time step respectively in x and y coordinates, $n_x$ and $n_y$ the position after the time step in the respective x and y coordinates as well as $t_x$ and $t_y$ the target coordinates before the time step respectively in x and y.

$$d_i = \sqrt{(t_x - o_x)^2 + (t_y - o_y)^2} \tag{4.2}$$

$$d_{i+1} = \sqrt{(t_x - n_x)^2 + (t_y - n_y)^2} \tag{4.3}$$

$$r_{i+1} = d_i - d_{i+1} \tag{4.4}$$

In order to assess the distance that the robot moved towards the target over the course of one time step, the distance between robot and target before the time step (4.2) is compared to the distance between robot and target after the time step (4.3). During the time step, the robot has moved whereas the target position stayed constant and is updated after calculating the reward. The reward signal $r$ equals the difference between the distance before and after the time step and is displayed in equation (4.4). This way, it rewards any movement towards the target and penalizes movement away from it. But movement away from the target that would eventually result in a better position can also yield a higher overall reward. Because the discount factor $\gamma$ is set to 0.99, expected rewards in the future are less important to the algorithm than immediate rewards, but can still strongly influence its policy.

## 4.2 Indirect Locomotion Control Experiment

In the indirect locomotion control experiment, reinforcement learning and a forward slithering gait are combined to solve a target tracking task. The aim of this experiment is to explore the possibilities of a cooperation of a distinct form of static locomotion combined with a RL algorithm that is exclusively steering. In order to properly perform this indirect locomotion control, the algorithm would need to learn the characteristics of the given static slithering gait and adapt its steering orders to these locomotion behaviors. Desirably, it would discover weaknesses in the indirect control and overcome these weaknesses by adapting its policy to evade them. One such weakness in the cooperation could be that the snake-like robot controlled by the slithering gait would have problems making a right

turn while its head is on the left. In such a case, the RL algorithm could theoretically evade this action and wait with the turn until the head is in the right position.
The slithering gait was briefly discussed in chapter 3. The gait equation will provide a forward motion by calculating the 8 angles that each corresponding joint should rotate towards in the next time step. It does not obtain any information from the environment except the current time step. The reinforcement learning algorithm will perceive the observation as explained in chapter 4.1.3 and obtain the reward by the reward definition of chapter 4.1.4. Its action space is one floating point number in the range of $[-10, 10]$. The action will set the offset parameter $b$ in the gait equation for the next time step, thus providing the direction for the slithering gait. The other parameters will stay constant at their default values that are displayed in 3.1.

## 4.3 Direct Locomotion Control Experiment

The direct locomotion control experiment is conducted in order to explore the possibility of direct locomotion control for snake-like robots. In addition, it provides an adequate alternative to evaluate and compare the indirect control agent. The direct locomotion control experiment provides a completely self-taught locomotion that learns how to move and to track the target at the same time and without any prior knowledge.
In contrast to the first experiment, the direct locomotion control experiment consists of one reinforcement learning algorithm that is in direct control over the snake-like robot. It obtains the same observation and gets feedback based upon the same reward definition but controls the joints directly instead of using the gait equation. It doesn't act in one simple direction but instead adapts the whole movement to result in a motion towards this new direction. The action space consists of 8 floating point numbers that translate to the target joint angles. The direct locomotion control experiment sets the joints' target angles in the same way that the slithering gait did. For the next time step, the joints rotate towards their target angle and then receive the next target angle after the time step. This way we can directly compare the locomotion of the gait equation to the learned one.

# 5 Results

In the previous chapter, two types of locomotion control for target tracking with a snake-like robot were presented. The concepts as well as the differences and implementation details were described.
To assess the outcomes of the experiments, in this chapter, training details are given and the locomotion controllers are tested and evaluated in tracking accuracy.

## 5.1 Training Scenario for Target Tracking

The agents of both approaches are trained on the same target tracking scenario. Initially, the snake robot's head module is pointing towards the target which is placed 6 meters away from the head on the x-axis. Firstly, the target moves for 8 meters straight away from the robot and then begins to follow a randomized sinus-like path along the x-axis. The monotonic periods $h_i$ alternate between strictly increasing and strictly decreasing y-values. To the observation of the snake-like robot, this has the effect that the target is taking left and right turns in arbitrary distances. The length of each period $h_i$ is a random amount of time steps in the range $[30, 210]$. Figure 5.1 shows an exemplary target path for this scenario. The position update of the target per time step is displayed in equation (5.1), where $s_x$ denotes the x position of the robot's head and $x_i$ denotes the target's x position at time step $i$.

$$x_{i+1} = \max(x_i + 0.08m, s_x + 4m) \tag{5.1}$$

The pace along the x-axis is the maximum of 0.08 meters per time step and the speed of the snake-like robot. The target's speed along the y-axis is constant at 0.05 meters per time step. Due to the random length of the periods $h$, the trajectories for each episode are nondeterministic.

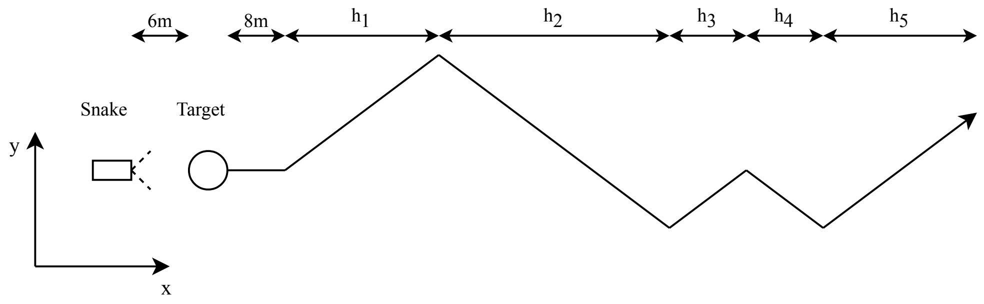

**Figure 5.1:** Exemplary extract of the target trajectory of the randomized training scenario. The initial distance between target and robot is 6 meters. The periods $h_i$ are of the same gradient per time step and the sign alternates each period. The length of each period $h$ is a random amount of time steps in the range $[30, 210]$. Hence, the number of periods can vary between 5 and 35 in a full episode.

## 5.2 Evaluation Scenario

The evaluation of tracking success is performed on an evaluation course where the target follows a circle trajectory after an initial straight section of 8 meters. This previously unseen maneuver is supposed to assess the robustness of the trained target tracking agents. After the initial straight section, the target makes a turn of 90° to the right and follows the path of a circle with a radius of 40 meters clockwise. The center of the circle lies 40 meters left on the x-axis from the point of the first turn. Figure 5.2 shows the trajectory of the target in the evaluation scenario. To reduce simulation time in evaluation, the episode is finished once the target reaches a position 14 meters left to its initial position on the x-axis. The first turn is the important obstacle to overcome. The following circle consists of a constant wide turn to the right and is less complex to track.

## 5.3 Training Details

The training conditions were equal for both agents. The algorithms learned with the same hyperparameters and for 550 iterations each. Table 5.1 displays the hyperparameters that were set for the training of both PPO algorithms. They are identical to the hyperparameters that OpenAI Baselines [43] found by random search in the humanoid robot locomotion training example for PPO1. OpenAI states that these hyperparameters are "good enough to make humanoid walk, but whether those are an absolute best or not is not certain" [43]. The *MlpPolicy* from Baselines [43] served as policy network with 2 hidden layers and a hidden layer size of 64.
The slope of the mean reward that an RL agent obtained during training is a sign of how

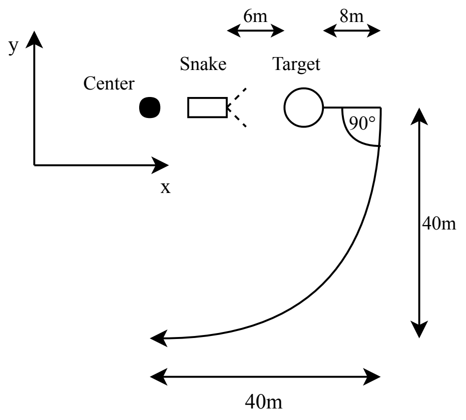

**Figure 5.2:** Target trajectory in the evaluation scenario. The initial distance between target and robot is 6 meters. After a first straight section, the target makes a 90° turn to the right and follows the path of a circle with radius of 40 meters clockwise. The center of the circle is 40 meters left on the x-axis from the point of the first turn.

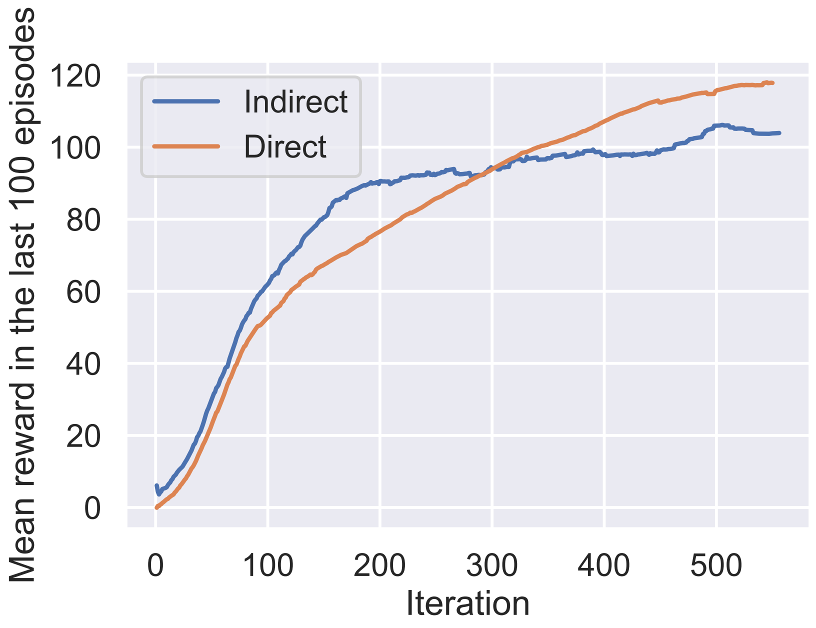

**Figure 5.3:** Mean reward of the last 100 episodes for each iteration.

**Table 5.1:** The PPO hyperparameters for both agents in this work. They are the same as in the humanoid example of Baselines [43]. The hyperparameters are not explained in this work. They are given for reproducibility.

| Hyperparameter | Value |
|---|---|
| Time steps per actorbatch | 2048 |
| Clipping parameter $\epsilon$ | 0.2 |
| Entropy coefficient | 0.0 |
| Optimization epochs | 10 |
| Optimization step size | 0.0003 |
| Optimization batch size | 64 |
| Discount factor | 0.99 |
| Advantage estimation | 0.95 |
| Schedule | linear |

fast its policy improves and the obtained rewards increases whereas the convergence of the mean reward of an agent means that it is unable to find a policy that further improves the obtained reward. For successful agents, the reward in this work is constantly growing over the course of the time steps in one episode. Hence, an unsuccessful tracking episode that is reset early has less potential to maximize the reward.
Over the course of training, the development of the mean reward behaved differently for the two agents. For every iteration in the training process, figure 5.3 shows the mean reward of the last 100 episodes. The indirect control agent's mean reward converges quicker whereas the mean reward of the direct locomotion control agent reaches a higher maximum. Both of these observations correlate to the nature of the reward definition and the difference of the agents. Initially, the movement capability of the direct control agent still needs to be discovered and then improved, while the indirect control agent has no impact on the robot's movement speed and moves fast from the very beginning. Once the concept of moving in the general direction of the target is grasped, the obtained reward is only affected by the tracking accuracy and the movement speed of the robot. The indirect control agent is able to achieve high reward scores with its constant speed as soon as it reaches a sufficient tracking accuracy. On the other hand, the direct control agent learns increasingly faster ways of movement and thus is able to obtain increasingly higher rewards with more effective locomotion capabilities. Finally, the direct control agent is able to obtain a higher overall mean reward because its final way of movement is faster than the movement of the indirect control agent. However, this aspect isn't an achievement without further assessment. The movement speed of the slithering gait relies on its parameters, especially $\omega$. The value of $\omega$ for the gait equation in this work was chosen to be approximately as fast as the trained direct control agent on a scenario with a straight line target trajectory. It may be inferred that the locomotion that was learned

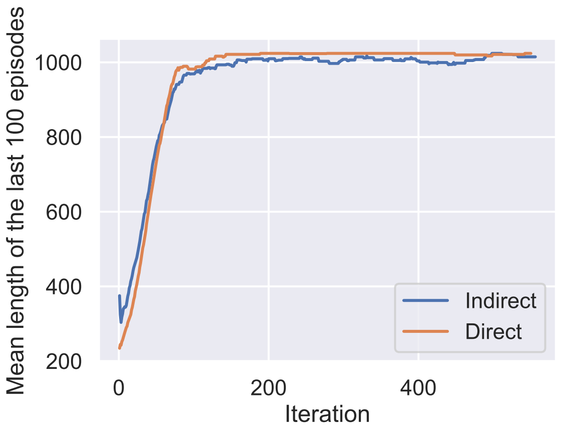

**Figure 5.4:** Mean episode length of the last 100 episodes for each iteration.

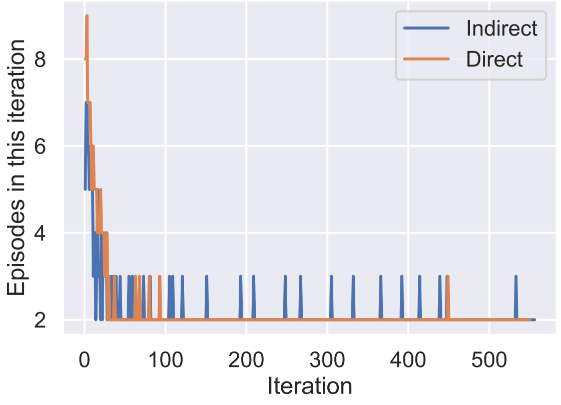

**Figure 5.5:** Episodes per iteration.

by the direct control agent is able to move faster through curves than the slithering gait. However, this could also be due to unoptimized steering by the indirect control agent.

## 5.4 Comparison of Target Tracking Capabilities

In this work, the main metric for evaluation and comparison of the two experiments is the target tracking accuracy that the agents achieve. After analyzing the training performance for first assessments, both agents are evaluated and compared on the training scenario explained in chapter 5.1 as well as on the evaluation scenario discussed in chapter 5.2.
The length of a complete episode without reaching the failure condition is 1024 time steps, the length of each iteration is 2048 time steps. Hence, every iteration with a number of episodes $n > 2$ involves $n-2$ unsuccessful target tracking scenarios that reached the failure condition. Figure 5.4 shows the mean episode length of the last 100 episodes for every iteration. After 76 and 86 iterations, the direct control agent and the indirect control agent respectively reached a mean episode length of more than 950. The direct control agent converges towards 1024 but the mean episode length of the indirect control agent fluctuates around 1000 for the whole training process. The number of episodes per iteration displayed in figure 5.5 shows that the successful periods of episodes without tracking failures are a lot shorter for the indirect agent. Both agents have their last iteration with more than one failure before iteration 30. After the first 100 iterations, the indirect control agent fails 16 further target tracking scenarios whereas the direct control agent only fails one more episode. It is not certain, if 550 iterations was the optimal amount of training. Too many episodes can cause the agent to overfit and thereby worsen generalization whereas

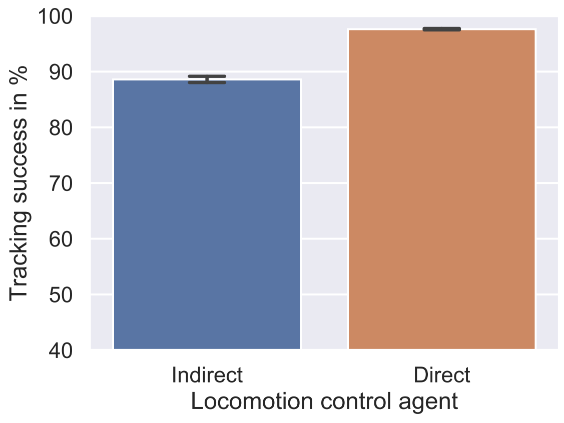

**Figure 5.6:** Comparison of target tracking accuracy in the training scenario.

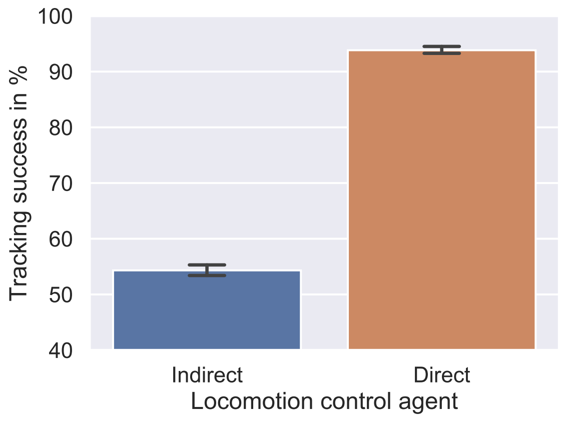

**Figure 5.7:** Comparison of target tracking accuracy in the evaluation scenario.

too little training episodes can result in an agent with an unoptimized policy.
To evaluate their tracking accuracy, the agents are tested on the training scenario displayed in figure 5.1 as well as on the evaluation scenario depicted in figure 5.2. Every assessment runs for 1000 episodes, is repeated five times and the mean results are evaluated. Altogether, the evaluation was performed on 20.000 episodes with 10.000 episodes per agent. Target tracking accuracy was assessed on the training scenario as well as on the evaluation scenario. The training scenario is less complex as its turns are not as sharp as the turn of the evaluation scenario. Figure 5.6 shows the target tracking accuracy of both agents in the training scenario. The indirect control agent achieved a mean of 88.60% successful episodes with a standard deviation of 0.731%, whereas the direct control agent performed a mean of 97.62% successful tracking tasks with a standard deviation of 0.164%. While both agents achieved a good tracking accuracy, the direct agent was more robust to follow the sudden turns of the target.
To further assess their capabilities on sudden sharp turns, the agents were then tested in the evaluation scenario. The target tracking task of this scenario is significantly harder due to its sharp 90° turn. Figure 5.7 shows the target tracking accuracy of both agents in the evaluation scenario. The direct control agent achieved a mean of 93.84% successful episodes with a standard deviation of 0.757%. In contrast, in this scenario, the indirect control agent only performed a mean of 54.32% successful episodes with a standard deviation of 1.232%. Observing the simulation shows that the indirect control agent only successfully performs the 90° turn if it begins to make the turn on the first 4 meters. Otherwise, the curve is too sharp to follow for this agent. Figure 5.8 shows exemplary trajectories of a successful episode for both agents in the evaluation scenario. The indirect control agent visibly turns earlier on its path. Considering this, one reason is that the

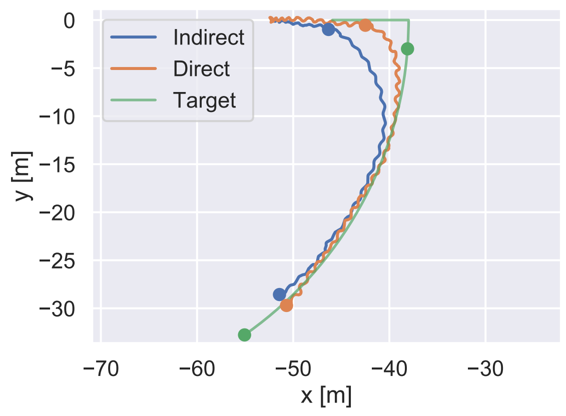

**Figure 5.8:** Exemplary path comparison of indirect and direct control agent in the evaluation scenario. The dots are placed at the location of each agent at the time step of the corresponding target location. The indirect control agent took 33 time steps longer to finish this episode.

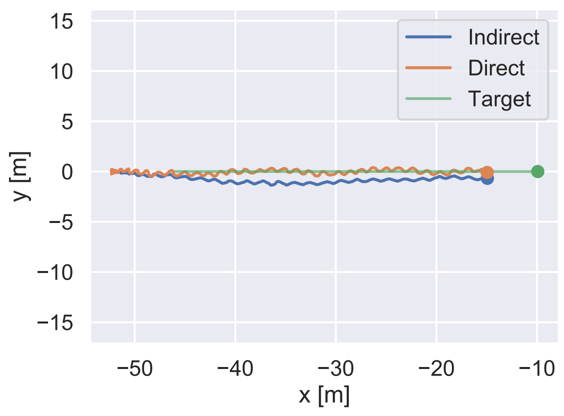

**Figure 5.9:** Exemplary path comparison of indirect and direct control agent in the line scenario. The dots are placed at the location of each agent at the time step of the corresponding target location. The indirect control agent took 50 time steps longer to finish this episode.

indirect agent's movement speed is a bit slower, especially during the first time steps due to its smoothing as explained in chapter 3.2. Thus, the robot hasn't moved as far when it first perceives that the target is making a turn. The dots in figure 5.8 show the respective location of the robot's head at the time step that the target was in the same location. It shows that the direct control agent is closer to the target and performs a sharper turn. Moreover, although the slithering gait equation could theoretically make sharp turns, the indirect control agent trained in this work was only capable of performing wider turns.
The learned locomotion performed by the direct control agent is similar to the slithering gait of the indirect control agent. Figure 5.8 and figure 5.9 show exemplary trajectories on the evaluation scenario as well as on a simple line scenario. In both figures, the curves of the indirect control agent and the direct control agent move in a similar path. The similarity of the curves shows how close the two resulting types of locomotion are.
In theory, the emergence of a completely different way of movement for planar snake-like robots with passive wheels could have been possible. The similarity between the human-crafted locomotion that was inspired by real snakes and the locomotion that was learned by discovery of an RL agent shows that the results of both control approaches can, in theory, yield very similar results.

# 6 Discussion

Controlling locomotion of robots for target tracking scenarios requires suitable locomotion control agents. In this work, two different approaches with the goal to achieve robust target tracking with snake-like robots were presented.
The first approach controlled the locomotion of the robot indirectly. It was built upon the locomotion control by a slithering gait equation and steering the movement via an RL agent. Controlling the direction of this movement, the indirect agent successfully tracked the target in the training scenario. However, it struggled with the sharp turn of the target in the evaluation scenario and only performed 54.32% of the tracking episodes in this scenario successfully whereas it lost the target in almost every other episode. The RL agent was limited to steering and could not influence the other parameters of the slithering gait equation such as the frequency or the amplitude of the swings. In the observation of the simulation of this work, the slithering gait equation seemed to be more suitable for stable movement in a constant direction. When given a new direction, the script seemed to reset and override the complete posture of the snake to the new direction before getting into efficient movement again. The many sudden changes in direction resulted in a constant resetting. These discontinuous movements made the resulting locomotion look hectic and like an unoptimized way of movement. Hence, the smooth locomotion and effectiveness of the slithering gait with a constant direction was lost to a certain degree. This effect was observed to be worse whith lower values of frequency $\omega$.
In the second approach, an RL agent directly controlled the locomotion of the robot and learned how to move and track the target from scratch. It has proven to be a more robust agent for target tracking in the training scenario and even achieved 93.84% successful episodes in the evaluation scenario. The learned locomotion of the direct control agent appears to cope better with the frequent directional changes of target tracking. Instead of resetting the whole posture, it seemed to make the changes that are needed for the transition of the turn and keep the motion more steady. Controlling direction and locomotion from the same source could make adaption to changes easier and the locomotion control agent more robust. The learned locomotion of the direct control agent is remarkably similar to the slithering gait and performs a similar sinus-like movement.
Although the evaluation of the experiments in this work fell in favour of the direct

locomotion control agent, the inferiority of the indirect locomotion control approach can not be inferred from these findings alone. Firstly, the offset of the limitation of constant parameters imposed on the indirect agent would be likely to improve the movement capabilities as well as the target tracking capabilities of the indirect agent. Potentially, there is a lot of room for improvement by dynamically changing all parameters that characterize the slithering gait equation. It is possible that strategies such as to reduce the amplitude in a sharp turn would improve the movement capabilities of the slithering gait. Secondly, even a worse gait equation is favorable to a more efficient RL agent in some cases. The decision making process of an RL agent is not as predictable, transparent and debuggable as a script calculating an equation. Especially in critical scenarios such as disaster rescue, the predictability of an agent may be more important than a slightly better movement capability with some degree of uncertainty. Moreover, changing the behavior of an RL agent requires retraining and potentially the design of a new reward definition while some changes are easily applied in a gait equation. Finally, the deployment location of an agent for critical disaster rescue tasks can never be certain. While possibly worse, a human operator may still be able to move a robot through priorly unseen environments whereas the RL agent might show undesired behavior when presented with a new kind of obstacle or environment. In the case of modular control for locomotion and steering, a human operator can replace the steering agent if necessary. However, this is not possible with the single direct control agent approach.
In conclusion, both approaches for locomotion control agents remain promising for future applications with snake-like robots. The target tracking capabilities of both approaches could be further improved by using a Convolutional Neural Network instead of the Multi-Layer Perceptron as policy network for the PPO algorithm. Additionally, the investigation of the performance of other RL algorithms as well as performing hyperparameter optimization might lead to more robust and adaptable agents. The importance of movement speed on the obtainable reward in this work made it hard to analyze other aspects of the agents and a different task and reward definition could make more diverse analysis and comparison of the agents possible. Furthermore, both approaches could be tested for tasks that relies on more than tracking accuracy and speed, such as to hold a specific distance to the target or move into its vicinity and stop. In the future, the application of both kinds of agents in more realistic and rich environments with obstacles could be studied for the emergence of better movement capabilities as Heess et al. [16] have shown. Due to the findings of this work, a third approach for practical use of locomotion control for snake-like robots is proposed. It might prove to be useful to utilize an RL locomotion control agent that reacts to a steering signal and performs a movement towards that direction, similarly to how the slithering gait equation

was utilized in this work. A first concept could use the travelled distance into the desired direction as reward definition. In a way, the direct control agent of this work was one such agent which was steered by the direction of the target. The resulting locomotion control agent could prove to be more robust and adaptable than a gait equation and still retain the possibility to be steered by a human operator similar to a gait equation. Finally, the exploration of suitable locomotion control agents for target tracking with snake-like robots that perform three-dimensional movement remains an important goal. The application of planar wheeled snake-like robots is limited and a wide range of operations of snake-like robots require robust and adaptable three-dimensional movement.

# Bibliography

[1] OpenAI. OpenAI Five, 2018. URL `https://blog.openai.com/openai-five/`.

[2] Kelvin Xu, Jimmy Lei Ba, Ryan Kiros, Kyunghyun Cho, Aaron Courville, Ruslan Salakhutdinov, Richard S Zemel, and Yoshua Bengio. Show, Attend and Tell: Neural Image Caption Generation with Visual Attention. Technical report, 2015. URL `http://proceedings.mlr.press/v37/xuc15.pdf`.

[3] Waymo. On the Road, 2018. URL `https://waymo.com/ontheroad/`.

[4] Uber. Steel City's New Wheels, 2016. URL `https://www.uber.com/blog/pennsylvania/new-wheels/`.

[5] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction, Second Edition.* MIT Press, 2018. ISBN 9780262039246. URL `https://mitpress.mit.edu/books/reinforcement-learning-second-edition`.

[6] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep Residual Learning for Image Recognition. Technical report, 2015.

[7] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification, 2015. URL `https://www.cv-foundation.org/openaccess/content_iccv_2015/html/He_Delving_Deep_into_ICCV_2015_paper.html`.

[8] Large Scale Visual Recognition Challenge 2015, 2015. URL `http://image-net.org/challenges/LSVRC/2015/results`.

[9] David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 1 2016. ISSN 0028-0836. doi: 10.1038/nature16961. URL `http://www.nature.com/articles/nature16961`.

[10] Fan Hui and Lucas Baker. Innovations of AlphaGo, 2017. URL `https://deepmind.com/blog/innovations-alphago/`.

[11] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, Yutian Chen, Timothy Lillicrap, Fan Hui, Laurent Sifre, George van den Driessche, Thore Graepel, and Demis Hassabis. Mastering the game of Go without human knowledge. *Nature*, 550(7676):354–359, 10 2017. ISSN 0028-0836. doi: 10.1038/nature24270. URL `http://www.nature.com/doifinder/10.1038/nature24270`.

[12] Demis Hassabis and David Silver. AlphaGo Zero: Learning from scratch, 2017. URL `https://deepmind.com/blog/alphago-zero-learning-scratch/`.

[13] OpenAI. The International 2018: Results, 2018. URL `https://blog.openai.com/the-international-2018-results/`.

[14] OpenAI. OpenAI Five Benchmark: Results, 2018. URL `https://blog.openai.com/openai-five-benchmark-results/`.

[15] OpenAI. Dota 2, 2017. URL `https://blog.openai.com/dota-2/`.

[16] Nicolas Heess, Dhruva TB, Srinivasan Sriram, Jay Lemmon, Josh Merel, Greg Wayne, Yuval Tassa, Tom Erez, Ziyu Wang, S. M. Ali Eslami, Martin Riedmiller, and David Silver. Emergence of Locomotion Behaviours in Rich Environments. 7 2017. URL `http://arxiv.org/abs/1707.02286`.

[17] Pål. Liljebäck. *Snake robots : modelling, mechatronics, and control.* Springer, 2013. ISBN 9781447129967.

[18] Cornell Wright, Austin Buchan, Ben Brown, Jason Geist, Michael Schwerin, David Rollinson, Matthew Tesch, and Howie Choset. Design and Architecture of the Unified Modular Snake Robot. Technical report, 2012. URL `http://biorobotics.ri.cmu.edu/papers/paperUploads/ICRA2012_Wright.pdf`.

[19] Shigeo Hirose. *Biologically inspired robots : snake-like locomotors and manipulators.* Oxford University Press, 1993. ISBN 0198562616. URL `https://dl.acm.org/citation.cfm?id=562623`.

[20] Shigeo Hirose - Relentless passion for the creation of robots, 2013. URL `https://www.titech.ac.jp/english/research/stories/shigeo_hirose.html`.

[21] Carnegie Mellon Snake Robot Used in Search for Mexico Quake Survivors, 2017. URL `https://www.cmu.edu/news/stories/archives/2017/september/snakebot-mexico.html`.

[22] Matthew Tesch, Kevin Lipkin, Isaac Brown, Ross Hatton, Aaron Peck, Justine Rembisz, and Howie Choset. Parameterized and Scripted Gaits for Modular Snake Robots. *Advanced Robotics*, 23:1131–1158, 2009. doi: 10.1163/156855309X452566. URL `www.brill.nl/ar`.

[23] M Freese E. Rohmer S. P. N. Singh. V-REP: a Versatile and Scalable Robot Simulation Framework. In *Proc. of The International Conference on Intelligent Robots and Systems (IROS)*, 2013.

[24] Richard .S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*, volume 9. MIT Press, 1998. ISBN 0262193981. doi: 10.1109/TNN.1998.712192.

[25] Martin Riedmiller, Roland Hafner, Thomas Lampe, Michael Neunert, Jonas Degrave, Tom Van De Wiele, Volodymyr Mnih, Nicolas Heess, and Tobias Springenberg. Learning by Playing-Solving Sparse Reward Tasks from Scratch. Technical report, 2018.

[26] Hisato Ando, Yuichi Ambe, Akihiro Ishii, Masashi Konyo, Kenjiro Tadakuma, Shigenao Maruyama, and Satoshi Tadokoro. Aerial Hose Type Robot by Water Jet for Fire Fighting. *IEEE Robotics and Automation Letters*, 3(2):1128–1135, 4 2018. ISSN 2377-3766. doi: 10.1109/LRA.2018.2792701. URL `http://ieeexplore.ieee.org/document/8255553/`.

[27] Elliot W. Hawkes, Laura H. Blumenschein, Joseph D. Greer, and Allison M. Okamura. A soft robot that navigates its environment through growth. *Science Robotics*, 2(8): eaan3028, 7 2017. ISSN 2470-9476. doi: 10.1126/scirobotics.aan3028. URL `http://robotics.sciencemag.org/lookup/doi/10.1126/scirobotics.aan3028`.

[28] Alessandro Crespi, Konstantinos Karakasiliotis, Andre Guignard, and Auke Jan Ijspeert. Salamandra Robotica II: An Amphibious Robot to Study Salamander-Like Swimming and Walking Gaits. *IEEE Transactions on Robotics*, 29(2):308–320, 4 2013. ISSN 1552-3098. doi: 10.1109/TRO.2012.2234311. URL `http://ieeexplore.ieee.org/document/6416074/`.

[29] David Rollinson, Yigit Bilgen, Ben Brown, Florian Enner, Steven Ford, Curtis Layton, Justine Rembisz, Mike Schwerin, Andrew Willig, Pras Velagapudi, and Howie Choset. Design and Architecture of a Series Elastic Snake Robot. Technical report, 2014.

[30] K Y Pettersen, P Liljebäck, Ø Stavdahl, and J T Gravdahl. Snake Robots From Biology to Nonlinear Control. 2013. doi: 10.3182/20130904-3-FR-2041.00062.

[31] David L Hu, Jasmine Nirody, Terri Scott, Michael J Shelley, and David Hu. The mechanics of slithering locomotion. Technical Report 25, 2009. URL `www.pnas.orgcgidoi10.1073pnas.0812533106`.

[32] Zhenshan Bing, Long Cheng, Kai Huang, Zhuangyi Jiang, Guang Chen, Florian Röhrbein, and Alois Knoll. Towards Autonomous Locomotion: Slithering Gait Design of a Snake-like Robot for Target Observation and Tracking. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 9 2017.

[33] Zhenshan Bing, Long Cheng, Guang Chen, Florian Röhrbein, Kai Huang, and Alois Knoll. Towards autonomous locomotion: CPG-based control of smooth 3D slithering gait transition of a snake-like robot. *Bioinspiration & Biomimetics*, 12(3):35001, 2017. doi: 10.1088/1748-3190/aa644c.

[34] Vincent Vanhoucke and Melanie Saldaña. Google AI Blog: A Summary of the First Conference on Robot Learning, 2017. URL `https://ai.googleblog.com/2017/12/a-summary-of-first-conference-on-robot.html`.

[35] CoRL2017 - Conference on Robot Learning 2017, 2017. URL `http://www.robot-learning.org/home/corl2017`.

[36] Wayve. Wayve — Dreaming about Driving., 2018. URL `https://wayve.ai/blog/dreaming-about-driving-imagination-rl`.

[37] Alex Kendall, Jeffrey Hawke, David Janz, Przemyslaw Mazur, Daniele Reda, John-Mark Allen, Vinh-Dieu Lam, Alex Bewley, and Amar Shah. Learning to Drive in a Day. Technical report, 2018. URL `https://wayve.ai/blog/l2diad`.

[38] Xue Bin Peng, Marcin Andrychowicz, Wojciech Zaremba, and Pieter Abbeel. Sim-to-Real Transfer of Robotic Control with Dynamics Randomization. 10 2017. doi: 10.1109/ICRA.2018.8460528. URL `http://arxiv.org/abs/1710.06537http://dx.doi.org/10.1109/ICRA.2018.8460528`.

[39] OpenAI, :, Marcin Andrychowicz, Bowen Baker, Maciek Chociej, Rafal Jozefowicz, Bob McGrew, Jakub Pachocki, Arthur Petron, Matthias Plappert, Glenn Powell, Alex Ray, Jonas Schneider, Szymon Sidor, Josh Tobin, Peter Welinder, Lilian Weng, and Wojciech Zaremba. Learning Dexterous In-Hand Manipulation. 8 2018. URL `http://arxiv.org/abs/1808.00177`.

[40] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal Policy Optimization Algorithms. 7 2017. URL `http://arxiv.org/abs/1707.06347`.

[41] OpenAI. Proximal Policy Optimization, 2017. URL `https://blog.openai.com/openai-baselines-ppo/`.

[42] Ziyu Wang, Victor Bapst, Nicolas Heess, Volodymyr Mnih, Remi Munos, Koray Kavukcuoglu, and Nando de Freitas. Sample Efficient Actor-Critic with Experience Replay. 11 2016. URL `http://arxiv.org/abs/1611.01224`.

[43] Prafulla Dhariwal, Christopher Hesse, Oleg Klimov, Alex Nichol, Matthias Plappert, Alec Radford, John Schulman, Szymon Sidor, Yuhuai Wu, and Peter Zhokhov. OpenAI Baselines, 2017. URL `https://github.com/openai/baselines`.

[44] Szymon Sidor and John Schulman. OpenAI Baselines: DQN, 2017. URL https://blog.openai.com/openai-baselines-dqn/.

[45] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. OpenAI Gym, 2016.

[46] Yuri Soares. V-REP Env, 2017. URL https://github.com/ycps/vrep-env.

[47] Coppelia Robotics. Vision Sensors, 2018. URL http://www.coppeliarobotics.com/helpFiles/en/visionSensors.htm.

[48] Jack Clark and Dario Amodei. Faulty Reward Functions in the Wild, 2016. URL https://blog.openai.com/faulty-reward-functions/.

[49] Horst. Siebert. *Der Kobra-Effekt : wie man Irrwege der Wirtschaftspolitik vermeidet.* Dt. Verl.-Anst, 2001. ISBN 3421055629. URL https://books.google.de/books?id=s9chPwAACAAJ&dq=3421055629&hl=en&sa=X&ved=0ahUKEwiDtKDw463dAhXrposKHayLCqwQ6AEIKTAA.