Undergraduate/Master's Thesis

Your Title Goes Here

Robin Uhrich

Examiner: Prof. Dr. Joschka Boedecker
Advisers: Jasper Hoffman

University of Freiburg
Faculty of Engineering
Department of Computer Science
Neurobotics Lab

July 9, 2023

Writing Period

05. 07. 2016 – 05. 10. 2016

Examiner

Prof. Dr. Joschka Boedecker

Second Examiner

Prof. Dr. Wile E. Coyote

Advisers

Jasper Hoffman

Bachelor Thesis

Your Title Goes Here

Robin Uhrich

Gutachter: Prof. Dr. Joschka Boedecker
Betreuer: Jasper Hoffman

Albert-Ludwigs-Universität Freiburg
Technische Fakultät
Institut für Informatik
Lehrstuhl für Neurorobotik

July 9, 2023

Bearbeitungszeit

05. 07. 2016 – 05. 10. 2016

Gutachter

Prof. Dr. Joschka Boedecker

Zweitgutachter

Prof. Dr. Wile E. Coyote

Betreuer

Jasper Hoffman

Declaration

I hereby declare that I am the sole author and composer of my thesis and that no other sources or learning aids, other than those listed, have been used. Furthermore, I declare that I have acknowledged the work of others by providing detailed references of said work.
I hereby also declare that my Thesis has not been prepared for another examination or assignment, either wholly or excerpts thereof.

Place, Date	Signature

Abstract

foo bar

Zusammenfassung

German version is only needed for an undergraduate thesis.

Contents

List of Figures

List of Tables

List of Algorithms

1 Introduction

This is a template for an undergraduate or master's thesis. The first sections are concerned with the template itself. If this is your first thesis, consider reading Section 1.3.

The structure of this thesis is only an example. Discuss with your adviser what structure fits best for your thesis.

1.1 Template Structure

- To compile the document either run the makefile or run your compiler on the file 'thesis_main.tex'. The included makefile requires latexmk which automatically runs bibtex and recompiles your thesis as often as needed. Also it automatically places all output files (aux, bbl, ...) in the folder 'out'. As the pdf also goes in there, the makefile copies the pdf file to the parent folder. There is also a makefile in the chapters folder, to ensure you can also compile from this directory.

- The file 'setup.tex' includes the packages and defines commands. For more details see Section 1.2.

- Each chapter goes into a separate document, the files can be found in the folder chapters.

- The bib folder contains the .bib files, I'd suggest to create multiple bib files for different topics. If you add some or rename the existing ones, don't forget to also change this in thesis_main.tex. You can then cite as usual [1, 2, 3].

- The template is written in a way that eases the switch from scrbook to book class. So if you're not a fan of KOMA you can just replace the documentclass in the main file. The only thing that needs to be changed in setup.tex is the caption styling, see the comments there.

1.2 setup.tex

Edit setup.tex according to your needs. The file contains two sections, one for package includes, and one for defining commands. At the end of the includes and commands there is a section that can safely be removed if you don't need algorithms or tikz. Also don't forget to adapt the pdf hypersetup!!
setup.tex defines:

- some new commands for remembering to do stuff:
  - `\todo{Do this!}`: **(TODO: Do this!)**
  - `\extend{Write more when new results are out!}`:
    (EXTEND: Write more when new results are out!)
  - `\draft{Hacky text!}`: **(DRAFT: Hacky text!)**

- some commands for referencing, 'in `\chapref{chap:introduction}`' produces 'in Chapter 1'
  - `\chapref{}`
  - `\secref{sec:XY}`

  - `\eqref{}`
  - `\figref{}`
  - `\tabref{}`

- the colors of the Uni's corporate design, accessible with
  `{\color{UniX} Colored Text}`
  - UniBlue
  - UniRed
  - UniGrey

- a command for naming matrices `\mat{G}`, $\mathbf{G}$, and naming vectors `\vec{a}`, $\mathbf{a}$. This overwrites the default behavior of having an arrow over vectors, sticking to the naming conventions normal font for scalars, bold-lowercase for vectors, and bold-uppercase for matrices.

- named equations:

```
\begin{align}
    d(a,b) &= d(b,a)\\ \eqname{symmetry}
\end{align}
```

$$d(a,b) = d(b,a) \tag{1}$$

symmetry

1.3 Advice

This section gives some advice how to write a thesis ranging from writing style to formatting. To be sure, ask your advisor about his/her preferences.
For a more complete list we recommend to read Donald Knuth's paper on mathematical writing. (At least the first paragraph). `http://jmlr.csail.mit.edu/reviewing-papers/knuth_mathematical_writing.pdf`

- If you use formulae pay close attention to be consistent throughout the thesis!

- In a thesis you don't write 'In [24] the data is..'. You have more space than in a paper, so write 'AuthorXY et al. prepare the data... [24]'. Also pay attention to the placement: The citation is at the end of the sentence before the full stop with a no-break space. `... last word~\cite{XY}.`

- Pay attention to comma usage, there is a big difference between English and German. '...the fact that bla...' etc.

- Do not write 'don't ', 'can't' etc. Write 'do not', 'can not'.

- If an equation is at the end of a sentence, add a full stop. If it's not the end, add a comma: $a = b + c \quad (1)$,

- Avoid footnotes if possible.

- Use ``'' for citing, not "".

- It's important to look for spelling mistakes in your thesis. There are also tools like aspell that can help you find such mistakes. This is never an excuse not to properly read your thesis again, but it can help. You can find an introduction under `https://git.fachschaft.tf/fachschaft/aspell`.

- If have things like a graph or any other drawings consider using tikz, if you need function graphs or diagrams consider using pgfplots. This has the advantage that the style will be more consistent (same font, formatting options etc.) than when you use some external program.

- Discuss with your advisor whether to use passive voice or not. In most computer science papers passive voice is avoided. It's harder to read, more likely to produce errors, and most of the times less precise. Of course there are situations where the passive voice fits but in scientific papers they are rare. Compare the sentence: 'We created the wheel to solve this.' to 'The wheel was created to solve this', you don't know who did it, making it harder to understand what is your contribution and what is not.

- In tables avoid vertical lines, keep them clean and neat. See **??** for an example. More details can be found in the 'Small Guide to Making Nice Tables' `https://www.inf.ethz.ch/personal/markusp/teaching/guides/guide-tables.pdf`

2 Related Work

LASER: Learning Latent Action Space for Efficient Reinforcement Learning

3 Background

The following part is to provide a short introduction into the theory of reinforcement learning, the Soft Actor Critic Algorithm (SAC) as well to Variational Autoencoders (VAE) and Conditional Variational Autoencoders (CVAE). **(TODO: update!!!)**

3.1 RL Framework

Learning is a topic that is present in all our lives since their beginning. We all learn to move our arms, like an infant that is wiggling around, learn to walk, to speak and more or less any other skill that we find in our personal repository until now.

Reinforcement Learning (RL) is about to have a computational approach to this kind of learning. In the class of RL-Problems the environment provides a challenge like, riding a bicycle. But it is not told how this goal can be achieved. Therefore it has be learned how to derive actions from situations on or off the bike in order to complete this task or - speaking of the numerical approach - maximize a reward function. In most cases the problem solving methods that are classified as RL-methods are applied on tasks that require not only a sequence of actions to interact with an environment but also some kind of feedback. Therefore these setups are designed to provide this kind of feedback. Either if it is an immediate feedback like an additional time step the agent has not crashed the bike or if it is more like a long shot feedback, for example a bonus that the agent has earned after a board game. These principals can be boiled down to three main characteristics of reinforcement learning problems:

- There is a closed loop mechanism between actions and feedback that is provided by the environment where the agent is operating in.

- The agent is not given any direct instructions how to solve the given problem.

- The actions an agent is performing have consequences over time. Either for him or the environment he is working in.

Before we can have an exemplary look on the closed loop mechanics, we would like to introduce a couple of key components of RL including the Markov Decision Process.

Every agent exists in a specified environment and also interacts within a sequence of discrete time steps $t \in \mathbb{N}$. In order to do so the agent has to perceive the environment through a **state** representation $S_t \in \mathcal{S}$ which is an element of all possible states, the state space $\mathcal{S}$. To really interact with the environment the agent has to perform an **action** $A_t \in \mathcal{A}(S_t)$ where $\mathcal{A}(S_t)$ is the set of all possible actions in given state S_t. To chose an action the agent calculates for each possible action A_t a probability $\pi_t(A_t|S_t)$ for a given state S_t and chooses the maximum with respect to the given actions. To achieve a more compact notation we will refer to the action $A_t = a$, the state $S_t = s$ and therefore also the policy as $\pi_t(a|s)$. This probability distribution is called the **policy**. After the agent has chosen and executed an action, which also mean we will move one time step further $t \to t + 1$, the environment will return the next state S_{t+1} the agent will perceive and a **reward** $R_{t+1} \in \mathbb{R}$.
The long term goal of the agent is to adapt its policy to maximize the total amount reward the agent receives from the environment.

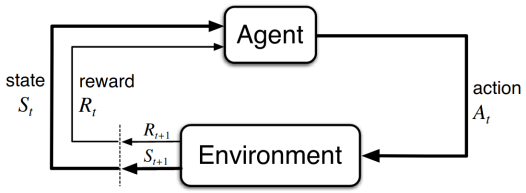


Figure 1: interaction between agent and the environment

(TODO: cite figure)

(TODO: Explain the concept of reinforcement learning and its applications Describe the Markov decision process (MDP) and the Bellman equation Discuss the challenges of using reinforcement learning in practice)

3.2 Soft Actor Critic

Briefly explain what the SAC algorithm is and why it is important Provide a preview of what the chapter will cover

L1 smooth loss for $x, y \in \mathbb{R}$

$$\mathcal{L}_\beta(x_i, y_i) = \begin{cases} \frac{1}{2\beta} \cdot (x_i - y_i)^2 & \text{if } |x_i - y_i| < \beta \\ |x_i - y_i| - \frac{\beta}{2} & \text{else} \end{cases} \tag{2}$$

3.2.1 Actor-Critic Algorithms

The basic actor-critic architecture is a type of reinforcement learning algorithm that consists of two components: an actor and a critic. The actor is responsible for selecting actions based on the current state, while the critic evaluates the quality of the actor's actions by estimating the expected return from the current state. The actor uses this feedback from the critic to adjust its policy and improve its performance.

On-policy learning and off-policy learning refer to different methods for updating the policy in reinforcement learning. In on-policy learning, the agent learns from the data generated by its current policy, while in off-policy learning, the agent learns from data generated by a different policy. On-policy learning can be more stable, but it may require more data to converge. Off-policy learning can be more efficient, but it can be more sensitive to the quality of the data.

The advantage of using a critic in the actor-critic algorithm is that it provides a more stable feedback signal than using only rewards. The critic estimates the expected return from a state, which takes into account the long-term consequences of actions. This allows the actor to learn from the critic's feedback and improve its performance more efficiently than if it only received reward signals. Additionally, the critic can help to generalize across different states and actions, improving the overall performance of the algorithm.

One limitation of the basic actor-critic architecture is that it can suffer from high variance and slow convergence due to the interaction between the actor and critic. This can be addressed through the use of techniques such as baseline subtraction and eligibility traces.

3.2.2 Soft Actor Critic Algorithm

3.2.3 SAC in Practice

In robotics, the SAC algorithm has been used to control robots in tasks such as grasping objects, locomotion, and manipulation. The SAC algorithm's ability to handle continuous action spaces makes it a suitable choice for robotic control, where fine-grained control is often required.

In game playing, the SAC algorithm has been used to train agents to play video games such as Atari and Super Mario Bros. The SAC algorithm's ability to balance exploration and exploitation makes it effective in game playing, where agents must learn to navigate complex environments and respond to changing conditions.

The chapter also discusses how the SAC algorithm can be used in continuous action spaces. The SAC algorithm uses a Gaussian distribution to represent the policy, which allows it to handle continuous action spaces. The use of a Gaussian distribution

Algorithm 1 Soft Actor Critic

Input: initial policy parameters θ, Q-function parameters ϕ_0, ϕ_1, empty replay buffer $\mathcal{D}$

Set target parameters equal to main parameters $\phi_{\text{target},0} \leftarrow \phi_0$, $\phi_{\text{target},1} \leftarrow \phi_1$

for i in number of epochs **do**

$s \leftarrow$ reset environment

for t number of timesteps **do**

$a \sim \pi_\theta(\cdot|s)$

s', r, $d \leftarrow$ execute a in environment

Store (s, a, r, s', d) in replay buffer $\mathcal{D}$

if d is true **then**

break and reset environment

end if

$s \leftarrow s'$

end for

if $|\mathcal{D}| >$ minimal buffer size **then**

for number of train iterations **do**

sample minibatch: $(s_\mathcal{B}, a_\mathcal{B}, r_\mathcal{B}, s'_\mathcal{B}, d_\mathcal{B}) = \mathcal{B} \leftarrow \mathcal{D}$

compute td target with $\tilde{a}'_\mathcal{B} \sim \pi_\theta(\cdot|s'_\mathcal{B})$

$$td(r_\mathcal{B}, s'_\mathcal{B}, d_\mathcal{B}) = r + \gamma \cdot d \cdot \left(\min_{i \in \{0,1\}} \left(Q_{\phi_{\text{target},i}}(s'_\mathcal{B}, \tilde{a}'_\mathcal{B}) \right) - \alpha \cdot \log \pi_\theta(\tilde{a}'_\mathcal{B}|s'_\mathcal{B}) \right)$$

Update Q-functions for parameters ϕ_i $i \in \{0, 1\}$ using:

$$\nabla_{\phi_i} \frac{1}{|B|} \sum_{k \in |\mathcal{B}|} \mathcal{L}_\beta \left(td(r_{\mathcal{B},k}, s'_{\mathcal{B},k}, d_{\mathcal{B},k}), Q_{\phi_i}(s_{\mathcal{B},k}, a_{\mathcal{B},k}) \right)$$

Update policy with $\tilde{a}_\mathcal{B} \sim \pi_\theta(\cdot|s_\mathcal{B})$ using:

$$\nabla_\theta \frac{1}{|\mathcal{B}|} \sum_{k \in |\mathcal{B}|} \min_{i \in \{0,1\}} Q_{\phi_i}(s_{\mathcal{B},k}, \tilde{a}_{\mathcal{B},k}) - \alpha \cdot \log \pi_\theta(\tilde{a}_{\mathcal{B},k}|s_{\mathcal{B},k})$$

Update α with target entropy H_{target} and $\tilde{a}_\mathcal{B} \sim \pi_\theta(\cdot|s_\mathcal{B})$ using:

$$\nabla_\alpha - \frac{\alpha}{|\mathcal{B}|} \sum_{k \in |\mathcal{B}|} \left(\log \pi_\theta(a'_{\mathcal{B},k}|s_{\mathcal{B},k}) - H_{\text{target}} \right)$$

end for

end if

end for

also enables the SAC algorithm to handle multimodal policies, where multiple actions can be appropriate for a given state.

Finally, the chapter discusses the limitations of the SAC algorithm and how they can be addressed. The SAC algorithm can suffer from high variance and slow convergence, especially in high-dimensional state spaces. This can be addressed through techniques such as batch normalization, automatic entropy adjustment, and value function regularization. The chapter also discusses future directions for research, such as incorporating model-based methods into the SAC algorithm to improve sample efficiency.

Conclusion: The Soft Actor Critic algorithm is a powerful approach to reinforcement learning that has been applied in a variety of domains, including robotics and game playing. Its ability to handle continuous action spaces makes it a suitable choice for many real-world applications. However, the SAC algorithm has limitations that must be addressed, such as high variance and slow convergence. These limitations can be addressed through the use of appropriate techniques and future research directions. The SAC algorithm is an exciting area of research with potential applications in many fields.

3.3 Neural Networks

Briefly explain what neural networks are and why they are important Provide a preview of what the chapter will cover

3.3.1 The Rise of Deep Learning

Explain how the development of new algorithms and the availability of large datasets led to a resurgence in interest in neural networks in the 2000s Describe the key

breakthroughs that made deep learning possible, including the development of convolutional neural networks and recurrent neural networks Provide examples of important applications of deep learning, such as image and speech recognition

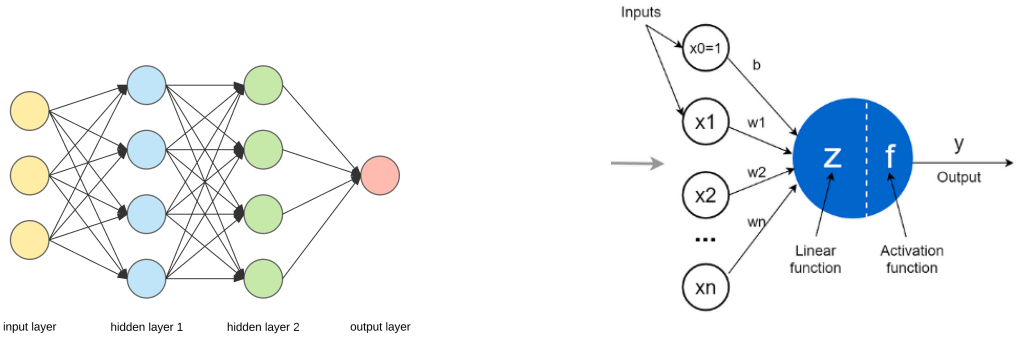
3.3.2 Neural Network Architecture

Neural networks are a type of machine learning model inspired by the structure and function by the cell type of neurons. It consists like their biological paradigm, of interconnected layers of individual units, called neurons.

The basic architecture as in Figure 2a, of a neural network includes an input layer (yellow), one or more hidden layers (blue and green), and an output layer (red). The input layer receives the input data, which is then passed through the hidden layers before producing the output. This is why we call this basic architecture a feed forward neural network. The number of nodes in each layer and the connections between them are dictated by the architecture of a network.

Each neuron as in Figure 2b, is designed in the same way. First it calculates a weighted z of its inputs x, the corresponding weights w and bias b before passing it on to an activation function h. This function is designed to introduce nonlinearity into the model, allowing it to capture complex relationships between variables otherwise the whole network would be a single linear combination of its inputs and weights. Common activation functions include the sigmoid function, the hyperbolic tangent function, and the rectified linear unit (ReLU) function as in Figure 3

There are several types of neural networks, including feedforward networks, recurrent networks, and convolutional networks. Feedforward networks are the simplest type of neural network, consisting of a series of layers that process information in a single direction. Recurrent networks, on the other hand, allow information to be passed between nodes in a cyclical manner, making them suitable for processing



(a) schematic drawing of a feed forward neural network

(b) schematic drawing of a single neuron

Figure 2: neural network schema drawing

(a) Sigmoid

(b) hyperbolic tangent function

(c) rectified linear unit

Figure 3: common activation functions in a neural network

sequential data. Convolutional networks are designed for image processing tasks and use convolutional layers to identify patterns and features within images.

Neural networks are trained using techniques such as backpropagation and stochastic gradient descent. Backpropagation is an algorithm for calculating the gradient of the error with respect to the weights of the network, which can then be used to update the weights and improve the performance of the model. Stochastic gradient descent is a technique for minimizing the error of the model by iteratively adjusting the weights based on randomly selected subsets of the training data. These techniques allow neural networks to learn from large datasets and make accurate predictions on new data.

3.3.3 Supervised Learning

3.3.4 Applications of Neural Networks

Neural networks are widely used in real-world applications like in computer vision, marketing or medical applications. Particularly in combination with reinforcement learning they are successfully used in robotics for object recognition and manipulation, in gaming for complex decision-making, and in finance for predicting stock prices.

3.4 Variational Autoencoder

In this chapter we want to provide a intuition about Autoencoders Based on the manifold hypothesis

appoaches to solve this by principal component analysis Is a technique of unsupervised learning

3.4.1 Introduction

Variational autoencoders (VAEs) and conditional variational autoencoders (CVAEs) are types of deep generative models that are used for unsupervised learning. Unsupervised learning is a type of machine learning where the model is not given labeled data for training. Instead, the model is tasked with finding patterns or structure in the data on its own. The goal of unsupervised learning is often to find hidden relationships or groupings within the data that can be used for further analysis or decision-making VAEs and CVAEs are important because they can learn to generate realistic and diverse samples from complex high-dimensional data distributions, such as images or audio.

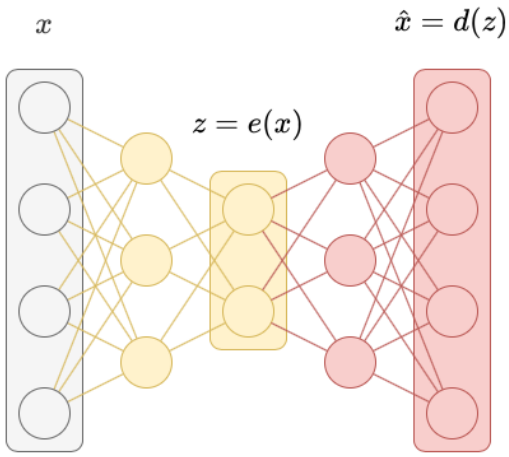


Figure 4: schematic drawing of a Autoencoder

3.4.2 Autoencoders

Lets start with a Autoencoder. An Autoencoder $f = d \circ e$ consists of two function approximators in series, one encoder e and one decoder d. In most cases these function approximators are resembled by neural networks.

In a forward pass through the module first the encoder maps the high dimensional input $x \in \mathbb{X}$ into a lower dimensional latent space $z \in \mathbb{Z}$, $z = e(x)$. This latent vector z contains with a trained encoder, an encoding of certain properties from the input x. Therefor similar input values x should correspond to similar latent vectors z.
In the second stage we map the latent vector z back into the high dimensional features space X, $d : \mathbb{Z} \to \mathbb{X}$. This should optimally reconstruct the given input $\hat{x} = d(z) = d(e(x))$.

3.4.3 Variational Autoencoders

A VAE is very similar to a basic Autoencoder. It also contains an encoder and a decoder. The key difference lays in the in the latent variable. Instead of feeding the

output from the encoder directly into the decoder the latent variable gets sampled from a parameterized distribution. The parameters for this distributions are the output from the encoder.

Explain the concept of a latent variable and how it relates to VAEs Describe the architecture of a VAE and how it differs from a traditional autoencoder Explain how the encoder and decoder are trained using a variational inference approach Describe the role of the Kullback-Leibler (KL) divergence in the loss function

3.4.4 Conditional Variational Autoencoders

Figure 5: schematic drawing of a conditional Variational Autoencoder

A Conditional Variational Autoencoder (CVAE) is an extension of the VAE architecture that takes into account conditional information. In a CVAE, both the encoder and decoder are modified to accept an additional input that represents the conditional information. This input is usually in the form of a label or a set of labels that describe the class or category to which the input belongs.

The architecture of a CVAE is similar to that of a VAE, with the addition of the conditional input. The encoder takes the input data and the conditional input, and maps them to a latent space representation. The decoder takes the latent space representation and the conditional input, and maps them back to the input space.

CVAEs can be used for supervised learning tasks, where the goal is to generate samples that belong to a specific category or class. In this case, the conditional input represents the class label or category to which the input data belongs. By conditioning the generation process on the class label, the CVAE can learn to generate samples that are representative of that class.

The role of the conditional input in the encoder is to help the model capture the conditional dependencies between the input data and the class label. The encoder must learn to map both the input data and the conditional input to a meaningful latent space representation that captures the relevant information for the generation process. In the decoder, the conditional input is used to guide the generation process and ensure that the generated samples are representative of the specified class.

3.4.5 VAEs and CVAEs in Practice

VAEs and CVAEs have been successfully applied to a wide range of real-world applications. In image generation, VAEs have been used to generate novel images of faces, objects, and scenes. Similarly, CVAEs have been used for conditional image generation, allowing for the generation of images based on specific attributes or classes.

In text generation, VAEs have been used to generate natural language sentences and paragraphs.

VAEs and CVAEs can also be used for data compression and denoising. By learning a compressed representation of the input data, VAEs and CVAEs can reduce the dimensionality of the input space while preserving important features. Similarly, by learning to reconstruct the original input from noisy or corrupted data, VAEs and CVAEs can be used for denoising and data restoration.

One of the advantages of VAEs and CVAEs is their ability to learn a continuous latent representation of the input data. This allows for easy manipulation and exploration of the latent space, enabling applications such as image editing and style transfer. Additionally, VAEs and CVAEs can be trained using stochastic gradient descent, making them computationally efficient for large datasets.

However, VAEs and CVAEs have some limitations. The generated samples may not be as sharp or detailed as those produced by other generative models such as GANs. Additionally, the trade-off between the reconstruction loss and the KL divergence term can be difficult to balance, potentially leading to overfitting or underfitting. Nonetheless, VAEs and CVAEs remain a popular and powerful tool for generative modeling and data compression.

3.5 Inverse Kinematics

Inverse kinematics (IK) is a fundamental problem in robotics, animation or virtual reality that involves finding a required joint angle configuration or positions to reach a desired end-effector position and optionally a desired orientation. It plays a crucial role in controlling the motion and manipulation of robotic systems, enabling them to interact with the environment and perform complex tasks. In this section you will find a brief summary of existing approaches, a deeper explanation of the Cyclic

Coordinate Descent algorithm and the description of the used inverse kinematics RL environment.

3.5.1 Existing Approaches to Solve Inverse Kinematics

(TODO: subsection created by chatGPT)

Numerous approaches have been proposed to solve the inverse kinematics problem. These approaches can be broadly categorized into analytical methods, numerical methods, heuristic methods, sampling-based methods, and machine learning-based methods.

Analytical Methods: Some robotic systems with simple geometries and kinematic structures can be solved analytically using closed-form solutions. Analytical methods often rely on geometric techniques and mathematical equations such as the Denavit-Hartenberg (DH) parameters to derive explicit formulas for the joint angles. These methods provide fast and exact solutions when applicable, but they are limited to specific cases with well-defined geometric relationships.

Numerical Methods: Numerical methods for inverse kinematics aim to iteratively approximate the joint angles that satisfy the desired end-effector position and orientation. Jacobian-based methods, such as the Jacobian transpose method and the Jacobian pseudo-inverse method, leverage the Jacobian matrix to relate the end-effector velocities to the joint velocities. These methods iteratively adjust the joint angles based on the discrepancy between the actual and desired end-effector positions. Other numerical techniques, including cyclic coordinate descent, Gauss-Newton, Levenberg-Marquardt, and optimization algorithms like Particle Swarm Optimization and Genetic Algorithms, aim to find optimal joint configurations by minimizing an objective function that represents the error between the desired and actual end-effector positions.

Heuristic Methods: Heuristic methods, such as the Forward and Backward Reaching Inverse Kinematics (FABRIK) algorithm, iteratively propagate positions along the kinematic chain to converge on the desired end-effector position. These methods provide an intuitive and efficient way to solve inverse kinematics problems and handle complex robotic structures with multiple degrees of freedom.

Sampling-Based Methods: Sampling-based methods, such as Randomized Kinodynamic Planning (RRT), adopt a randomized search strategy to explore the joint space and find feasible solutions to the inverse kinematics problem. By sampling and connecting valid configurations in the joint space, these methods can discover feasible joint angles that satisfy the end-effector constraints.

Machine Learning-based Methods: Machine learning approaches, including neural networks and reinforcement learning algorithms, have been increasingly explored for solving inverse kinematics. Neural networks can be trained to approximate the inverse kinematics mapping based on input-output data pairs. Reinforcement learning algorithms, such as Soft Actor-Critic (SAC), can learn inverse kinematics policies through trial-and-error interactions with the environment, optimizing joint configurations to achieve desired end-effector positions.

3.5.2 Cyclic Coordinate Descent

Cyclic Coordinate Descent (CCD) is a popular numerical method for solving inverse kinematics. It is an iterative algorithm that adjusts the joint angles of a robotic system one at a time, from a base joint to the end-effector, in order to align the end-effector with the desired target position.

The algorithm works by iteratively updating the joint angles based on the discrepancy between the current and desired end-effector positions (p_{target}). At each iteration, CCD focuses on a single joint and adjusts its angle to minimize the positional error.

By sequentially updating the joint angles in a cyclic manner, CCD aims to converge towards a solution that satisfies the desired end-effector position.

Algorithm 2 Cyclic Coordinate Descent Pseudo Code

Input: Current joint angles q, Desired end-effector position p_{target}.
while unitl convergence **do**
for each i in $[N-1, \ldots, 0]$ **do**
Calculate the vector from the current joint poisition to the end-effector position:
$V_{\text{current}} \leftarrow p_{N-1} - p_i$
Calculate the vector from the current joint position to the target position:
$v_{\text{target}} \leftarrow p_{\text{target}} - p_i$
Calculate the rotation necessary to align v_{current} with v_{target}:
$\delta q_i \leftarrow \text{angle\_between}(v_{\text{current}}, v_{\text{target}})$
Update the joint angel:
$q_i \leftarrow q_i + \delta q_i$
end for
end while

As illustrated in Algorithm 2 and Figure 6 in each iteration, CCD calculates the vector from the current joint position p_i with p_i as the position of the ith joint with $i \in \{0, \ldots, N-1\}$, to the end-effector position (v_{current}) and the vector from the current joint position to the target position (v_{target}). By finding the rotation necessary to align v_{current} with v_{target}, represented as $\delta\phi$, the algorithm updates the joint angle accordingly. This process is repeated for each joint in the kinematic chain until convergence is achieved.

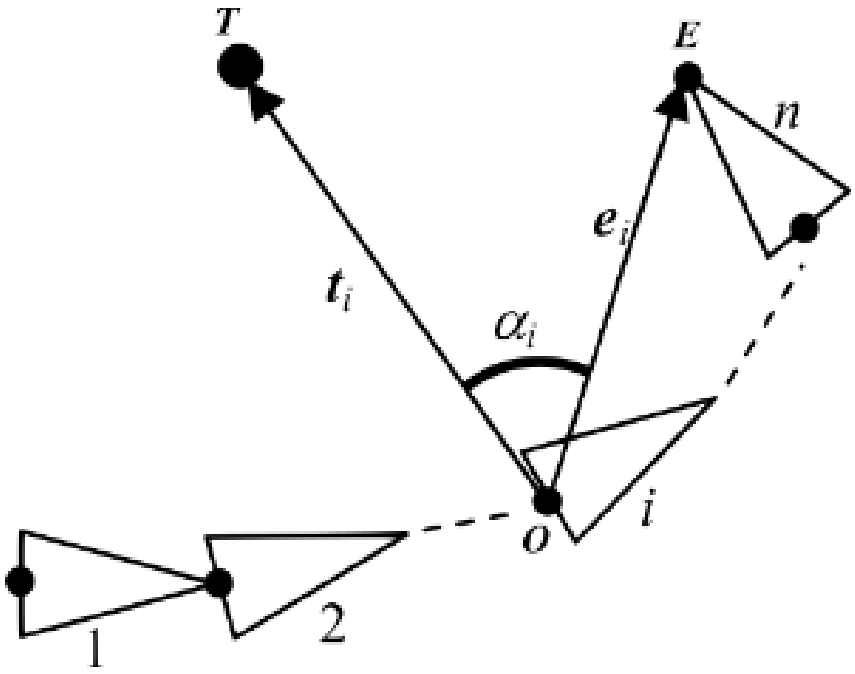


Figure 6: CCD geometry

(TODO: make own figure)

(TODO: experiments for runtime analysis in dependence of number of joints)

4 Approach

The approach starts with the problem definition and continues with what you have done. Try to give an intuition first and describe everything with words and then be more formal like 'Let g be ...'.

4.1 Problem Definition

The problem we concentrate is to solve inverse kinematics for a varying number of joints. This setting was chosen because it satisfies two properties. It has an scaleable action space and is easy to extendable to make it more complex so standard solver like Cyclic-Coordenate-Descent would fail **(TODO: show that CCD would fail)**. But before we started to make the environment more complex we started with solving inverse kinematics without any constraints.

4.1.1 the environment

4.2 Variational Autoencoder

In this section we will discuss the creation of different datasets to solve the inverse kinematics problem. **(TODO: very similar actions with CCD) (TODO: Show: there are multiple actions for a single target position, https://www.alanzucconi.com/2018/05/02/2d-1/ for two joints)**

4.2.1 Pure Actions

4.2.2 Conditioning on the States

(TODO: pseudo code)

4.2.3 Fitting Random Noise

4.3 Supervised learning

In this section we will discuss the approach to solve the inverse kinematics problem with supervised learning. This approach was only chosen to show that inverse kinematics is solveable with a neural network

distance function

$$\mathcal{L}(x, \hat{x}) = \frac{1}{2} \cdot \sum_{i \in \{0,1\}} (x_i - fk(\hat{x})_i) \tag{3}$$

fk is forward kinematics

4.4 SAC

4.4.1 MLP

4.4.2 RNN

4.4.3 imitation learning

4.4.4 hyperparamter tuning

(TODO: make a description about the tuning of action magnitude and target entropy)

4.4.5 strategic vs one shot

(TODO: look at the code and)

4.4.6 latent actor

4.4.7 Problems with exploration

4.4.8 actions with constraints

5 Experiments

5.1 VAE

In the folling section I will present the results of my experiments with the Variational-Autoencoder to encode and decode actions from the environment. Consistent over all experiments with the VAE sizes of the datasets are consistent:

- train: 10.000 actions
- validation: 2000 actions
- test: 1000 action

5.1.1 Pure Actions

In every state of the sequential decision making process there are multiple actions available to go from the current state s_t to the next state s_{t+1} where the arm end position is at the goal position. To gain a dataset that contains such actions I sampled the robot arm angles in s_t from a uniform distribution $\mathcal{U}_[0, 2\pi]$ and applied the CCD algorithm to get access to the action that leads from s_t to s_{t+1} with the robot arm end position near the goal state.

The results of the training process are shown in Figure **??**

n	latent dim	r loss	kl loss
2	2		
2	1		
5	5		
5	4		
5	3		
5	2		
5	1		
10	2		
10	2		
10	2		
10	2		

Table 1: n is the number of joints, latent dimension is the size of the dimension the action is compressed to, r loss is the test reconstruction loss I used the MSE as the metric for the reconstruction loss, kl loss is the Kullback Leiber diveregence from the standard normal distribution on the test dataset

5.1.2 Conditioning on States

After watching the pure action encoding and decoding fail in the all important reconstruction loss. I got inspired by the paper **(TODO: cite LASER)** and rerolled the experiments with an conditional Variational-Autoencoder where the conditional information is the information from s_t. The results of the training process are shown in Figure **??**

We can observe that the test reconstruction loss is much lower that before and the kl loss

(TODO: Concering the table: make a lot of experiments so I can get an std ?, or should I just say that the training turns out to be not as stable as it should be in comparison to an VAE on MNIST and that we sampled runs until we got a good result?)

5.1.3 Fitting Random Noise

In this section I will present the results on how we tried to encode and decode actions sampled from a parameterized distributuion. The idea is that in every state the agent can choose an action from $[-1, 1]^n$. Therefor I sampled a dataset independent and identically distributed from $\mathcal{U}^n_{[-1,1]}$. this could be describe of trying to fit random noise. The results for 800 epochs are shown in Figure **??**.

5.2 SAC

5.2.1 Hyperparameters

(TODO: description?)

5.2.2 Baseline

In this section I want to present the results of the baseline experiments. To mimic a sequential decision making process and constrain the actions I manually tuned the parameter "action magnitude" between 1, 0.5, 0.2, and 0.1. The results are shown in figures: Figure **??**

Stichworte die rein sollen - Mit erhörter Anzahl an joint wid die Performance ober alle experimente immer schlechter - Affällig hohe Varianz bei kleiner werdeneder Anzahl joint - Absacken der Performance von 2 joints mit weiter Erniedrigung der action magnitude - 5 joints scheinen eine ähnliche Performance über die action magnitude abzu geben - Scheinbar habe ich bei 0.2 einen Wert gefunden der auch für höhere Anzahlen von joints gut zu funktionieren scheint. -> Fahre daher weiter mit diesem Parameter fort.

name	values
task	ReachGoalTask, ImitationTask
lr pi	0.0005
lr q	0.001
init alpha	0.01
gamma	0.98, 0
batch size	32
buffer limit	50000
start buffer size	1000
train iteration	20
tau	0.01
target entropy	-40.0
lr alpha	0.001
n epochs	5000
action covariance mode	independent
action covariance decay	0.5
action magnitude	0.1

Table 2: Hyperparameters for the Soft-Actor-Critic algorithm

6 Conclusion

7 Acknowledgments

First and foremost, I would like to thank Jasper Hoffman for his support.

I also thank Jan Ole von Harzt and Jens Rahnfeld for great discussions about Variational Autoencoders

- advisers
- examiner
- person1 for the dataset
- person2 for the great suggestion
- proofreaders

Bibliography

[1] D. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[2] J. Bromley, J. W. Bentz, L. Bottou, I. Guyon, Y. LeCun, C. Moore, E. Säckinger, and R. Shah, "Signature verification using a "siamese" time delay neural network," *International Journal of Pattern Recognition and Artificial Intelligence*, vol. 7, no. 04, pp. 669–688, 1993.

[3] M. Muja and D. G. Lowe, "Fast approximate nearest neighbors with automatic algorithm configuration.," *VISAPP (1)*, vol. 2, no. 331-340, p. 2, 2009.