Hochschule Fulda
University of Applied Sciences

A study of reinforcement learning algorithms in simulated robotics scenarios

Alejandro Pajares Chirre

Masters Thesis submitted to the Faculty of AI at HS Fulda

Matriculation No: 1331534
Supervisor: Prof. Dr. Alexander Gepperth
Co-Supervisor: Prof. Dr. David James

Submitted on dd.mm.yyyy

Abstract

The world of robotics has seen significant advancements in recent years, and this is largely due to the integration of machine learning techniques. Robots are now able to learn from their surroundings, make decisions, and carry out tasks with minimal human intervention. Machine learning has enabled robots to interact with humans and perform tasks that were previously considered impossible. In particular, reinforcement learning (RL) is a type of machine learning that models how humans learn from sensory input and motor responses in response to rewards. RL is based on the idea that an agent interacts with an environment by taking actions and receiving feedback in the form of rewards or punishments. Q-learning is a popular algorithm used in RL to learn the optimal policy, i.e., the best sequence of actions to maximize reward, for an agent. This thesis focuses on the application of reinforcement learning (RL) and Q-learning algorithms in controlling the motion of three joints of a six-degree-of-freedom (6-DOF) robotic arm in a simulated environment. To apply the Q-learning algorithms, the problem needs to be modeled as a Markov Decision Process, and during the learning process, the exploration and exploitation rate need to be balanced. The robotic arm is modeled in Gazebo, and the control commands are sent using the Robot Operating System (ROS 2). The objective of the robotic arm is to touch the target. The RL algorithm learns to maximize the reward function, which is based on the current and previous distance between the target and one end of the robotic arm and the angles of the three joints being controlled. The experimental results demonstrate that RL and Q-learning algorithms can effectively control the motion of a robotic arm in a simulated environment. The robotic arm successfully learns to approach and touch the target.

Contents

List of Figures

List of Tables

1 Introduction

Robotic systems are increasingly being used in a wide range of applications, from manufacturing and logistics to healthcare and entertainment. However, the effectiveness of these systems depends on their ability to perform tasks accurately and efficiently, which requires the development of intelligent control strategies that can adapt to different environments and objectives. Reinforcement learning offers a promising approach to achieve this goal, by enabling robots to learn from feedback and optimize their behavior based on the rewards received from the environment. Numerous companies have embraced the use of robotics to aid humans in tasks that are monotonous, physically demanding, or hazardous. Yet, acquiring a robot and hiring a robotic engineer to create a tailored solution for a particular task requires a significant investment of resources. The duties of a robot engineer include setting up communication, designing control scripts, computing coordinate transformations, and creating error-handling programs. Typically, a technician takes on the task of operating the robot on a daily basis, or the robot operates on its own. However, if the task requirements or processes change, it is difficult to modify the existing robotic solution to suit a new configuration or application without the assistance of a robot engineer, despite the significant resources invested in acquiring and developing it. Instead of relying on a robotic engineer to manually program a robot's operations for a new application, companies could employ deep reinforcement learning to train an intelligent agent to control the robot specifically for that application. This approach would enable the resources invested in robotics to be more adaptable and versatile, suitable for a broader range of applications and purposes.

This thesis is about reinforcement learning which is a type of machine learning that involves an agent learning from its interactions with an environment in order to maximize a reward signal over time [1]. It is a method of learning that involves trial and error, with the agent receiving feedback in the form of rewards or punishments for its actions. According to Kaelbling, Littman, and Moore [2], reinforcement learning can be defined as "a problem faced by an agent that learns behavior through trial-and-error interactions with a dynamic environment". Many different concepts and methodologies can be used to break down reinforcement learning. This work focuses on Deep Q learning and to study it a simulated robotic arm is trained to touch a can.

1.1 Context

What is the context of the presented work? If you work in a company, present the company first. Then the broad scientific or technological background in which the work is embedded should be presented and explained.

1.2 Problem statement

What is the problem that is being addressed or solved? Why is it important or beneficial to find a solution to this problem?

1.3 Goals

Here, give a list of bullet points of quantifiable goals of the presented work. These achievement of these goals will be shown in the experiments section. The list should have 3-4 entries. No blabla here, hard goals!

1.4 Related Work

Deep Q-learning has been successfully applied in various robotic control tasks, including manipulators and robotic arms. In this section, we summarize the results and contributions of six relevant papers on deep Q-learning for robotic arm control.

In [3], the authors proposed a method for robotic grasping using deep Q-networks (DQNs). The results showed that the proposed method outperforms other traditional grasping methods in terms of grasping success rate and efficiency.

In [4], the authors presented a framework for real-time control of a robotic arm using deep reinforcement learning. They used a DQN-based algorithm to learn the control policy for the robotic arm, and the results demonstrated that the proposed method can achieve accurate and robust control.

List works that have a similar goals, plus a short explanation (3-4 sentences at most) as to how they differ from your work. Do NOT make comparisons here (better than my work or similar), that happens in the discussion section.

Admissible related work is (in descending order of acceptability):

- Peer-reviewed scientific publications, ideally with a DOI. Use Google scholar for searching (GS can export BibTeX entries that you can copy into the .bib file of this project).
- White papers and publicly available documents without review, cite with title, URL and date of access. In addition, you need to submit the PDFs in electronic form.
- Web pages, especially for software projects (e.g., TensorFlow, nginx, react, Django). Cite via URL and date of access. A github/gitlab/etc link is acceptable as well. Nothing needs to be submitted electronically, but only use such sources of there is no other way.

Literature is cited like this: as shown in clemen1989combining, blablaba. Or: [**?**] has a similar scope in the domain of perverted numerical integration, however without considering the aspect of cupidity. Or: In [**?**], a study of perverted diagonal matrix perversions is presented. You need not include page numbers.

See also Kap. 7.9, 7.10.

1.5 Contribution

Here, you present a bullet ist of your personal contributions to the topic of the thesis. For example:

- Implementation of a bash script that did not work and was hard to read
- Comparison of different implementations for matrix perversion
- Implementation of a web service that provides jokes about professors via a ReST API.

2 Foundations

The targeted group are computer scientists with at least a Bachelor's degree. Here, you explain aspects that go beyond what this group would not usually know. For example:

- Specialized libraries and their use
- Complex concepts of the chosen programming language
- Basics of machine learning, or neural networks, or both
- Description of used databases or datasets
- rviz
- rqt
- urdf file
- sdf file
- stl file

In subsequent chapters, you can reference this one to avoid having to explain everything over and over again. This means that you just include things here that are necessary for the understanding of later chapters, nothing more.

2.1 Topic 1

2.2 Topic 2

2.3 Topic 3

3 Implementation

3.1 Robotic Arm

The robotic arm used in the simulation is the Kuka KR210. The simulation was taken from https://github.com/udacity/RoboND-Kinematics-Project. To use the simulated arm, a ROS2 package was created and after that the files of type *URDF*, *DAE*, and *STL* were copied into our package. To make the robotic arm compatible with ROS2, the necessary plugins ros2_control and gazebo_ros2_control were added to the *URDF* file.

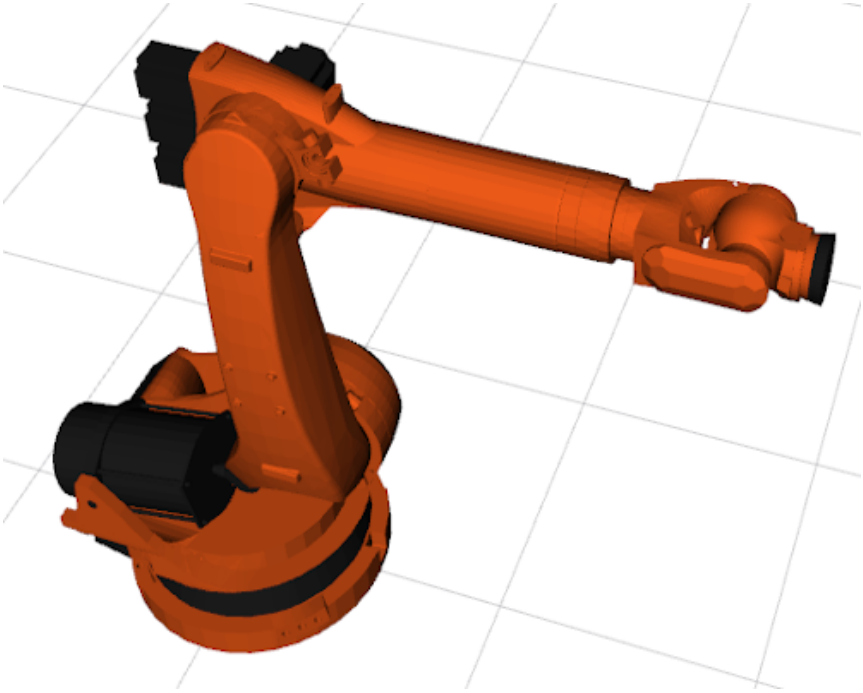


Figure 1: Kuka KR210 in Rviz2.

The robotic arm consists of six joints and seven links that connect them. The six joints of the KR210 are numbered J1 to J6, and they allow the robot to move in various directions and orientations.

- J1: The first joint is the base joint, which allows the robot to rotate horizontally around its vertical axis.

- J2: The second joint is the shoulder joint, which allows the robot to lift and lower its arm vertically.
- J3: The third joint is the elbow joint, which allows the robot to bend its arm vertically.
- J4: The fourth joint is the wrist roll joint, which allows the robot to rotate its wrist around its vertical axis.
- J5: The fifth joint is the wrist pitch joint, which allows the robot to tilt its wrist up and down.
- J6: The sixth joint is the wrist yaw joint, which allows the robot to rotate its wrist horizontally.

The arm also has links that connect the joints, including the base, lower arm, upper arm, wrist, and end-effector. These links are designed to provide strength and rigidity to the robot arm while allowing for smooth and precise movement. These links are:

- Base Link: This is the fixed part of the robot that is attached to the ground.
- Link 1: This is the first link that connects the base to joint 1.
- Link 2: This link connects joint 1 to joint 2.
- Link 3: This link connects joint 2 to joint 3.
- Link 4: This link connects joint 3 to joint 4.
- Link 5: This link connects joint 4 to joint 5.
- End Effector (Link 6): This is the final link that connects to the tool or object being manipulated.

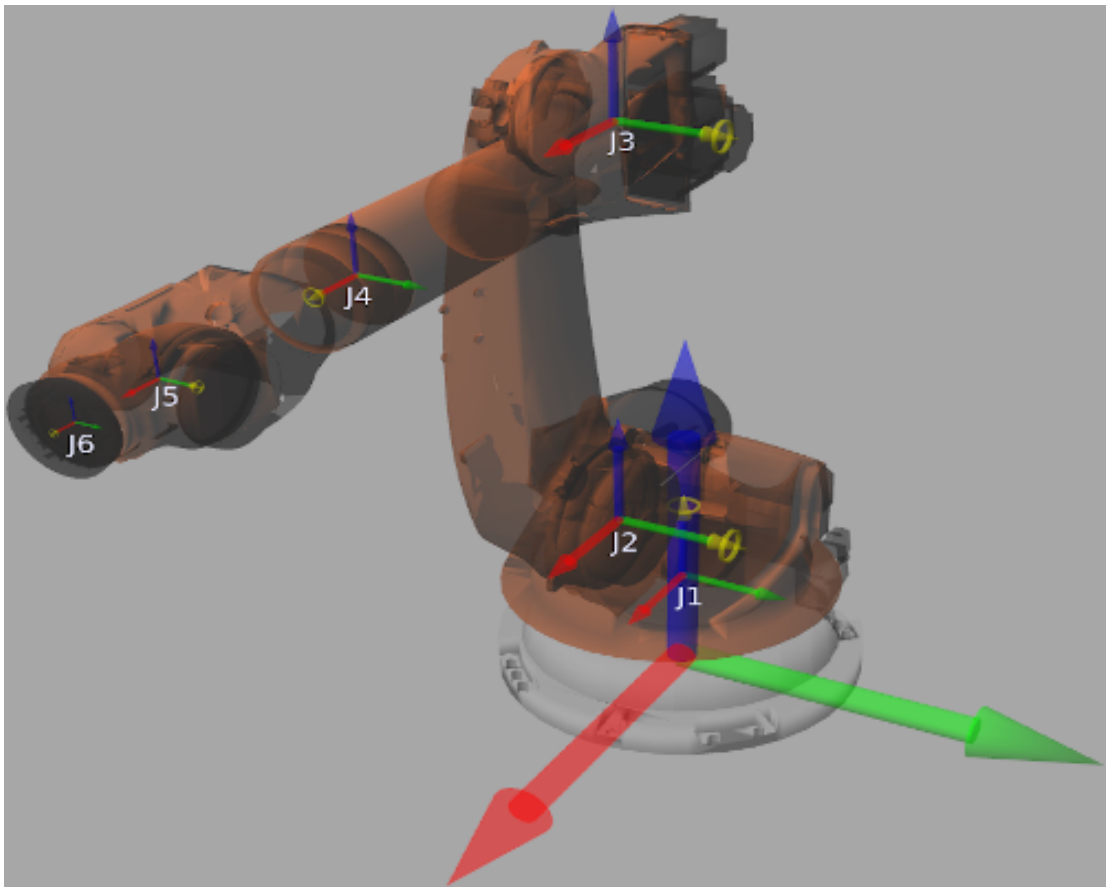
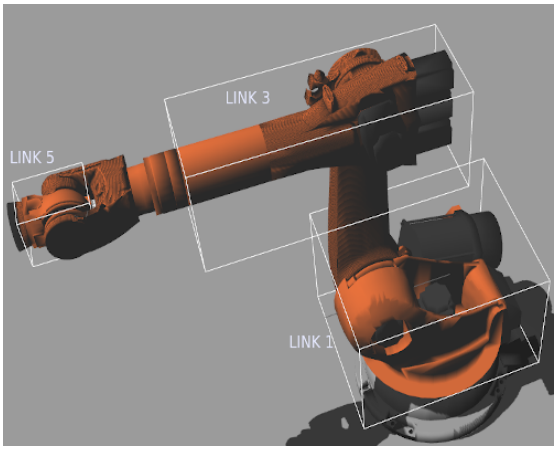
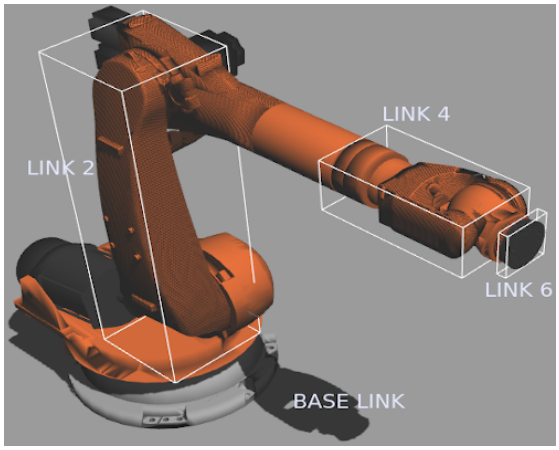


Figure 2: long caption



(a) subcaption.



(b) subcaption.

Figure 3: caption.

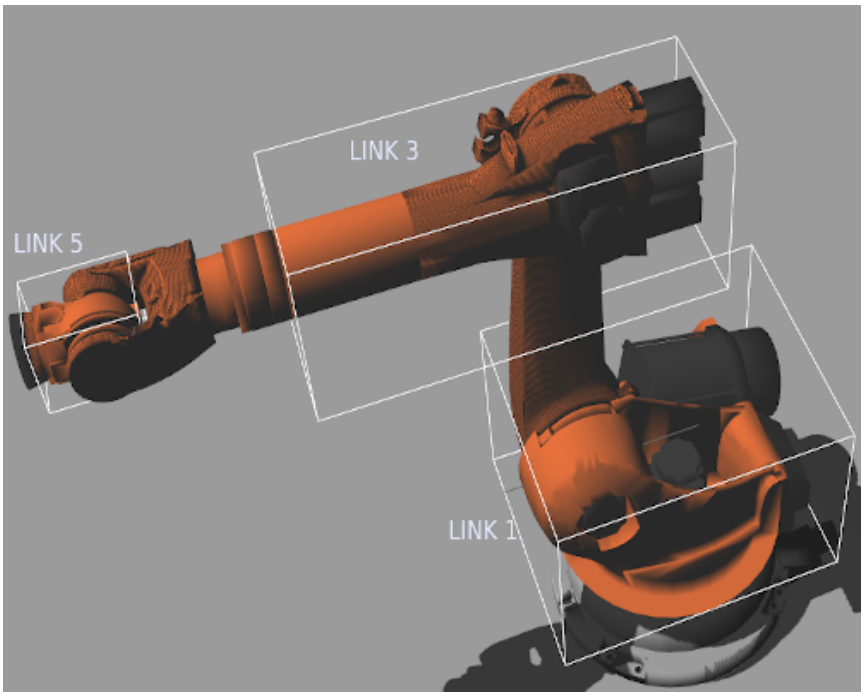


Figure 4: long caption

A Bumper sensor was added to links 4, 5, and 6 to detect collisions with these links (As shown in the code snippet) to the URDF.

A Camera sensor is added (As shown in the code snippet) to the URDF.

The object the robotic arm has to learn to touch is a can taken from gazebo models (http://models.gazebosim.org/beer). A world file is created containing the can.

The Gazebo ROS state plugin was added (as shown in 3) to the world file to have the positions, velocities, and other properties from our models published and to be able to modify them programmatically.

3.2 Moving the robotic arm

The AnglesPublisher node was created to move the arm by publishing the target angles for the joints. To do this the Joint Trajectory Controller is used, this controller generates and executes trajectories for the robot joints. It subscribes to a JointTrajectory message that specifies the desired trajectory, and then generates a control signal to move the robot's joints along the trajectory. By publishing messages of type JointTrajectory to the joint_trajectory topic the arm is moved.

3.3 Getting Images from the camera

The CameraSubscriber node was created to get images from the environment. The plugin previously used takes care of publishing the images to the topic /camera/image_raw,

the CameraSubscriber just subscribes to this topic and gets the images as messages of type Image.

ROS control module is used to move the arm. ROS Control is a framework for building and controlling robots in ROS (Robot Operating System). It provides a standardized way to manage hardware interfaces, controllers, and state machines, making it easier to develop and integrate robot applications.

The Joint State Broadcaster is a ROS node that broadcasts the state of robot joints. It reads joint positions, velocities, and efforts from a robot's hardware interfaces or controllers and publishes them as a JointState message on a ROS topic. This message contains the current state of all joints in the robot's model, allowing other ROS nodes to subscribe and use this information.

The Joint Trajectory Controller is a ROS controller that generates and executes trajectories for robot joints. It subscribes to a JointTrajectory message that specifies the desired trajectory, and then generates a control signal to move the robot's joints along the trajectory. It can use different algorithms to generate this control signal, such as PID, computed-torque, or model-predictive control. This controller is often used in combination with the Joint State Broadcaster to provide closed-loop control of the robot's joints.

Overall, the Joint State Broadcaster and Joint Trajectory Controller are two important components of ROS Control that enable the control of robot joints. The Joint State Broadcaster provides the current state of the joints to other ROS nodes, while the Joint Trajectory Controller generates and executes trajectories for the joints based on desired goals. Together, these components provide a powerful toolset for controlling robot motion in ROS.

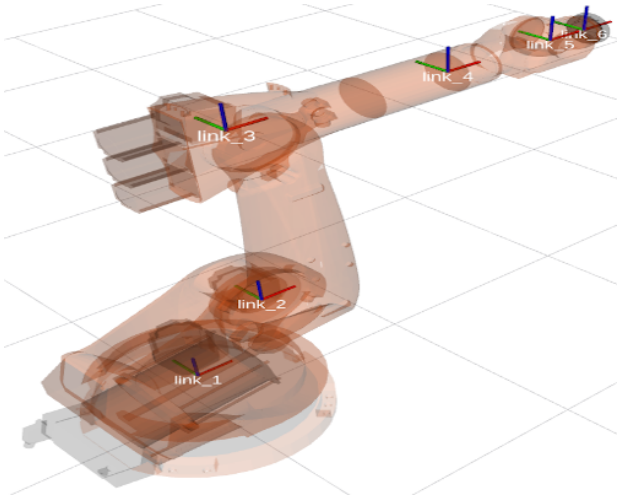


Figure 5: long caption

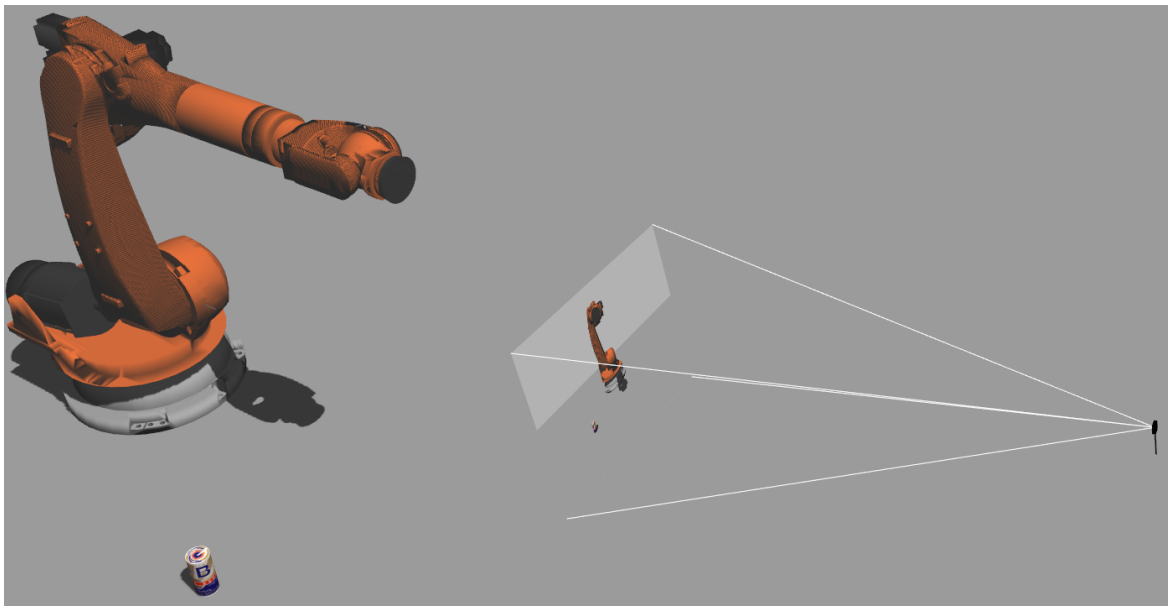


Figure 6: long caption

Figure 7: long description

Should refer, where possible, to the preceding chapter, e.e.: Singular value decomposition of the matrix Σ is conducted as explained in Sec. **??** using the *lapack* library (see Sec. 2.2).

For software development: what is the logic of the developed code, which of it was done by yourself? Sequence diagrams or UML are good tools here.

Please give code snippets only if they take up less that 0.25 pages, and only if it is unavoidable. Longer snippets go to the appendix and are referenced like this: see App. **??**.

4 Experiments

Show here that the goals from the introduction were achieved (or not achieved), you need at least one experiment per goal. Use screenshots, diagrams, plots, photos, etc. as necessary.

5 Discussion

2-3 pages are a good idea here. Picks up goals from the introduction (see 1.3) and experiments (see 4) and explains what was achieved and what was not (and why not in this case). Compares results with results from related work, see Sec. **??**. Draws a preliminary conclusion for the whole thesis.

6 Conclusion

Give an executive summary for important decision makers here, as well as an outlook (what would you do if you had another 3 months). 2-3 pages are ok here.

7 Using LaTeX, erase this chapter later

I was too lazy to translate this, it will be translated later. But I believe the ideas are clear!

7.1 Mathematische Gleichungen

Eine mehrzeilige Gleichung sieht so aus (die Symbole nach den und-Zeichen werden untereinander gesetzt). Die nonmber-Befehle verhindern dass die Gleichung nummertiert wird (Geschmackssache, ist nie falsch wenn eine Gleichung nummeriert ist). Aber: eine Gleichung auf die man refernziert (also die ein Label hat), muss nummeriert sein!

$$\begin{aligned} A &= \sum_{i=1}^{N} x_i \\ B &= \frac{\pi}{2} \end{aligned} \tag{1}$$

Eine inline-Gleichung: $x = 45b + \frac{2}{3}\pi$. Der Text geht weiter! Auf inline-Gleichungen kann man keine Refernzen erstellen.

7.2 Das ist eine Auflistung

1. Element 1
2. Element 2

7.3 Das ist eine Bullet-Liste

- Element 1
- Element 2

7.4 Eine Grafik bindet man so ein

Zulässige Formate sind generell eps, pdf und png.

Figure 8: Logo der HAW Fulda

7.5 So schreibt man einen Algorithmus

Algorithm 1: How to write algorithms

```
Data: this text
Result: how to write algorithm
initialization;
while not at end of this document do
    read current;
    if understand then
        go to next section;
        current section becomes this one;
    else
        go back to the beginning of current section;
    end if
end while
```

7.6 So gestaltet man eine Tabelle

Table 1: Beispielstabelle

A	B	C
D	per gram	11.65
	each	1.01
E	stuffed	32.54
F	stuffed	73.23
G	frozen	8.39

7.7 Interne Referenzen

So wird ein Kapitel oder Unterkapitel referenziert: Kap. 1, Kap. 7.10. Auf Gleichungen bezieht man sich so: Wie in Gl. (1) gezeigt, sehen Gleichungen in der Regel gut aus. Auf Abb. 8 bezieht man sich so. Auf Tab. 1 referenziert man so. Algorithmen sind analog: siehe Alg. 1. Generell kann man alles zitieren was ein Label hat.

7.8 Textformatierung

So wird dick geschrieben und *so kursiv*.

7.9 Zitieren

Generell zitiert man so: wie in [**?**] gezeigt, blablaba. Für jedes zitierte Werk ist ein BibTex-Eintrag nötig! Eine gute Quelle ist Google Scholar!!

7.10 Webquellen zitieren

So wird eine Webquelle zitiert: [5], siehe auch den Eintag im BibTeX-File. Wichtig: für jede Web-Quelle ein BibTeX-Eintrag! Wenn Sie das auf die hier gezeigte Art machen, werden URLs (fast) automatisch getrennt. Kontrollieren Sie trotztdem die Literaturliste, es kann sein dass das nicht immer funktioniert.

7.11 Literaturverzichnis erstellen

Hierzu müssen BibTeX-Einträge in die Datei literatur.bib eingefügt werden. Die BibTeX-Keys sind jeweils Argumente für die cite-Kommandos! Wenn Sie literatur.bib ändern müssen Sie alles mindestens 5x compilieren: 3x mit latex, 1x mit BibTex und dann noch 2x mit LaTeX (in der Reihengfolge). Am besten Sie machen ein Skript dafür!

References

[1] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 2018.

[2] M. L. Littman L. P. Kaelbling and A. R. Moore. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, 4:237–285, 1996.

[3] Sergey Levine, Peter Pastor, Alex Krizhevsky, Julian Ibarz, and Deirdre Quillen. Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection. *The International Journal of Robotics Research*, 37(4-5):421–436, 2018.

[4] Shixiang Gu, Ethan Holly, Timothy Lillicrap, and Sergey Levine. Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, page 3389–3396. IEEE Press, 2017.

[5] RStudio. Welcome to shiny. `https://shiny.rstudio.com/tutorial/written-tutorial/lesson1/`.

A Code Snippets

```
<gazebo reference="link_4">
  <!-- contact sensor -->
  <sensor name="end_effector_sensor" type="contact">
    <selfCollide>true</selfCollide>
    <alwaysOn>true</alwaysOn>
    <update_rate>500</update_rate>
    <contact>
      <collision>link_4_collision</collision>
    </contact>
    <!-- gazebo plugin -->
    <plugin name="gazebo_ros_bumper_sensor" filename="libgazebo_ros_bumper.so">
      <ros>
        <namespace>contact_sensor</namespace>
        <remapping>bumper_states:=bumper_link_4</remapping>
      </ros>
      <frame_name>link_4</frame_name>
    </plugin>
  </sensor>
</gazebo>
```

Listing 1: Bumper Sensor

```
  <gazebo reference="camera_link">
    <material>Gazebo/Black</material>
    <sensor name="camera" type="camera">
      <pose>0 0 0 0 0 0</pose>
      <visualize>true</visualize>
      <update_rate>10</update_rate>
      <camera>
        <horizontal_fov>1.089</horizontal_fov>
        <image>
          <format>R8G8B8</format>
          <width>640</width>
          <height>480</height>
        </image>
        <clip>
          <near>0.05</near>
          <far>8.0</far>
        </clip>
      </camera>
      <plugin name="camera_controller" filename="libgazebo_ros_camera.so">
        <frame_name>camera_link_optical</frame_name>
      </plugin>
    </sensor>
  </gazebo>
```

Listing 2: Camera Sensor

```
<plugin name='gazebo_ros_state' filename='libgazebo_ros_state.so'>
  <ros>
    <namespace>/gazebo_state</namespace>
    <argument>model_states:=model_states_demo</argument>
    <argument>link_states:=link_states_demo</argument>
  </ros>
  <update_rate>1.0</update_rate>
</plugin>
```

Listing 3: Gazebo ROS state plugin

```
  <gazebo>
    <plugin filename="libgazebo_ros2_control.so" name="gazebo_ros2_control">
      <robot_sim_type>gazebo_ros2_control/GazeboSystem</robot_sim_type>
      <parameters>/home/ros/ros2-projects/my-workspace/src/kuka_kr210/config/jtc.yaml</para
    </plugin>
  </gazebo>
```

Listing 4: Gazebo plugin and ROS2 controller configuration file.

```
<ros2_control name="GazeboSystem" type="system">
  <hardware>
    <plugin>gazebo_ros2_control/GazeboSystem</plugin>
  </hardware>
  <joint name="joint_1">
    <command_interface name="position">
      <param name="min">-3.14</param>
      <param name="max">3.14</param>
    </command_interface>
    <command_interface name="velocity">
      <param name="min">-3.15</param>
      <param name="max">3.15</param>
    </command_interface>
    <state_interface name="position"/>
    <state_interface name="velocity"/>
    <state_interface name="effort"/>
    <param name="initial_position">0.0</param>
  </joint>
  <joint name="joint_2">
    <command_interface name="position">
      <param name="min">-3.14</param>
      <param name="max">3.14</param>
    </command_interface>
    <command_interface name="velocity">
      <param name="min">-3.15</param>
      <param name="max">3.15</param>
    </command_interface>
    <state_interface name="position"/>
    <state_interface name="velocity"/>
    <state_interface name="effort"/>
    <param name="initial_position">-1.57</param>
  </joint>
  <joint name="joint_3">
    <command_interface name="position">
      <param name="min">-3.14</param>
      <param name="max">3.14</param>
    </command_interface>
    <command_interface name="velocity">
      <param name="min">-3.15</param>
      <param name="max">3.15</param>
    </command_interface>
    <state_interface name="position"/>
    <state_interface name="velocity"/>
    <state_interface name="effort"/>
```

```
      <param name="initial_position">0.0</param>
    </joint>
    <joint name="joint_4">
      <command_interface name="position">
        <param name="min">-3.14</param>
        <param name="max">3.14</param>
      </command_interface>
      <command_interface name="velocity">
        <param name="min">-3.2</param>
        <param name="max">3.2</param>
      </command_interface>
      <state_interface name="position"/>
      <state_interface name="velocity"/>
      <state_interface name="effort"/>
      <param name="initial_position">-1.57</param>
    </joint>
    <joint name="joint_5">
      <command_interface name="position">
        <param name="min">-3.14</param>
        <param name="max">3.14</param>
      </command_interface>
      <command_interface name="velocity">
        <param name="min">-3.2</param>
        <param name="max">3.2</param>
      </command_interface>
      <state_interface name="position"/>
      <state_interface name="velocity"/>
      <state_interface name="effort"/>
      <param name="initial_position">0.0</param>
    </joint>
    <joint name="joint_6">
      <command_interface name="position">
        <param name="min">-3.14</param>
        <param name="max">3.14</param>
      </command_interface>
      <command_interface name="velocity">
        <param name="min">-3.2</param>
        <param name="max">3.2</param>
      </command_interface>
      <state_interface name="position"/>
      <state_interface name="velocity"/>
      <state_interface name="effort"/>
      <param name="initial_position">0.0</param>
    </joint>
  </ros2_control>
```

B Thesis defence

The defence is 15/20 minutes for Bachelor/Master, followed by questions and a discussion. Both examiners are present, and you can invite external persons since defences are generally public.

Targetted group are non-computer scientists, e.g., from higher management, NOT the examiners. Means that at least $\frac{1}{3}$ if the presentation is introduction/context/problem statement. You should re-use text/images/graphs/etc from the corresponding chapters here!

1 Slide per minute is a good guideline. If you can guess that some questions are going to be asked anyway, prepare some slides specifically for these questions, makes a good impression, and you can show them in the discussion time, not during the 15 minutes of the presentation.

Defences are not graded, you can only pass or not pass.

Students are responsible for finding dates for the defence and coordinating this with both supervisors.

Some common advice is:

- Speak slowly and loadly
- If you do not have enough time left for all slides, leave some out rather than rushing through all of them!!
- Slide numbers!
- In presence: be there 10 minutes ahead of time to check projectors etc. Makes a very bad impression if this is not working. Same for online presentations: be there 5 minutes ahead of time to verify screen sharing works.
- do not read text from the slides. These should contains key words only, and you explain the rest in free presentation
- Defences can by all means be online, more convenient for companies
- in presence: always carry a USB key with a PDF of your slides. If you have to use another PC than yours, PowerPoint slides may look very differently (fonts, page setup etc.)
- No-Go: spelling errors on slides!!!
- Do not use animations, they may not work in an online setting

C Extras

C.1 Markov Decision Process

A Markov Decision Process (MDP) is defined by a tuple $< S, A, P, R, \gamma >$, where:

- S is the set of states in the environment
- A is the set of actions that can be taken in each state
- P is the state transition probability matrix, where $P^a_{ss'} = \mathbb{P}[S_{t+1} = s' \mid S_t = s, A_t = a]$
- R is the reward function, where $R^a_s = \mathbb{E}[R_{t+1} \mid S_t = s, A_t = a]$
- γ is the discount factor, where $\gamma \in [0, 1]$

The goal of an agent in a Markov Decision Process is to find a policy $\pi : S \to A$ that maximizes the expected discounted reward:

$$V_\pi(s) = \mathbb{E}\pi \left[\sum k = 0^\infty \gamma^k R_{t+k+1} \middle| S_t = s \right]$$

or the corresponding action-value function:

$$Q_\pi(s, a) = \mathbb{E}\pi \left[\sum k = 0^\infty \gamma^k R_{t+k+1} \middle| S_t = s, A_t = a \right]$$

C.2 Q learning Algorithm

The Q-learning algorithm is an off-policy temporal difference learning algorithm for finding the optimal action-value function $Q^(s, a)$ in a Markov Decision Process (MDP). The algorithm updates an estimate of $Q^(s, a)$ by iteratively applying the following update rule:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \Big[R_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t) \Big]$$

where s_t and a_t are the current state and action, R_{t+1} is the reward received after taking action a_t in state s_t and transitioning to state s_{t+1}, α is the learning rate, and γ is the discount factor.

The Q-learning algorithm can be summarized as follows:

1. Initialize the Q-value function $Q(s, a)$ for all state-action pairs.
2. Observe the current state s_t.

3. Choose an action a_t based on a policy, such as ϵ-greedy or softmax.
4. Take the action a_t and observe the next state s_{t+1} and reward R_{t+1}.
5. Update the Q-value function using the update rule: $Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \Big[R_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t) \Big]$.
6. Set $s_t = s_{t+1}$ and repeat from step 3 until the end of the episode or termination of the task.

The Q-learning algorithm is guaranteed to converge to the optimal action-value function $Q^*(s, a)$ under certain conditions, such as the MDP being finite and the learning rate α decaying over time.

C.3 Deep Q Learning

Deep Q-learning is a variant of the Q-learning algorithm that uses a deep neural network to approximate the action-value function $Q(s, a)$ in a Markov Decision Process (MDP). The algorithm combines reinforcement learning with deep neural networks to enable learning in high-dimensional and continuous state spaces.

The Deep Q-learning algorithm can be summarized as follows:

Algorithm: Deep Q-learning

1. Initialize the replay memory buffer D with capacity N.
2. Initialize the Q-network with random weights θ.
3. Initialize the target Q-network with weights $\theta^- = \theta$.
4. For each episode $e = 1, 2, ..., E$ do the following:
   (a) Initialize the environment with initial state s_0.
   (b) For each step $t = 1, 2, ..., T$ do the following:
      i. With probability ϵ select a random action a_t, otherwise select $a_t = \arg\max_a Q(s_t, a; \theta)$.
      ii. Execute action a_t and observe reward r_t and next state s_{t+1}.
      iii. Store the transition (s_t, a_t, r_t, s_{t+1}) in the replay memory buffer D.
      iv. Sample a mini-batch of transitions (s_j, a_j, r_j, s_{j+1}) from the replay memory buffer D.
      v. Compute the Q-learning target for each transition (s_j, a_j, r_j, s_{j+1}):
      $$y_j = \begin{cases} r_j & \text{if episode terminates at step } j+1 \\ r_j + \gamma \max_{a'} Q(s_{j+1}, a'; \theta^-) & \text{otherwise} \end{cases}.$$
      vi. Compute the loss between the predicted Q-value and the target Q-value: $L(\theta) = \frac{1}{B} \sum_{j=1}^{B} (y_j - Q(s_j, a_j; \theta))^2$.

vii. Update the Q-network weights using stochastic gradient descent: $\theta \leftarrow \theta - \alpha \nabla_\theta L(\theta)$.

viii. Every C steps update the target Q-network weights: $\theta^- \leftarrow \tau\theta + (1-\tau)\theta^-$.

ix. Set $s_t = s_{t+1}$.

In the Deep Q-learning algorithm, the replay memory buffer D is used to store experiences in order to prevent overfitting and stabilize learning. The target Q-network is used to compute the target Q-value in the Q-learning update, and its weights are periodically updated from the Q-network to prevent target overestimation.

The Deep Q-learning algorithm has been successfully applied to various tasks, such as playing Atari games, controlling robots, and playing board games.

The Deep Q-learning algorithm uses experience replay and target networks to improve stability and convergence of the algorithm. Experience replay randomly samples transitions from the replay memory buffer to decorrelate the data and prevent overfitting. Target networks are used to stabilize the training by keeping a separate target network with fixed parameters and periodically updating it with the weights of the online network.

The Deep Q-learning algorithm has been successfully applied to various tasks, such as playing Atari games, controlling robots, and playing board games.