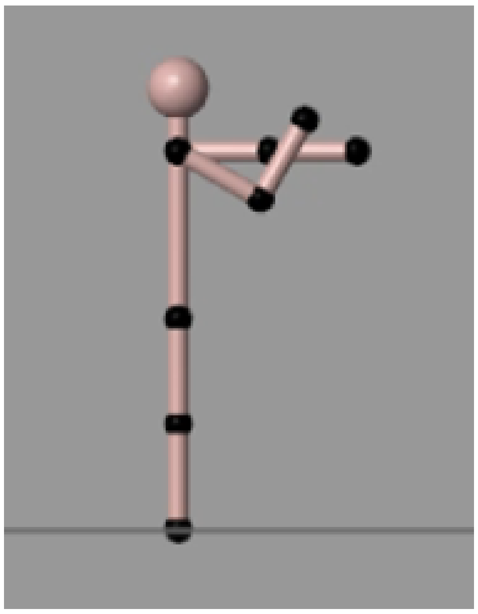
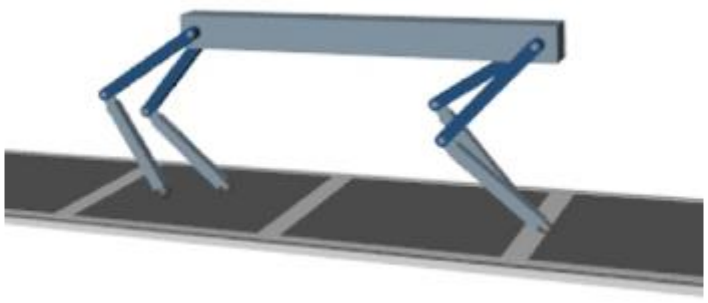
SIMULATION PRACTICE

Project Report

by

Martin Geiger





Submission: 30.08.2021

Group member: Robin Neubauer

Lecture: M23 Biologische Kybernetik - Biorobotik

Central aspects of robotic movement generation include both mechanical aspects and control software. Simulation of multibody models provides several benefits in this regard by allowing to test design concepts before construction and by facilitating decision making. This allows for early error detection and thus a reduction in development time and cost. Furthermore, multibody simulation enables testing of robots in a safe environment. The goal here was to apply different control concepts in order to generate robotic movement, including classic, bioinspired, and self-learning control.

1. Multibody Simulation and ideal torque control concept

Classical control concepts mainly use torque actuators for movement generation. The first task was to create a multibody model in Simulink in a specified environment. Head, torso, hip, and two legs consist of solid blocks, and hip and knee joints were implemented as revolute joints (Figure 1). The hip is rigidly connected to the world coordinate system and rigid transform blocks were used for rotation and translation of all other segments, all having a segment-specific coordinate system, to build the model. Afterwards, to generate joint torques and thus movement of the model, PID controllers were implemented for each of the four joints. Since PID control requires a desired trajectory, these were implemented and tuned via PID-tuner (Figure 2). Scope blocks were used to visualize differences between the current and desired state. This error is essential for movement because a PID controller tries to minimize the difference between the actual and the desired state of the system, which should be as small as possible however (Figure 3). Since the system output is fed back to the PID controller and its control signal changes depending on feedback, this is a closed loop control system.

2. Muscle implementation and bioinspired control concept

Biological actuators are muscles, acting as agonist and antagonist. Here we used bioinspired control concepts to generate movement of multibody models in Simulink. Muscles were implemented as Hill-type muscle models (Figure 4), which is a widely studied and applied muscle model in biomechanics. The first multibody model for this task consisted of head, torso, shoulder, and an arm with shoulder and elbow joint. The model holds a dumbbell in its hand (Figure 5), which should be moved up and down by implementing two Hill-type muscles, forming an antagonistic system, simulating biological actuation by a continuous change of stimulation. As a second task, we created a multibody model consisting of head, torso, shoulder, two arms with shoulder and elbow joints, and two legs with hip and knee joints. Two agonists and two antagonists were implemented in such a way, that they actuate a shoulder and an elbow joint by intermittent control. With this type of control movement is generated by switching between stimulation patterns where the system is at rest, because actuator forces cancel each other out, called equilibrium points (EPs). This was achieved by implementation of an EP controller. Our model simulates the movement of an archer, reaching to its back to grab an arrow from the quiver (EP1), then mounting it to the bowstring (EP2), and finally stretching the bowstring (EP3), before shooting the arrow (Figure 6). The main advantage of bioinspired control concepts is the ability to avoid ethical limitations of biological testing.

3. Self-learning control of robots with artificial intelligence

Reinforcement learning is a type of machine learning where the goal of an agent is to modify its policy as it interacts with the environment, so that eventually, given any state, it will always take the most advantageous action, the one that will produce the most reward in the long run. A more detailed description for the reward functions is provided on page 7 and for reinforcement learning on page 8. We first trained a quadruped robot, i.e. a DDPG-agent, to walk, by using a reward function (Figure 7+8, Equation 1). The agent generates eight actions, which are control signals, for its environment, the eight joints of the model. We then tripled the forward velocity reward and halved the penalty for keeping the trunk high (Equation 2). This resulted in a worse movement than before due to a higher reward and thus, as expected, an earlier termination of the training. Lastly, we trained the robot to walk backwards by negating the forward velocity reward (Figure 9, Equation 3). The main advantage of reinforcement learning is that it enables a robot to react to variable environments. Disadvantages include high computing times and difficulty to understand the robot’s behavior, which is why improvements are way harder to implement than with other control concepts.

REVIEW QUESTIONS

1. What is an equilibrium point in intermittent control?

2. What is the goal of a reinforcement learning agent?

FIGURES

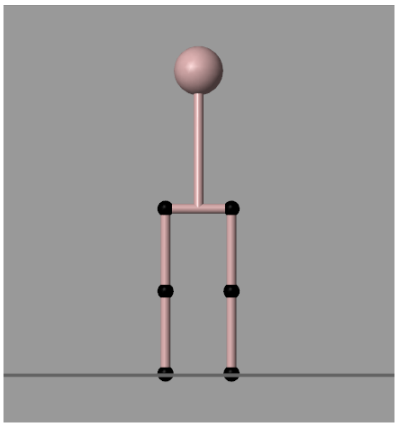
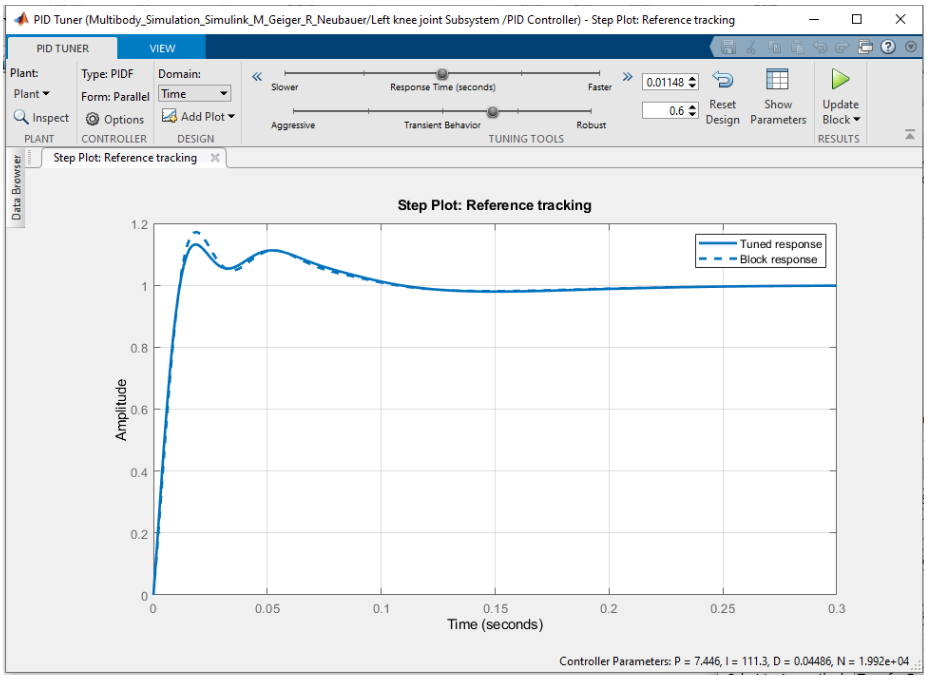


Figure 1: Multibody model created in Simulink, consisting of head, torso, hip, two legs, as well as hip and knee joints.



PID Tuner (Multibody_Simulation_Simulink_M_Geiger_R_Neubauer/Left knee joint Subsystem /PID Controller) - Step Plot: Reference tracking

Type: PIDF

Form: Parallel

Domain: Time

Response Time (seconds): 0.01148

Transient Behavior: 0.6

Controller Parameters: P = 7.446, I = 111.3, D = 0.04486, N = 1.992e+04

Figure 2: PID tuner in Simulink, applied to tune the implemented PID controllers to generate movement of the multibody model.

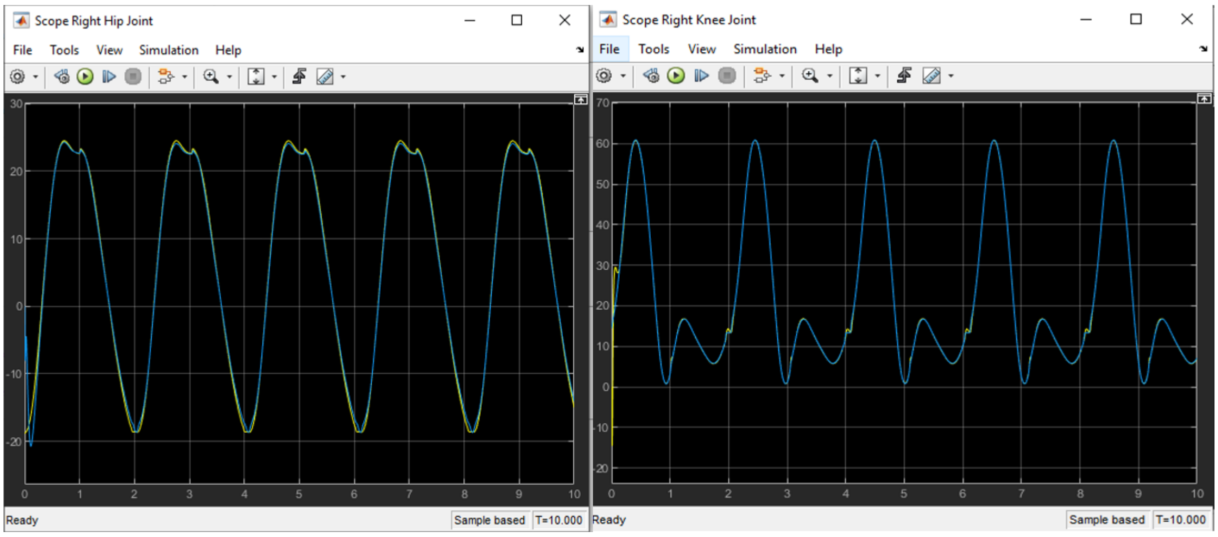


Figure 3: Scopes of hip (left) and knee (right) joint trajectories in Simulink to visualize differences between the current and desired state.

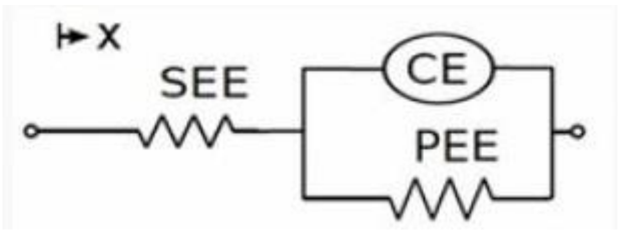


Figure 4: Mechanical Hill-type model of the muscle-tendon unit. The contractile element (CE) represents the muscle, the parallel elastic component (PEE) represents connective tissues, and the serial elastic component (SEE) represents the tendon (http://wiki.ifs-tud.de/biomechanik/muskel/mus02, last accessed: 30.08.2021, 11:52 a.m.).

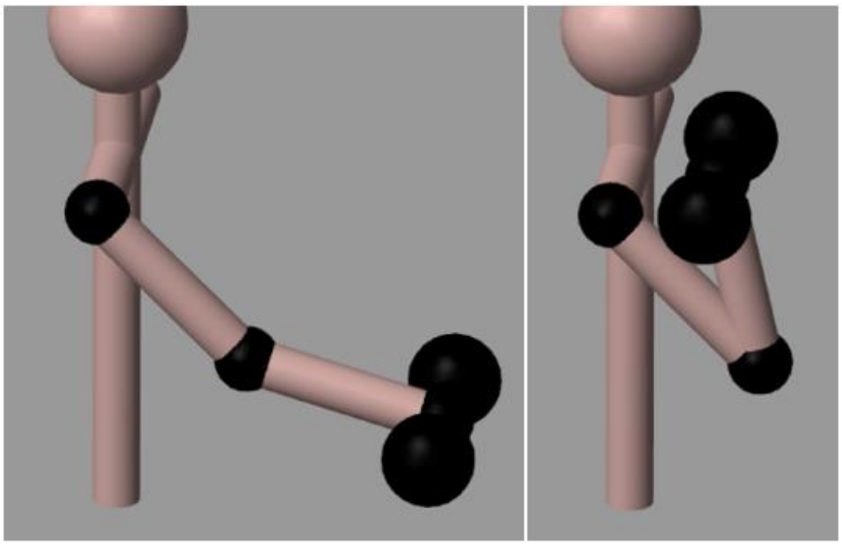


Figure 5: Multibody model created in Simulink, consisting of head, torso, shoulder, and an arm with shoulder and elbow joint. Two Hill-type muscle models were implemented in an antagonistic setup in order to move the dumbbell up and down.

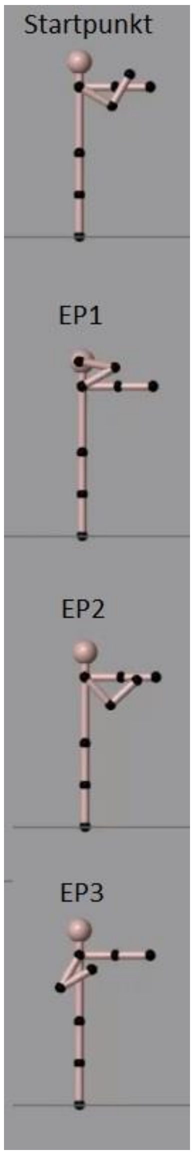


Figure 6: Multibody model created in Simulink, consisting of head, torso, shoulder, two arms with shoulder and elbow joints, and two legs with hip and knee joints. Movement was achieved by implementation of an EP controller. The model simulates the movement of an archer, reaching to its back to grab an arrow from the quiver (EP1), then mounting it to the bowstring (EP2) and finally stretching the bowstring (EP3) before shooting the arrow.

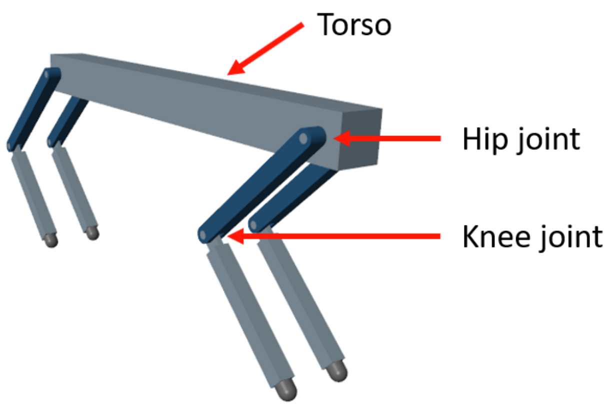
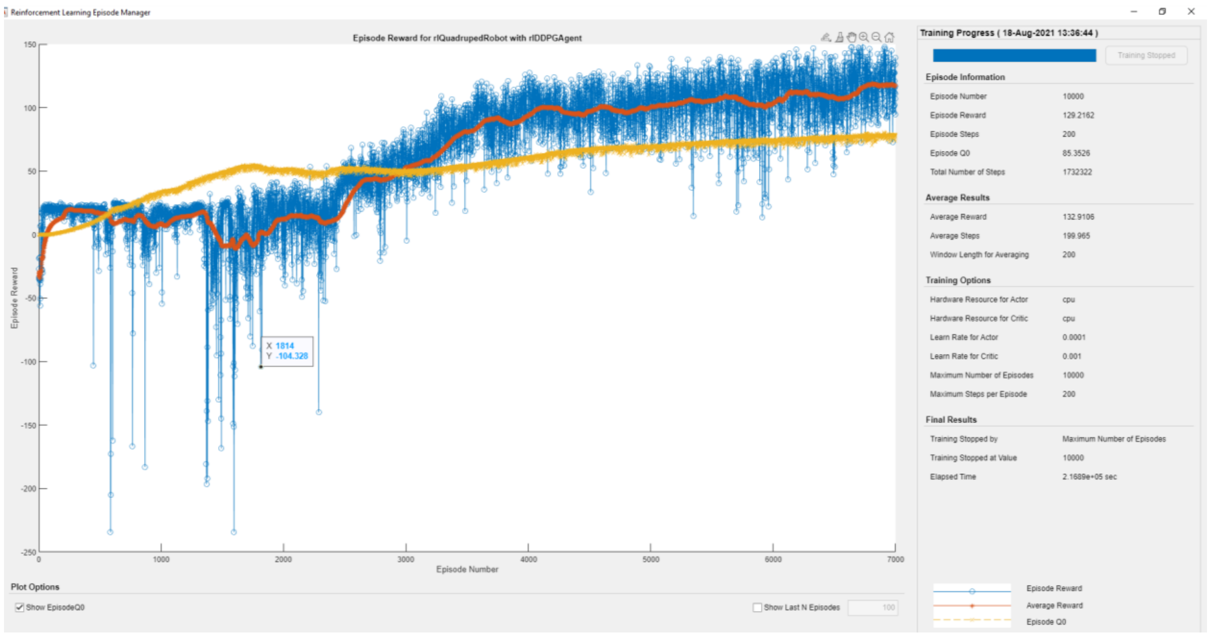


Figure 7: Quadruped robot model used for reinforcement learning, consisting of a torso and four legs with hip and knee joints. This model is the agent's environment and it generates eight actions for movement of the robot, which are control signal for the eight joints of the model.



Training Progress (18-Aug-2021 13:36:44)

Episode Information

Episode Number	10000
Episode Reward	129.2162
Episode Steps	200
Episode Q0	85.3526
Total Number of Steps	1732322

Average Results

Average Reward	132.9106
Average Steps	199.965
Window Length for Averaging	200

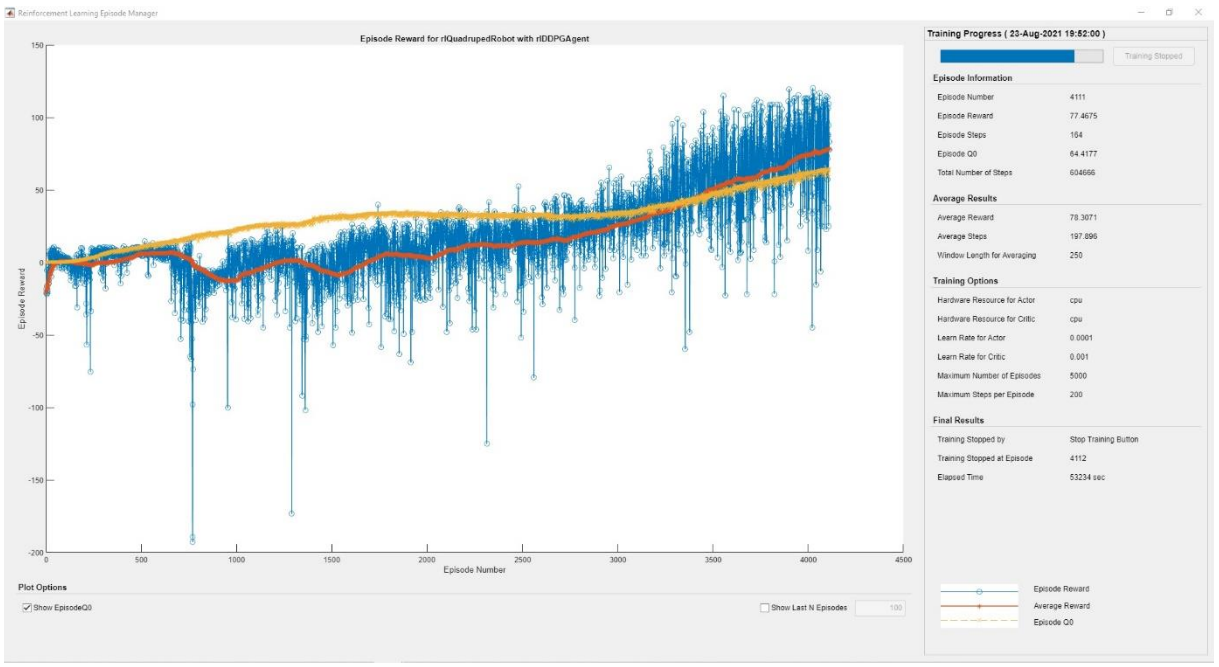
Training Options

Hardware Resource for Actor	cpu
Hardware Resource for Critic	cpu
Learn Rate for Actor	0.0001
Learn Rate for Critic	0.001
Maximum Number of Episodes	10000
Maximum Steps per Episode	200

Final Results

Training Stopped by	Maximum Number of Episodes
Training Stopped at Value	10000
Elapsed Time	2.1689e+05 sec

Figure 8: Training history for forward walking. Maximal episode number of 10.000 was reached before the average reward reached 200, what was required for saving an agent. Thus, no agent was saved. This problem could be solved by reducing the ‘SaveAgentValue’.



Training Progress (23-Aug-2021 19:52:00)

Episode Information

Episode Number	4111
Episode Reward	77.4675
Episode Steps	164
Episode Q0	64.4177
Total Number of Steps	604666

Average Results

Average Reward	78.3071
Average Steps	197.896
Window Length for Averaging	250

Training Options

Hardware Resource for Actor	cpu
Hardware Resource for Critic	cpu
Learn Rate for Actor	0.0001
Learn Rate for Critic	0.001
Maximum Number of Episodes	5000
Maximum Steps per Episode	200

Final Results

Training Stopped by	Stop Training Button
Training Stopped at Episode	4112
Elapsed Time	53234 sec

Figure 9: Training history for backward walking. The robot was trained for 4.111 episodes. Average reward was 78.3.

EQUATIONS

$$r_t = v_x + 25\frac{T_s}{T_f} - 50\hat{y}^2 - 20\theta^2 - 0.02\sum_i {u_{t-1}^i}^2$$

Equation 1: Reward function for forward walking.

$$r_t = 3v_x + 25\frac{T_s}{T_f} - 25\hat{y}^2 - 20\theta^2 - 0.02\sum_i {u_{t-1}^i}^2$$

Equation 2: Reward function for forward walking with tripled reward for forward velocity and halved penalty for keeping the trunk high.

$$r_t = -v_x + 25\frac{T_s}{T_f} - 50\hat{y}^2 - 20\theta^2 - 0.02\sum_i {u_{t-1}^i}^2$$

Equation 3: Reward function for backward walking with negated forward velocity reward.

EXPLANATION OF THE REWARD FUNCTION [1,3]

v_x is the velocity of the torso's center of mass in the x-direction.

→ forward velocity

T_s and T_f are the sample time and final simulation time of the environment, respectively.

→ walk as long as possible

$\hat{y}$ is the scaled height error of the torso's center of mass from the desired height of 0.75 m.

→ keep trunk high

θ is the pitch angle of the torso.

→ keep trunk horizontal

u_{t-1}^i is the action value for joint i from the previous time step.

→ minimize actuator effort

Reinforcement Learning

Reinforcement learning is a type of machine learning and provides a powerful paradigm for artificial intelligence and the enabling of autonomous systems to learn to make good decisions. A software, the agent, takes an action which affects the environment, changing its state, and the environment then produces a positive or negative reward for that action. The goal of the overall agent is to modify its policy as it interacts with the environment, so that eventually, given any state, it will always take the most advantageous action, the one that will produce the most reward in the long run. [1]

A DDPG agent is an actor-critic reinforcement learning agent that searches for an optimal policy that maximizes the expected cumulative long-term reward [2]. So, both the actor and the critic are neural networks. A neural network is a group of nodes, or artificial neurons, that are connected in a way that allows them to be a universal function approximator. We want to find a function that can take in a large number of observations and transform them into a set of actions to learn the optimal behavior [1]. The actor is a deterministic policy actor π(S), that takes observation S and returns the corresponding action that maximizes the long-term reward. The critic is a Q-value function critic Q(S,A), that takes observation S and action A as inputs and returns the corresponding expectation of the long-term reward. [2]

The environment is a quadruped robot, and the training goal is to make the robot walk in a straight line using minimal control effort. The main structural components are four legs and a torso. The legs are connected to the torso through revolute joints (Figure X). The agent generates eight actions normalized between –1 and 1. After multiplying with a scaling factor, these correspond to the eight joint torque signals for the revolute joints. [3]

A reward is provided to the agent at each time step during training. The reward function encourages the agent to move and to avoid early termination by providing a constant reward. The reward function also includes penalties that discourage unwanted states, such as large deviations from the desired torso height and orientation or the use of excessive joint torques (Equation 1). [3]

In this way, we can essentially design a controller, which is a policy, without knowing anything about the system itself. And without having to solve any of the traditional control problems, we just let the computer learn the right parameters on its own through experience, or trial-and-error, to parameterize a neural network. There is no labeled data. [1]

Sources

[1] https://de.mathworks.com/videos/series/reinforcement-learning.html, last accessed: 27.08.2021, 2:43 p.m.

[2] https://de.mathworks.com/help/reinforcement-learning/ug/ddpg-agents.html, last accessed: 27.08.2021, 2:58 p.m.

[3] https://de.mathworks.com/help/reinforcement-learning/ug/quadruped-robot-locomotion-using-ddpg-gent.html, last accessed: 27.08.2021, 3:32 p.m.