DLR-IB-RM-OP-2023-70

Tele Running – Energy Efficient Locomotion for Elastic Joint Robots by Imitation Learning

Master's thesis

David Wandinger

Deutsches Zentrum
DLR für Luft- und Raumfahrt

ERKLÄRUNG

gemäß §15 Abs. 5 APO in Zusammenhang mit §35 Abs. 7 Rapo

Name: Wandinger
Vorname: David
Geburtsdatum: 07.10.1997
Studiengang: Mechatronik / Feinwerktechnik
Studiengruppe: MFM
Matrikel-Nr.: 48556416
Sommer / Wintersemester: WS22 / 23
Betreuer (HM): Prof. Dr. Alexander Steinkogler
Betreuer (DLR): Dipl.-Ing. Manuel Keppler

Hiermit erkläre ich, das ich die Masterarbeit selbständig verfasst, noch nicht anderweitig für Prüfungszwecke vorgelegt, keine anderen als die angegeben Quellen oder Hilfsmittel benutzt, sowie wörtliche und sinngemäße Zitate als solche gekennzeichnet habe.

Ort, Datum

Unterschrift

Abstract

This thesis presents an imitation learning approach to energy-efficient trajectory generation for elastic, legged robots. The trajectories are generated by teleoperation with force feedback. The presented framework allows an operator to achieve locomotion on an one-leg hopper by controlling its foot tip. The force feedback is designed to assist the operator to find gaits which exploit the natural harmonics of the hopper and thus improve energy efficiency.
The resulting trajectory is approximated, parameterized, and replayed on the robot. The operator achieves a cost of transport of 0.25 at $0.63\,\mathrm{m\,s^{-1}}$, considering the mechanical energy. Black-box optimization is used to keep this value with varying hardware parameters, such as different foot-tip stiffness. A reinforcement learning algorithm stabilizes lateral movement by active balance in simulation. Learning on hardware shows an improvement in stability.
The concept is extended to multi-legged robots by teleoperating the two feet of the biped DLR C-Runner in simulation. The force feedback assists the operator to find stable gaits where the center of mass does not leave the support polygon of the feet.
On both systems, the presented teleoperation framework utilizes the human's capability of estimating the properties of non-linear dynamics by designing appropriate haptic feedback.

Keywords: Legged Robots; Cost of Transport; Black-Box Optimization; Reinforcement Learning; Humanoids

Contents

List of Figures

List of Tables

List of Symbols and Abbreviations

Throughout this thesis, vectors are written in bold lower case letters while matrices are bold capitals. The index of a frame is describing the corresponding axis in this frame. Trajectories are described explicitly as functions of time, while this argument is dropped for other time-dependent signals.

Abbreviations

BBO	Black-Box Optimization
SAC	Soft Actor-Critic
CoT	Cost of Transport
CoM	Center of Mass
DCM	Divergent Component of Motion
DDPG	Deep Deterministic Policy Gradient
DLR	Deutsches Zentrum für Luft- und Raumfahrt (German Aerospace Center)
DoF	Degrees of Freedom
DMPs	Dynamic Motor Primitives
ESπ	Elastic Structure Preserving Impedance Control
IMU	Inertial Measurement Unit
MSBE	Mean Squared Bellman Error
SEA	Series Elastic Actuator
SLIP	Spring-Loaded Inverted Pendulum
TCP	Tool Center Point, foot tip of the hopper

TD3	Twin Delayed Deep Deterministic Policy Gradient
TPE	Tree-structured Parzen Estimator

Signals

$\boldsymbol{f}_{ext,\mathcal{F}}$	Vector of Cartesian external forces in the foot tip frame
$\boldsymbol{f}_{\Omega}$	Vector of Cartesian forces for force feedback
$\boldsymbol{g}$	Gravity torque vector
$\boldsymbol{q}_a$	Motor coordinate vector
$\boldsymbol{q}_u$	Link coordinate vector
$\boldsymbol{q}_{d,a}(t)$	Desired motor-side trajectory
$\boldsymbol{q}_{d,u}(t)$	Desired link-side trajectory
$\check{\boldsymbol{q}}_u$	Link coordinate vector in serial coordinates, for the hopper
$\boldsymbol{\mathcal{Q}}_a$	External motor-side torque vector, motor friction
$\boldsymbol{\mathcal{Q}}_u$	External link-side torque vector
$\boldsymbol{\tau}_j$	Spring torque vector
$\texttt{traj}_0(t)$	Rhythmic trajectory function, generated by demonstration
$\boldsymbol{u}$	Control torque input vector
$\boldsymbol{x}_b$	Base position vector relative to the world frame (unless other frame specified)
$\boldsymbol{x}_d$	Desired foot position vector
$\boldsymbol{x}_{d,o}$	Desired foot position vector, given by the operator
$\boldsymbol{x}_{d,sat}$	Desired foot position vector, saturated to the workspace
$\boldsymbol{x}_f$	Foot position relative to the base frame (unless other frame specified)

Matrices

$\boldsymbol{B}$	Diagonal motor inertia matrix
$\boldsymbol{C}$	Centrifugal and Coriolis matrix
$\boldsymbol{J}_{\mathcal{F}}$	Link-side Jacobian in foot tip coordinates
$\boldsymbol{K}$	Diagonal spring stiffness matrix
$\boldsymbol{M}$	Link inertia matrix
$\check{\boldsymbol{T}}$	Transformation matrix from coupled to serial coordinates

Frames

$\mathcal{B}$	Base frame
$\mathcal{C}$	CoM frame
$\mathcal{F}$	TCP, foot tip frame
Ω	Omega.3 and Falcon frame
$\mathcal{W}$	World frame

Optimization Parameters

CoT_{v0}	Cost function of CoT with a weight on the desired forward velocity
A_1	Gain on the trajectory for joint 1
A_2	Gain on the trajectory for joint 2
d_1	Offset of the trajectory for joint 1
d_2	Offset of the trajectory for joint 2
ω	Base frequency of a rhythmic trajectory
φ	Phase between the two joints of a rhythmic trajectory
Θ	Set of optimization parameters

Reinforcement Learning

a_t	Action at timestep t
$\mathcal{D}$	Replay buffer
$\mathop{\mathbb{E}}_{x \sim p}$	Expectation, under the condition that the random variable x is selected from distribution p
μ	Deterministic policy
o_t	Observation at timestep t
P	State-action transition matrix
π	Stochastic policy
Q	On-policy action-value function
r_t	Reward at timestep t
s_t	State at timestep t
V	On-policy value function

Other Parameters and Symbols

d	Traversed distance of the robot
E	Consumed energy of the robot
g	Gravity constant
γ	Rotation angle of the robot's base relative to the world frame
m	Mass of the robot
t	Time vector
T	Period of a rhythmic trajectory
v	Forward velocity of the robot

1 Introduction

Mobile robots fulfill more and more important tasks in our society. Rovers are used as a crucial tool in search and rescue missions. Others explore unknown and for humans yet unreachable environments like the surface of Mars. Wheeled locomotion is amongst the highest in efficiency, however accessibility is limited. Legged robots on the other hand are superior in overcoming obstacles. The state-of-the-art quadruped Spot of Boston Dynamics is used for search and rescue in fires and other hazardous environments with rough terrain.

The design and control of legged robots is a topic of active research. Raibert (1986) achieved a breakthrough by designing robotic hoppers that run with dynamic balance [1]. These mono-, bi-, and quadrupeds were designed by the MIT Leg Laboratory and use elastic linear actuators. They balance by hopping and continuously adjusting the foot position relative to their Center of Mass (CoM) when they are in the air. Other hoppers of that type are the ARL Monopod I and II [2, 3]. While these robots use prismatic joints, the SPEAR robot uses revolute joints with parallel elastic actuators [4].

Elastic robots have useful properties for legged locomotion. They are robust against impacts and the elastic elements can store and release energy. In robots with Series Elastic Actuators (SEAs) the mechanical spring is located between the motor and the link of each joint. State-of-the-art legged robots with SEAs are the biped C-Runner [5] and the quadruped bert [6] of the German Aerospace Center (DLR). State-of-the art trajectory generation for multi-legged locomotion relies on complex computations and CoM trajectories like the Divergent Component of Motion (DCM) framework of Englsberger et al. (2015) [7].

These systems, as well as their inherent harmonics, are nonlinear. Exiting them can be used for energy efficient locomotion. However, these non-linear oscillations

are mathematically hard to find, as the established modal analysis methods only apply in the linear case. They depend heavily on hardware parameters, external forces and friction. Exciting them by a model-based control approach and the transfer from simulation to hardware is still a topic of active research [8, 9, 10, 11, 12]. Model-free learning approaches have shown good results on these systems. Raffin et al. (2022) combined Black-Box Optimization (BBO) and reinforcement learning to stabilize and improve the locomotion of the DLR quadruped bert [6] while drawing on the elastic properties [13].

Apart from machine learning algorithms, humans are able to excite non-linear oscillations easily, e.g. exciting the natural oscillation of a long ruler blindfolded [14]. By feeling the reaction forces of systems, one gets an intuition of the underlying dynamical properties [15].

The basic research idea of this thesis is combining the human intuition for oscillatory dynamics with learning algorithms to find trajectories for energy efficient locomotion of robots with serial elastic joints. This method is known as imitation learning. Energy efficiency will be measured by the Cost of Transport (CoT), which is relating the (mechanical) energy consumption and traveled distance of the robot. The MIT Cheetah is amongst the most energy efficient legged robots. It achieved a CoT of 0.13 [16].

The main part of this thesis is researched on a SEA one-leg testbed (cf. Figure 1.1). It is part of the unpublished next generation of the DLR bert [6]. Throughout Chapter 3, the imitation learning process is shown. The haptic device omega.3 of the company force dimension® (cf. Figure 1.2) is used to control the hopper with teleoperation (as depicted in Figure 1.3). High precision and low latency force feedback is used to enable the operator to find gaits which match the natural frequency. The recorded data is parameterized and used as a baseline trajectory for learning algorithms to improve energy efficiency and stability. The operator achieved a CoT of 0.25 at $0.63\,\mathrm{m\,s^{-1}}$. The optimization showed only slim improvement. During teleoperation and optimization, the robot is mechanically prevented from tipping over. In order to stabilize the hopper without this constraint, a controller is designed by reinforcement learning. It is successfully trained in simulation and on hardware. Chapter 4 expands the concept to bipedal locomotion. By using two input devices, a human operator commands a walking gait of the compliant biped DLR C-runner [5]. This is done in a real-time simulation.

In conclusion, the main contributions of this work are designing a framework to intuitively teleoperate the one-leg with appropriate force feedback, and achieve locomotion. The generated trajectories are optimized for energy efficiency. The locomotion is stabilized by a controller, designed by reinforcement learning. Additionally, the biped C-Runner is teleoperated in simulation by using two haptic input devices.

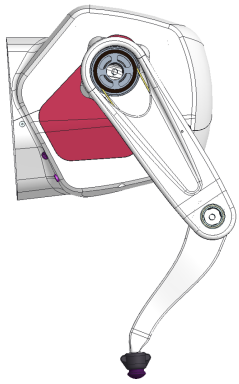


Figure 1.1: One-leg hopper with serial elastic joints

Figure 1.2: The omega.3 haptic input device

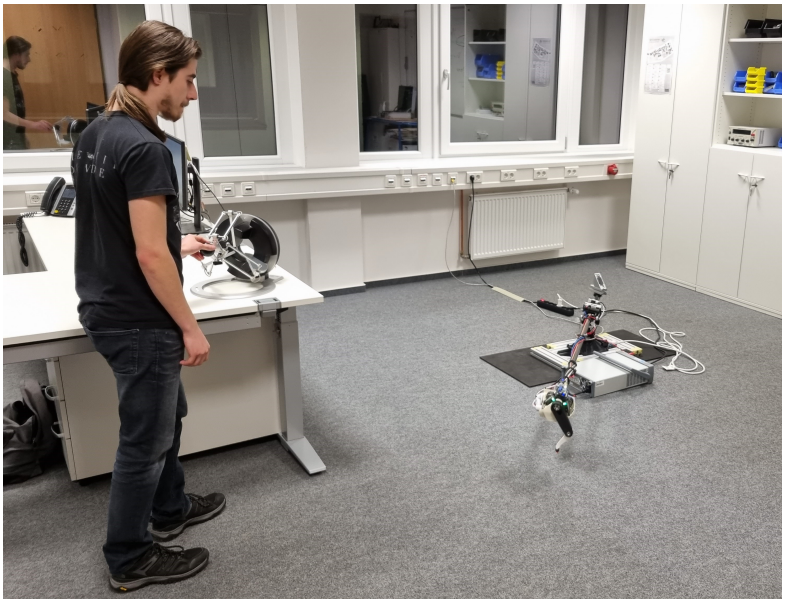


Figure 1.3: An operator controlling the elastic one-leg hopper by teleoperation

2 Theoretical Background

This chapter explains the fundamentals of energy efficient locomotion with elastic robots and the imitation learning method.

2.1 Evaluation of Energy Efficiency

Energy efficiency will be evaluated by the Cost of Transport (CoT), which is commonly used both in biology and in robotic locomotion. It is defined by the consumed energy E of the robot over its traversed distance d, with the robot's mass m and gravity constant g taken into account:

$$CoT = \frac{E}{m\,g\,d}\ . \tag{2.1}$$

Gregorio et al. (1997) achieved a CoT of 0.7 at $1.2\,\mathrm{m\,s^{-1}}$ with a electrically actuated hopper, the ARL Monopod I [2]. The CoT was improved to 0.22 with the ARL Monopod II [3]. They used a modified control approach of the simple but effective algorithm by Raibert (1986) [1]. The SPEAR robot is using parallel elastic actuators to achieve a CoT of 0.45 at $0.54\,\mathrm{m\,s^{-1}}$ [4]. The state-of-the-art quadruped MIT Cheetah has a CoT of 0.13 during running with $2.3\,\mathrm{m\,s^{-1}}$ [16]. These robots are depicted in Figures 2.2 and 2.3.

Furthermore, the CoT of each system depends on the systems forward speed. Mobile systems have a region of speed where their locomotion is most efficient. This is represented by the Kármán-Gabrielli diagram (cf. Figure 2.1). It was introduced by Gabrielli and von Kármán (1950) to compare the CoT over speed for different systems [17]. Only the mechanical energy of the motors is considered, i.e. the electrical energy and heat loss is neglected. This makes the different systems comparable over the different actuators and overall efficiency.

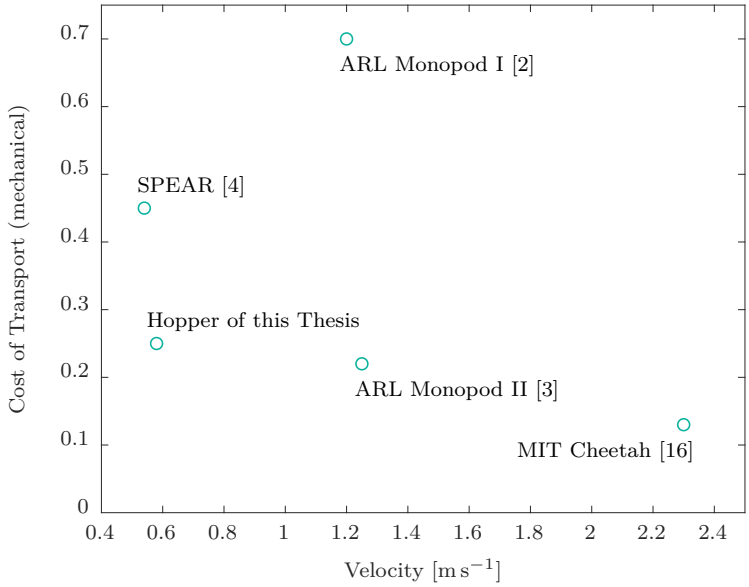
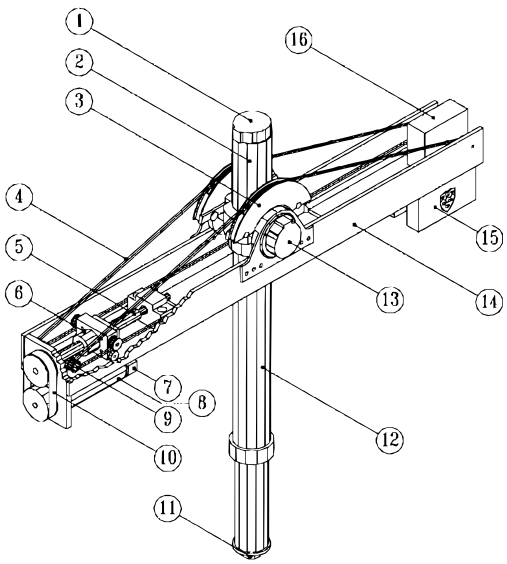


Figure 2.1: Kármán-Gabrielli diagram for state-of the-art elastic legged robots



1: leg motor sensor
2: leg motor
3: hip pulley
4: cables
5: ball screw
6: ball nut
7: hip motor sensor
8: hop motor
9: cable idler
10: limiting belt
11: rubber toe
12: leg
13: hip sensor
14: body
15: servo amplifiers
16: counter weight

Figure 2.2: The ARL Monopod I. Adapted from [2].

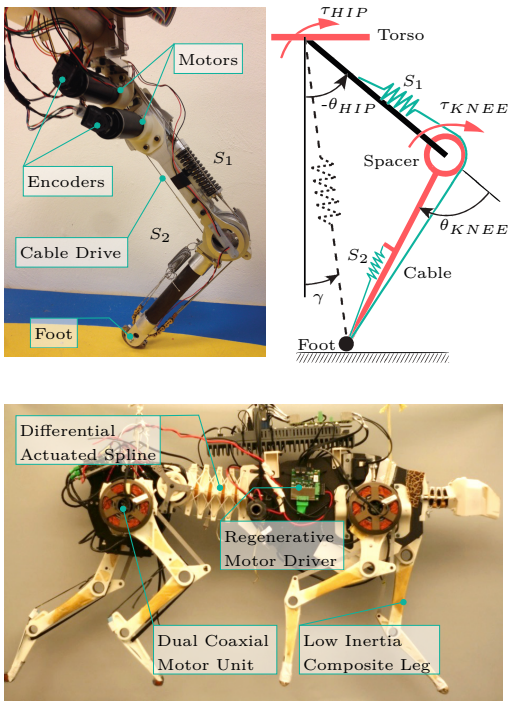


Figure 2.3: The SPEAR robot (top) and the MIT Cheetah robot (bottom). Adapted from [16, 18].

2.2 Serial Elastic Joint Robots

For robots with Series Elastic Actuators (SEAs), the motors and links are decoupled by a mechanical spring. Figure 2.4 depicts the principle sketch of a single SEA, with b and m as motor and link inertias, and q_a and q_u as motor and link positions respectively. Further, k denotes the spring stiffness. The inputs u and $\mathcal{Q}_u$ represent the control input and external forces exerted on the link.

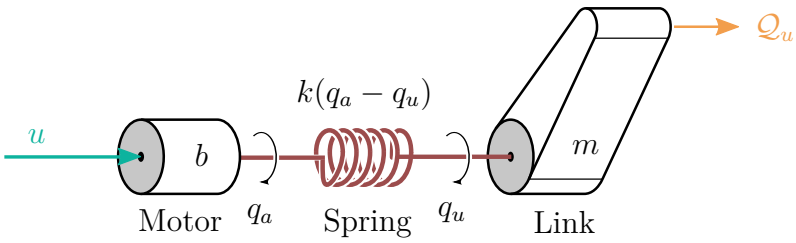


Figure 2.4: Principle sketch of a single series elastic joint

The equations of motion for articulated robots with multiple joints can be derived by the model proposed by Spong (1987) [19]:

$$\boldsymbol{M}(\boldsymbol{q}_u)\ddot{\boldsymbol{q}}_u + \boldsymbol{C}(\dot{\boldsymbol{q}}_u, \boldsymbol{q}_u)\dot{\boldsymbol{q}}_u + \boldsymbol{g}(\boldsymbol{q}_u) = \boldsymbol{\tau}_j + \boldsymbol{\mathcal{Q}}_u \ , \tag{2.2}$$

$$\boldsymbol{B}\ddot{\boldsymbol{q}}_a + \boldsymbol{\tau}_j = \boldsymbol{u} + \boldsymbol{\mathcal{Q}}_a \ . \tag{2.3}$$

The first equation of the system (2.2) reflects the properties of the rigid body dynamics of the n-link robot, with $\boldsymbol{M}(\boldsymbol{q}_u) \in \mathbb{R}^{n\times n}$ denoting the link inertia matrix and $\boldsymbol{C}(\dot{\boldsymbol{q}}_u, \boldsymbol{q}_u) \in \mathbb{R}^{n\times n}$ the centrifugal and Coriolis matrix. The influence of gravity is represented by the vector $\boldsymbol{g}(\boldsymbol{q}_u) \in \mathbb{R}^n$. These are configuration dependent, i.e. functions of of the link coordinates vector $\boldsymbol{q}_u \in \mathbb{R}^n$ and $\dot{\boldsymbol{q}}_u$ for $\boldsymbol{C}$.[1] Link-side external torques are represented by the vector $\boldsymbol{\mathcal{Q}}_u \in \mathbb{R}^n$.

The Spong model neglects the gyroscopic forces of the motor inertias [19]. The second equation (2.3) describes the motor dynamics of the robot, with $\boldsymbol{q}_a \in \mathbb{R}^n$ denoting the vector of motor coordinates. The diagonal matrix $\boldsymbol{B} \in \mathbb{R}^{n\times n}$ contains the motor inertias and the vector $\boldsymbol{\mathcal{Q}}_a \in \mathbb{R}^n$ describes the friction torque acting on the motors. The control input $\boldsymbol{u} \in \mathbb{R}^n$ actuates the motors.

[1] For brevity, the arguments of these matrices and vectors will be omitted throughout the rest of this thesis.

The system of equations is coupled by the spring torques $\boldsymbol{\tau}_j \in \mathbb{R}^n$, defined by:

$$\boldsymbol{\tau}_j \;=\; \boldsymbol{K}(\boldsymbol{q}_a - \boldsymbol{q}_u)\ , \tag{2.4}$$

with $\boldsymbol{K} \in \mathbb{R}^{n\times n}$, denoting the diagonal spring stiffness matrix.

As mobile robots are free to move in the world in m Degrees of Freedom (DoF), the floating base model applies. The rigid body dynamics equation (2.2) is enhanced by the base position $\boldsymbol{x}_b \in \mathbb{R}^m$ and its derivatives relative to the world frame $\mathcal{W}$:

$$\underbrace{\begin{bmatrix} \boldsymbol{M}_b & \boldsymbol{M}_{bq} \\ \boldsymbol{M}_{bq}^T & \boldsymbol{M} \end{bmatrix}}_{\underline{\boldsymbol{M}}} \begin{pmatrix} \ddot{\boldsymbol{x}}_b \\ \ddot{\boldsymbol{q}}_u \end{pmatrix} + \underline{\boldsymbol{C}} \begin{pmatrix} \dot{\boldsymbol{x}}_b \\ \dot{\boldsymbol{q}}_u \end{pmatrix} + \underline{\boldsymbol{g}} \;=\; \begin{pmatrix} \boldsymbol{0} \\ \boldsymbol{\tau}_j \end{pmatrix} + \underline{\boldsymbol{Q}}_u\ . \tag{2.5}$$

The submatrix $\boldsymbol{M}_b \in \mathbb{R}^{m\times m}$ is the inertia matrix of the base. The submatrix $\boldsymbol{M}_{bq} \in \mathbb{R}^{m\times n}$ describes how link accelerations apply inertial torques and forces on the base. This notation is adopted from [20].

2.3 Imitation Learning

Imitation learning is the method of observing an action and trying to repeat it. In machine learning, a model is trained by observing and imitating behavior in a specific task, rather than learning from explicit rules or objectives. This is useful in robotics, because imitation can be used to omit manual programing of e.g. trajectories, which will be the focus of this work.

Kober et al. (2008) taught a robotic arm to solve the Ball-in-a-Cup or Kendama game. A ball is connected by a string to a cup. The goal is to propel the ball into the cup by swinging the cup. Trajectories from a human demonstration are used to initialize Dynamic Motor Primitives (DMPs), which were improved by reinforcement learning [21].

2.3.1 The Basics of Reinforcement Learning

Reinforcement learning is searching for an agent's optimal behavior in an environment. The underlying principle is depicted in Figure 2.5. The agent receives, with discrete steps, information about the state of the environment s_t and gets a scalar reward signal r_t. The state is fulfilling the Markov property, meaning the current state of the environment is containing all information about the environment. The agent's behavior is determined by the policy, which is a set of rules, that determines the agent's actions a based on s. The policy can be deterministic, in which case it will be called $\mu(s)$ or stochastic $\pi(s)$. The reward r_t is the reward given, due to action a_t. The action in the next timestep is called a'. The terminal state is sent by the environment, when the episode is finished.

Usually, the complete state of the environment is unknown. Then the agent receives an observation signal o_t with the known state of the environment. The principle of the algorithms applies both to given states and observations. In this part, only states are considered. In the implementation, observations are used. States, observations, and actions are vectors with the size of their respective space.

The following derivation of the learning algorithms is leaned on the introduction to reinforcement learning [22] by Sutton and Barto (2018) and the documentation of the python library and resource platform Spinning Up in Deep RL! [23] of OpenAI®. In this thesis, the algorithms are implemented with the stable-baselines3 python library [24]. It features pre-tuned hyperparameters and easy implementations for a variety of algorithms.

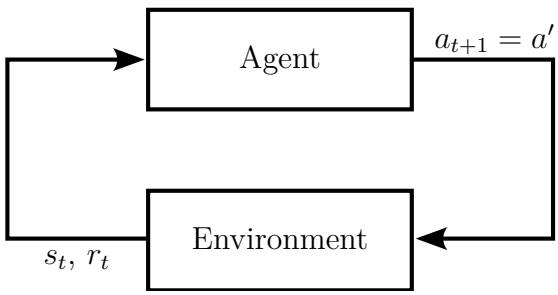


Figure 2.5: Agent-environment dynamic of reinforcement learning

The aim of reinforcement learning is maximizing the cumulative reward, called return R, which is defined by:

$$R(\tau) \;=\; \sum_{t=0}^{\infty} \gamma^t \, r_t \;, \tag{2.6}$$

where τ is a sequence of state-action pairs over time frame. Future returns are discounted by the factor γ. The quality or value of a policy is measured by its expected return. It depends on the interaction between policy and the dynamics of the environment:

$$J(\pi) \;=\; \int_{\tau} P(\tau|\pi)\, R(\tau) \;=\; \mathop{\mathbb{E}}_{\tau\sim\pi} [R(\tau)] \;. \tag{2.7}$$

The equation describes the expectation $\mathbb{E}$, that τ follows the policy π. The state-action transition matrix $P(\tau|\pi)$ is determining the probability of following the trajectory τ with the policy π, when the environment's dynamics are taken in account.[2] In short, the reinforcement learning problem is finding the best policy π^* by:

$$\pi^* \;=\; \mathop{\mathtt{argmax}}_{\pi} J(\pi) \;. \tag{2.8}$$

The covered algorithms rely on the Bellman optimality principle. Values are defined by their intermediate reward plus the expected cumulative reward in the future [25, Chapter 3.2]. The derived Bellman equations describe the value of policies and actions. A policy is evaluated by its on-policy value function:

$$V^{\pi}(s) \;=\; \mathop{\mathbb{E}}_{\substack{a\sim\pi \\ s'\sim P}} [r(s,a) + \gamma \, V^{\pi}(s')] \;=\; \mathop{\mathbb{E}}_{a\sim\pi} [Q^{\pi}(s,a)] \;. \tag{2.9}$$

It describes the expected return in state s, if the actions are determined by π and the next state s' is determined by P. Is is calculated by the immediate return r plus the discounted future rewards. Further, the on-policy action-value function describes the expected return if an arbitrary (off-policy) action is taken and thereafter the policy is followed:

$$Q^{\pi}(s,a) \;=\; \mathop{\mathbb{E}}_{s'\sim P} \Big[r(s,a) + \gamma \mathop{\mathbb{E}}_{a'\sim\pi} [Q^{\pi}(s',a')] \Big] \;. \tag{2.10}$$

[2] For brevity $P(\cdot|\pi)$ will be denoted by P.

The optimal value functions have the best expected return, they fulfill the Bellman optimality principle:

$$V^*(s) = \max_a \mathop{\mathbb{E}}_{s' \sim P} \left[r(s,a) + \gamma \, V^*(s') \right] = \max_a Q^*(s,a) \ , \tag{2.11}$$

$$Q^*(s,a) = \mathop{\mathbb{E}}_{s' \sim P} \left[r(s,a) + \gamma \max_{a'} Q^*(s',a') \right] \ . \tag{2.12}$$

The strategy of the covered learning algorithms is to explore the environment, create approximations of the value functions and optimize them to fulfil the Bellman optimality equations. The approximations are usually non-linear neural networks. One method is Q-Learning by Deep Deterministic Policy Gradient (DDPG) [26]. It is learning the on policy action-value function by approximating it with a neural network with the parameters ϕ. It is then minimizing the Mean Squared Bellman Error (MSBE) between the approximation Q_ϕ and a target $y(r, s', a')$. The target is satisfying the Bellman optimality principle in equation (2.12):[3]

$$L(\phi, \mathcal{D}) = \mathop{\mathbb{E}}_{(s,a,r,s') \sim \mathcal{D}} \left[\left(Q_\phi(s,a) - y(r,s',a') \right)^2 \right] \ , \tag{2.13}$$

$$y(r,s',a') = r + \gamma \max_{a'} Q_\phi(s',a') \ . \tag{2.14}$$

The data of a replay buffer is used, which is the set $\mathcal{D}$ of learned $\{s, a, r, s'\}$ pairs. If the terminal state is reached after s, no future reward is given. The corresponding policy is calculated by using the Bellman optimality principle in equation (2.11):

$$\max_\theta \mathop{\mathbb{E}}_{s \sim \mathcal{D}} \left[Q_\phi(s, \mu_\theta(s)) \right] \ . \tag{2.15}$$

This process is called policy optimization. The generated deterministic policy $\mu_\theta(s)$ is a neural net with parameters θ.

[3] In the implemented algorithm, the target's neural net parameters are determined by polyak averaging [27], to solve the optimization problem.

2.3.2 Reinforcement Learning using the Soft Actor-Critic Algorithm

The key feature of the Soft Actor-Critic (SAC) algorithm is that the tradeoff between exploring new policies and exploiting the return of the current policy is handled by rewarding entropy in the policy [28]. A policy with high entropy is more random. This enables on-policy exploration. It was used to demonstrate in-hand manipulation on the robot Justin of the German Aerospace Center (DLR). Using SAC, Justin was able to rotate a cube in its hand, while grasping it from the top [29]. No visual sensors were used. The SAC algorithm is able to stabilize highly sensitive systems and was therefore chosen for this thesis.

The entropy of a random variable x is calculated by the logarithmic likelihood log of its distribution:

$$H(P) \;=\; \mathop{\mathbb{E}}_{x\sim P}\left[-\log\left(P(x)\right)\right] \;. \tag{2.16}$$

Policies with a higher entropy are favoured, this is respected in the value functions:

$$\begin{aligned}
V^{\pi}(s) \;&=\; \mathop{\mathbb{E}}_{a\sim\pi}\left[Q^{\pi}(s,a)\right] + \alpha\, H(\pi(\cdot|s)) \;, && (2.17)\\
&=\; \mathop{\mathbb{E}}_{a\sim\pi}\left[Q^{\pi}(s,a) - \alpha\, \log(\pi(a|s))\right] \;, && (2.18)\\
Q^{\pi}(s,a) \;&=\; \mathop{\mathbb{E}}_{s'\sim P}\left[r(s,a) + \gamma \mathop{\mathbb{E}}_{a'\sim\pi}\left[Q^{\pi}(s',a')\right] + \alpha\,\gamma\, H(\pi(\cdot|s'))\right] \;, && (2.19)\\
&=\; \mathop{\mathbb{E}}_{s'\sim P}\left[r(s,a) + \gamma \mathop{\mathbb{E}}_{a'\sim\pi}\left[Q^{\pi}(s',a') - \alpha\, \log(\pi(a'|s'))\right]\right] \;. && (2.20)
\end{aligned}$$

The hyperparameter α determines how much entropy is favoured, and therefore effects the balance between exploration and exploitation.

Q-Learning often overestimates the action-values, which is exploited by the policy learning. This is counteracted by double Q-Learning, which was introduced by the Twin Delayed Deep Deterministic Policy Gradient (TD3) algorithm [30]. It is also used in SAC. Two action-value functions $Q_{\phi,i}$ are learned with the same target function. For each action, the target function takes the smaller action-value of both action-value functions:

$$L(\phi_1, \mathcal{D}) = \mathop{\mathbb{E}}_{(s,a,r,s')\sim\mathcal{D}} \left[\left(Q_{\phi,1}(s,a) - y(r,s')\right)^2 \right] , \tag{2.21}$$

$$L(\phi_2, \mathcal{D}) = \mathop{\mathbb{E}}_{(s,a,r,s')\sim\mathcal{D}} \left[\left(Q_{\phi,2}(s,a) - y(r,s')\right)^2 \right] , \tag{2.22}$$

$$y(r,s') = r + \gamma \min_{j=1,2} Q_{\phi,j}(s', \tilde{a}') \; - \alpha \log(\pi_\theta(\tilde{a}'|s')) \; . \tag{2.23}$$

The next actions in the target are sampled by the current stochastic policy: $\tilde{a}' \sim \pi_\theta(\cdot|s')$. The next state is determined by $\mathcal{D}$.

The policy is learned like in equation (2.15), but by using the minimum of both action-value functions and taking the entropy into account:

$$\max_\theta \mathop{\mathbb{E}}_{s\sim\mathcal{D}} \left[\min_{i=1,2} Q_{\phi,i}(s, \mu_\theta(s)) - \alpha \log(\pi_\theta(\mu_\theta(s)|s)) \right] \; . \tag{2.24}$$

The policy μ_θ is a deterministic approximation of the stochastic policy π_θ with added white noise.

2.3.3 Black Box Optimization with the The Tree-structured Parzen Estimator Algorithm

Black-Box Optimization (BBO) is an optimization method, closely related to machine learning. It is used, when the cost function cannot be directly accessed, but only evaluated by given inputs. A direct calculation of the gradient is not possible, therefore BBO algorithms often learn local models of the function and find their minima.

For this thesis, the Optuna framework in Python is chosen [31]. The algorithms in this framework are developed for hyper parameter optimization. Hyper parameters control the learning process in machine learning. They influence the performance and outcome of the algorithms. Typically, there are no functions that connect the hyperparameters to the performance metrics. Consequently, they are either tuned by hand or by BBO.

Depending on the problem, machine learning can take long computation time. Consequently, hyperparameter optimizers are developed, to find minima with few iterations. That makes them applicable for optimization problems in robotics, where hardware is used. Naturally the real robot can not perform faster than real time, in contrast to simulation, and needs attention and maintenance during

experiments. Thus, optimization methods on hardware are required to be as efficient as possible.

The algorithm of choice is the Tree-structured Parzen Estimator (TPE) [32]. It is a Bayesian optimization algorithm that approximates the cost function by local Gaussian distribution models. After each iteration, the TPE algorithm is creating two distribution models. The first model $l(x)$ describes the distribution of promising candidates, that resulted in a low cost function. The second model $g(x)$ is the distribution of the other candidates, that produced high cost. The parameter λ decides, which candidates are modeled in $l(x)$ or $g(x)$. Setting $\lambda = 0.5$ means, that from all previously evaluated candidates, the best 50 % are modeled in $l(x)$ and the rest in $g(x)$.

The candidate for the next iteration is the most promising candidate for minimizing cost of both distribution models, i.e. high probability in $l(x)$ and low probability in $g(x)$. It is found by:

$$x_{i+1} \;=\; \underset{x}{\texttt{argmin}} \frac{g(x)}{l(x)} \;. \tag{2.25}$$

The algorithm needs a distribution model to start. Therefore, it is bootstrapped by evaluating a number of random parameter sets at start.

3 Solution Approach on an Elastic One-Leg Hopper

This chapter applies the imitation learning method of Section 2.3 on an one-leg hopper. The framework for teleoperation is presented and the generated trajectories are discussed. A Black-Box Optimization (BBO) algorithm further optimizes the Cost of Transport (CoT) while reinforcement learning is used to stabilize lateral movement.

3.1 Kinematics of the Hopper

The kinematic structure of the hopper is depicted in Figure 3.1. The robot has two Degrees of Freedom (DoF) and the corresponding generalized link coordinates are called $q_{u,1}$ and $q_{u,2}$. The base frame $\mathcal{B}$ is attached to the floating base of the robot and the frame $\mathcal{F}$ to the Tool Center Point (TCP) or foot tip of the robot. It has the same orientation as $\mathcal{B}$, regardless of the configuration of the robot. The frame $\mathcal{C}$ is attached to the Center of Mass (CoM) of the robot, which is close to the base frame.

One distinctiveness of the robot is, that $q_{u,1}$ and $q_{u,2}$ are kinematically coupled. In serially articulated robots, each generalized coordinate rotates about an axis in the frame of its respective link. Conversely, with the coupled kinematic of the robot, both coordinates rotate about the y-axis of $\mathcal{B}$ ($\mathcal{B}_y$). This makes only a difference for $q_{u,2}$. Accordingly, if $q_{u,1}$ is actuated and $q_{u,2}$ is e.g. aligned parallel to $\mathcal{B}_z$, it will stay parallel to $\mathcal{B}_z$ unless it is actuated itself. The transformation matrix $\check{\boldsymbol{T}}$ converts the coupled coordinates $\boldsymbol{q}_u$ into serial coordinates $\check{\boldsymbol{q}}_u$:

$$\underbrace{\begin{bmatrix} 1 & 0 \\ -1 & 1 \end{bmatrix}}_{\check{\boldsymbol{T}}} \underbrace{\begin{pmatrix} q_{u,1} \\ q_{u,2} \end{pmatrix}}_{\boldsymbol{q}_u} = \underbrace{\begin{pmatrix} q_{u,1} \\ \check{q}_{u,2} \end{pmatrix}}_{\check{\boldsymbol{q}}_u} . \tag{3.1}$$

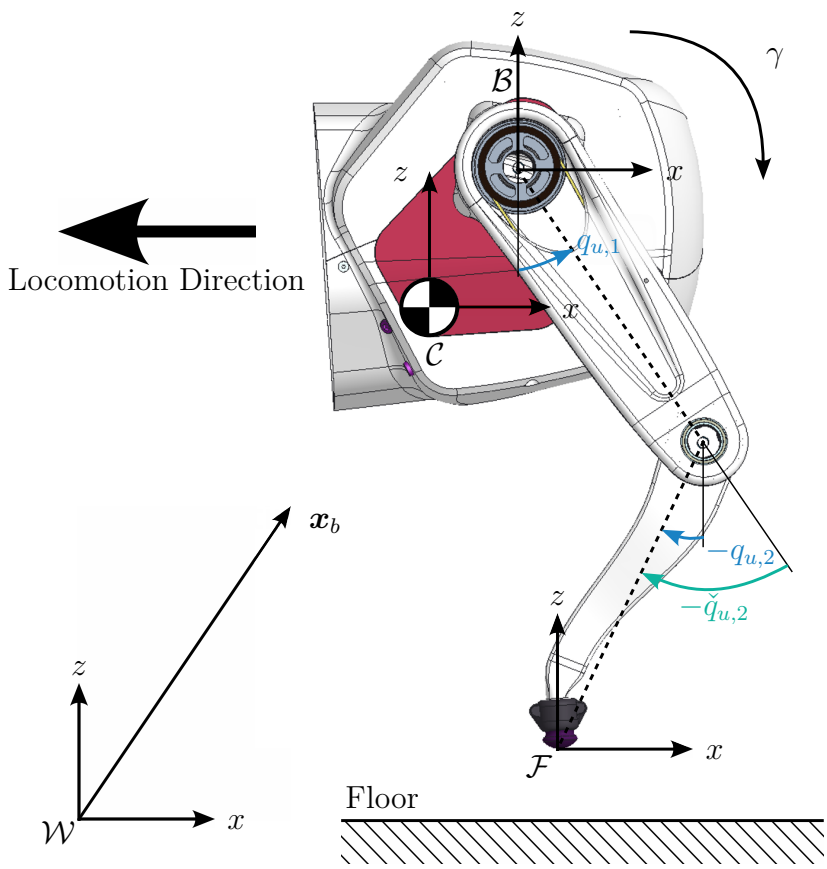


Figure 3.1: Kinematics of the one-leg hopper

The robot can move planar in the world frame $\mathcal{W}$. The vector $\boldsymbol{x}_b$ denotes the position of $\mathcal{B}$ in $\mathcal{W}$. An additional DoF γ allows a rotation of $\mathcal{B}$ in the $\mathcal{W}_{x,z}$-plane. It enables the hopper to tip over. This makes the system similar to a Spring-Loaded Inverted Pendulum (SLIP) model and control more challenging.

3.2 Hardware and Simulation of the Hopper

The floating base kinematics is implemented in hardware by mounting the hopper on a rod, called boom (cf. Figure 3.2). The boom can rotate about two axes and enables the robot to move in polar coordinates about the boom's suspension. By measuring the angles of the boom suspension (α and β) and its radius, the forward distance in $\mathcal{W}_x$ and jumping height in $\mathcal{W}_z$, and therefore the base position in the world $\boldsymbol{x}_b$, can be calculated by the circumferences. The axis γ is located slightly above the origin of $\mathcal{B}$ and can be locked by a clamp.

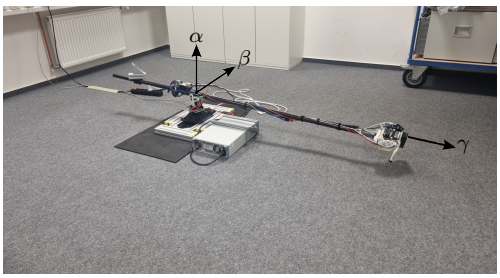


Figure 3.2: Hardware setup of the hopper with attached boom

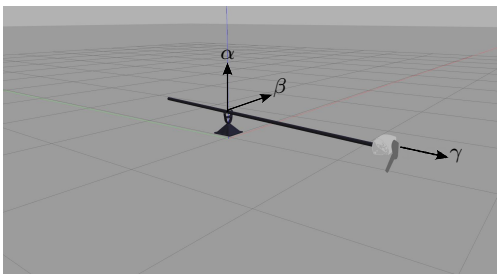


Figure 3.3: Simulation model of the hopper in Gazebo

The robot is actuated by Series Elastic Actuators (SEAs). The drivetrain consists of a servo motor, spring and link. The kinematic coupling is realized through belts. The servo motors can be torque or position controlled and have an internal controller of 8 kHz to utilize the commands. They have high friction and motor inertia with respect to the links. The mechanical springs are made of carbon fiber to reduce weight. Link positions are measured by magnetic position sensors. An Inertial Measurement Unit (IMU) is mounted on top of the hopper. It measures the accelerations of the base in frame $\mathcal{B}$. The complete list of known hardware parameters is shown in Table 3.1.

The communication with the robot is implemented with a real-time computer and the firmware is programmed with the MATLAB® Simulink Coder™ (formerly Real-Time Workshop®). Thus, the control framework for the robot can be directly programmed in Simulink® and communication with the robot in real-time is possible.

A digital twin of the hopper was created to support the development. A simulation model was created in the Gazebo multi-robot simulator [33] (cf. Figure 3.3). For rapid prototyping, the same Simulink® model can control the robot either on hardware or in simulation.

Table 3.1: Hardware specifications of the one-leg hopper

Motor inertia b:	0.020	$\mathrm{kg\,m^2}$
Spring stiffness $\boldsymbol{K}$:	$\mathtt{diag}(2.950, 3.100)$	$\mathrm{N\,m\,rad^{-1}}$
Robot mass m:	0.987	kg
Boom radius:	1.360	m
Boom inertia:	0.547	$\mathrm{kg\,m^2}$
Simulink control frequency:	1	kHz
Internal motor control frequency:	8	kHz
Static motor friction (estimate):	0.1	$\mathrm{N\,m}$
Viscous motor friction (estimate):	0.05	$\mathrm{N\,m\,s\,rad^{-1}}$

3.3 Trajectory Generation by Demonstration

In the following, the framework for controlling the robot with haptic teleoperation is laid out. The mapping from the omega.3 input to the robot aims to be intuitive and robust. In addition, the haptic force feedback supports exciting the natural harmonics of the hopper. Locomotion by teleoperation was not possible with lateral movement. Therefore, the hip rotation γ is locked during the teleoperation and optimization on the robot.

The trajectory, generated from the recorded data, is designed to have the following properties: It is closed and periodic, with a base frequency of ω and a phase between the two joint signals of φ. Thus $\boldsymbol{q}_{t=0} \triangleq \boldsymbol{q}_{t=T}$, with T representing the period. Further the trajectory needs to be continuously deformable by a set of parameters for optimization. To this end, imitation learning often relies on Dynamic Motor Primitives (DMPs) [34]. However, an approximation by Fourier series is chosen, to make the physical interpretation of the parameters more intuitive.

3.3.1 Input Processing and Control

In order to control the hopper, the input of the operator needs to be mapped to the two actuators of the robot. The vector $\boldsymbol{x}_{d,o}$ is the input in the Cartesian frame of the omega.3 device Ω, see Figure 1.2.[1] The position of the foot tip of the robot is controlled w.r.t. $\mathcal{B}$. This position vector is called $\boldsymbol{x}_f$. The corresponding joint positions are calculated by inverse kinematics. If only the configuration with the knee on the right side is considered (as shown in Figure 3.1), there exists a direct unique mapping from $\boldsymbol{x}_f \rightarrow \boldsymbol{q}$. This function, called `SOLVE()`, is only defined in the workspace of the robot. Therefore, $\boldsymbol{x}_{d,o}$ is saturated to the workspace, as illustrated in Figure 3.5. This function is called `SAT()`.[2] The sensitivity of the input device can be changed by the positive, diagonal gain matrix $\boldsymbol{K}_S \in \mathbb{R}^{2\times 2}$. An offset ensures, that the zero position of the input device sets the robot in a suitable configuration. In this case, the foot tip is positioned 0.22 m below the base.

Taken together, equation (3.2) describes the mapping from Ω to $\mathcal{B}$ to calculate $\boldsymbol{x}_{d,sat}$, and equation (3.3) shows the inverse kinematic calculation:

$$\boldsymbol{x}_{d,sat} = \texttt{SAT}\left(\boldsymbol{K}_S\,\boldsymbol{x}_{d,o} + \begin{pmatrix} 0 \\ 0.22\,\mathrm{m} \end{pmatrix}\right) , \tag{3.2}$$

$$\boldsymbol{q}_{d,sat} = \texttt{SOLVE}(\boldsymbol{x}_{d,sat}) . \tag{3.3}$$

Realizing the desired trajectory $\boldsymbol{q}_{d,sat}(t)$ on the link-side of elastic systems is challenging, as the link is not directly actuated. State-of-the-art link-side controllers like the Elastic Structure Preserving Impedance (ESπ) controller generate a virtual link-side control input [35]. However, depending on the stiffness of the system, there is a bandwidth limitation for link-side control. Further, these types of controllers need precisely modeled motor dynamics with torque control. These effects cause the ESπ controller to perform poorly on the one-leg hopper. Explosive movements from the operator can not be utilized.

[1] Inputs in the Ω_y axis are neglected.

[2] The function `SOLVE()` is calculated by the Symbolic Toolbox of MATLAB® . The function `SAT()` is calculated by a transformation in polar coordinates. Both functions are presented in Section A of the Appendix

Figure 3.4: The haptic input device with corresponding frame Ω

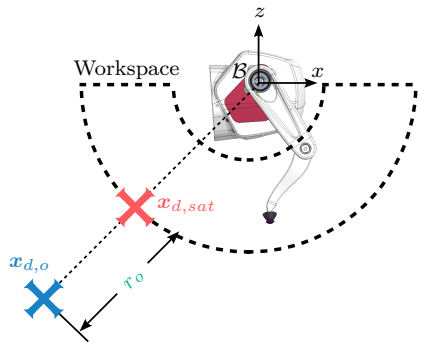


Figure 3.5: Input coordinates, overdrive gain and workspace limitation of the hopper

Consequently, the internal position controller of the motors is used. This has the advantage, that with 8 kHz control frequency and high control gains, fast changes in $\boldsymbol{q}_a$ can be commanded. Further, this control approach shapes the system dynamics minimally. With stiff motor PID controllers, the effect of motor friction and inertia is minimized and the dominant dynamics of the system is the link dynamics. The two-mass swinger of Figure 2.4 becomes a one-mass swinger. Controlling the motor side of the hopper means, that with no deflection of the spring, the link position is controlled and with deflection through e.g. external forces, the loading of the springs is affected.

To generate more explosive movements, the load of springs can be increased, if the link is in contact with the ground and the motors are driven further. This is implemented by using the radial distance between $\boldsymbol{x}_{d,o}$ and $\boldsymbol{x}_{d,sat}$ as an overdrive gain called r_o (cf. Figure 3.5). This effect can be influenced by the gain k_o and is saturated to 0.5. Taken together, the desired motor trajectory $\boldsymbol{q}_{d,a}(t)$ is calculated by:

$$\boldsymbol{q}_{d,a}(t) \;=\; \begin{bmatrix} -k_o\, r_o & 0 \\ 0 & k_o\, r_o \end{bmatrix} \boldsymbol{q}_{d,sat}(t)\ . \tag{3.4}$$

3.3.2 Haptic Force Feedback

The force feedback is designed to give the operator physical intuition of the system. The operator controls the foot tip of the system and feels the forces acting on the foot tip $\boldsymbol{f}_{ext,\mathcal{F}}$. These can be calculated by the link-side Jacobian in $\mathcal{F}$ called $\boldsymbol{J}_{\mathcal{F}}$ $\in \mathbb{R}^{2\times 2}$:

$$\boldsymbol{f}_{ext,\mathcal{F}} \;=\; \texttt{inv}\left(\boldsymbol{J}_{\mathcal{F}}^{T}\right)\, \boldsymbol{\mathcal{Q}}_u \;. \tag{3.5}$$

Without additional sensors, $\boldsymbol{\mathcal{Q}}_u$ cannot be measured. However, with small link inertias, the effects of centrifugal, gravitational, and inertial torques can be neglected and $\boldsymbol{\mathcal{Q}}_u \cong -\boldsymbol{\tau}_j$ (cf. equation (2.2)). A virtual spring is added that pushes the haptic device back into zero position.[3] This gives the operator a feeling of the workspace of the input device. Additionally, a damper is implemented to improve stability. The resulting function for the force feedback calculation is:

$$\boldsymbol{f}_{\Omega} \;=\; \boldsymbol{K}_F\, \boldsymbol{f}_{ext,\mathcal{F}} - \boldsymbol{K}_P\, \boldsymbol{x}_{d,o} - \boldsymbol{K}_D\, \dot{\boldsymbol{x}}_{d,o} \;. \tag{3.6}$$

The vector $\boldsymbol{f}_{\Omega}$ contains the forces in Ω_x and Ω_z direction and is sent to the haptic device. The positive, diagonal gain matrices $\boldsymbol{K}_F$, $\boldsymbol{K}_P$ and $\boldsymbol{K}_D$ are $\in \mathbb{R}^{2\times 2}$ and set the magnitude of the force feedback respectively. The principle sketch in Figure 3.6 depicts the behavior of the haptic device.

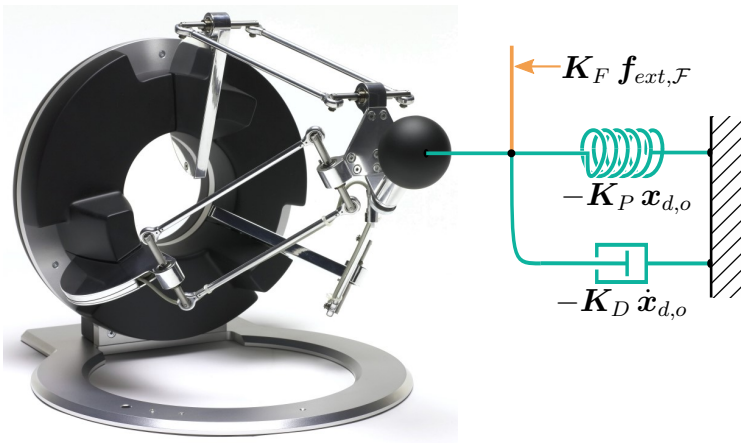


Figure 3.6: Principle sketch of the force feedback

[3]This spring also locks Ω_y, as it is set very stiff in this direction.

Figure 3.7 shows the link and motor coordinates over time of the teleoperated hopper. The operator starts with the standing robot. Thereafter the hopper is exited. The operator produces an oscillation with consistent period. At first, controlling the foot tip of the robot rather than the base seems unintuitive, but combined with the force feedback, a feeling of pushing against something is simulated. The sensitivity of the input device, combined with the gain of the force feedback, influence the behavior during teleoperation. A low input sensitivity, and small motor position changes are sufficient to excite the system. This indicates that the operator is stimulating the natural harmonics of the hopper. However, the operator was only able to create fluent locomotion after some training.

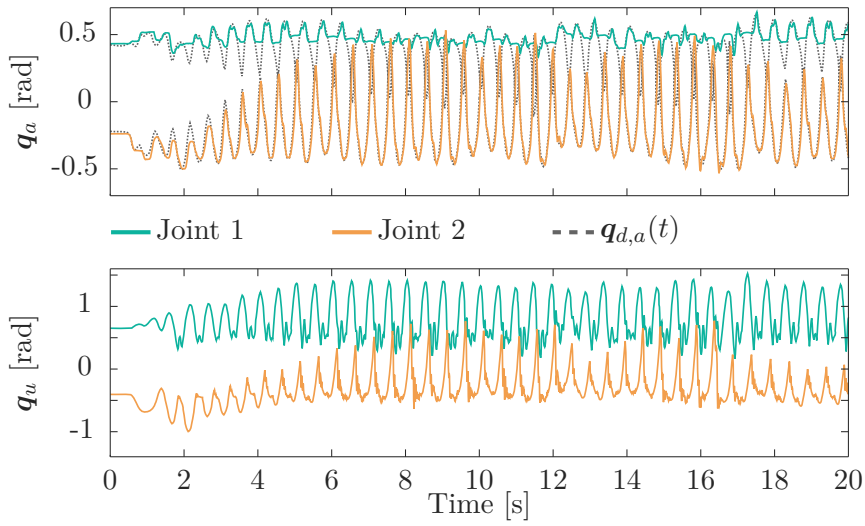


Figure 3.7: Joint coordinates over time during teleoperation of the hopper

3.3.3 Approximation of the Signal by Fourier Series

The desired motor trajectory signal $\boldsymbol{q}_{d,a}(t)$ of Figure 3.7 is approximated by Fourier series. The base frequency ω_0 and phase difference φ_0 are determined by fitting a sine wave to both signals via the least squares method. The chosen base frequency ω_0 is the mean of the frequencies of both fitted sine waves. Table 3.2 lists the identified frequencies. A frequency analysis of the signals shows, that this frequency also has the most signal power (cf. Figure 3.8). The harmonic at $2\,\omega_0$ is also recognizable while other frequencies are insignificant.

Table 3.2: Identified base frequency and phase of the demonstrated trajectory

Base frequency of $q_{d,a,1}$: $\omega_{0,1}$:	2.035	Hz
Base frequency of $q_{d,a,2}$: $\omega_{0,2}$:	2.040	Hz
Mean frequency: ω_0:	2.038	Hz
Mean period: T_0:	0.491	s
Mean phase: φ_0:	0.195	s

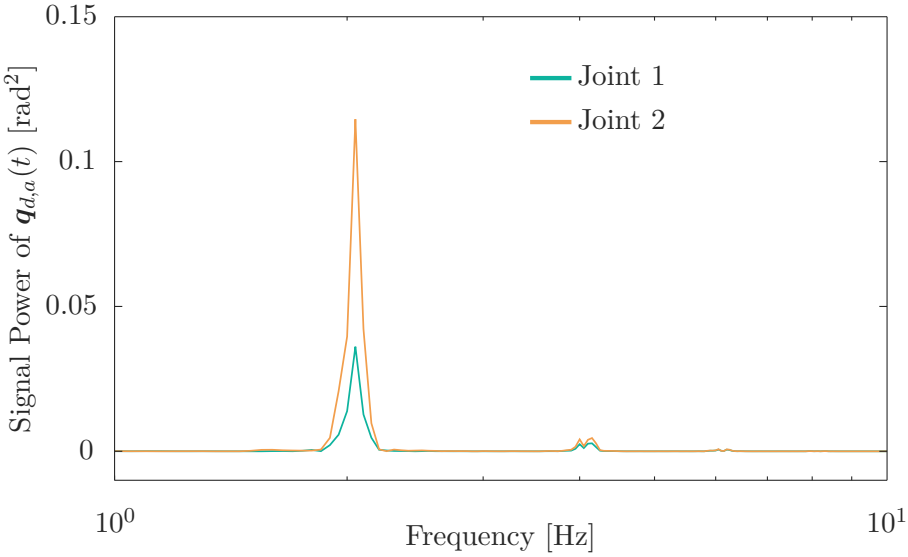


Figure 3.8: Power spectral density of the operator's input during locomotion

As previously noted, the input from the operator is quite periodic. In order to smoothen it, the mean signal is calculated over 23 periods, as depicted in Figure 3.9. A fourth-order Fourier series (cf. equation (3.7)) is fitted to the data by the least squares method:[4]

$$\texttt{traj}_0(t) = \begin{pmatrix} a_{0,1} + \sum_{i=1}^{4} a_{i,1} \cos(i\omega_0 t) + b_{i,1} \sin(i\omega_0 t) \\ a_{0,2} + \sum_{i=1}^{4} a_{i,2} \cos(i\omega_0 (t+\varphi_0)) + b_{i,2} \sin(i\omega_0 (t+\varphi_0)) \end{pmatrix} . \quad (3.7)$$

The base frequency ω is a parameter during the fit. It differed only slightly between joint 1 and joint 2. It is substituted by ω_0 in both functions, to satisfy the properties of rhythmic trajectories (cf. Section 3.3).

[4]The MATLAB® command `fit(xdata,ydata,'fourier4')` is used. The parameters are in Section B of the Appendix,

Figure 3.10 shows, that the mean signal is approximated accurately. Moreover, the fitted function $\texttt{traj}_0(t)$ is close to the recorded data from the operator (cf. Figure 3.11). The desired motor trajectory can also be described in Cartesian space. It is the trajectory, the foot tip has, if there is no control error and no deflection in the spring (cf. Figure 3.12).

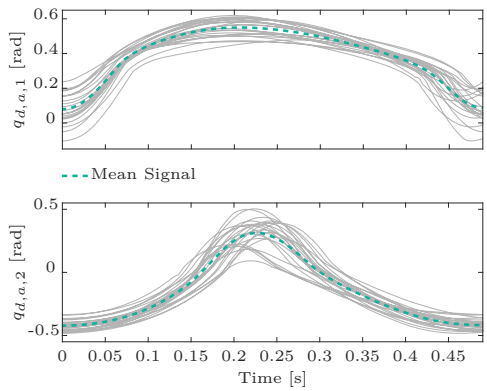


Figure 3.9: Mean signal of the demonstrated trajectory over 23 periods

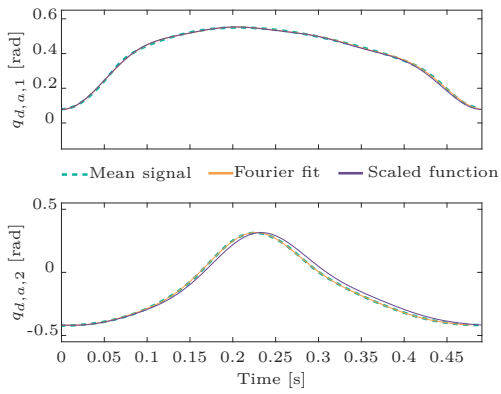


Figure 3.10: Comparison of $\texttt{traj}_0(t)$ and the mean signal of the demonstrated trajectory

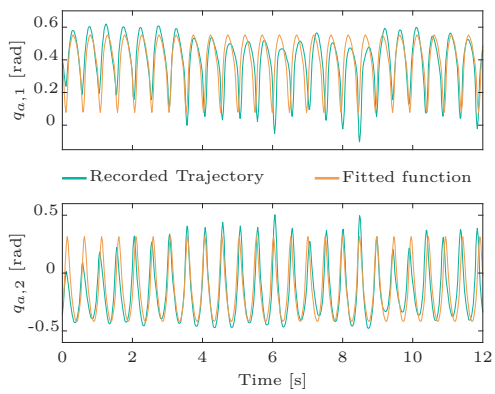


Figure 3.11: Comparison of the demonstrated trajectory and $\texttt{traj}_0(t)$ in joint space

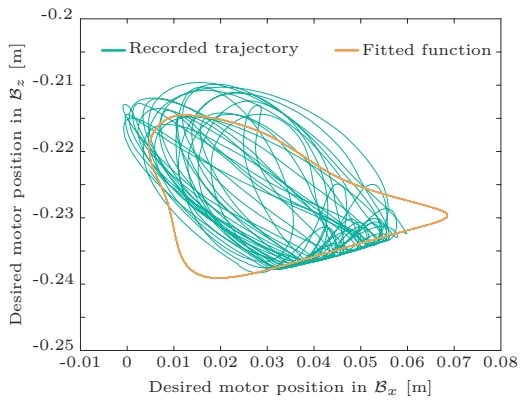


Figure 3.12: Comparison of the demonstrated trajectory and $\texttt{traj}_0(t)$ in $\mathcal{B}$

3.3.4 Trajectory Analysis

Figure 3.13 depicts the link-side foot tip trajectory of $\texttt{traj}_0(t)$ and the estimated phases in locomotion. The trajectory is projected relative to the CoM frame $\mathcal{C}$ in the x-axis and relative to the world frame $\mathcal{W}$ in the z-axis. Consequently, the z-axis shows the ground clearance of the foot tip $\boldsymbol{x}_f$. The movement of the foot tip is counter clockwise.

Notably, $\boldsymbol{x}_f$ is negative in the $\mathcal{W}_z$ axis as well. This implies that the foot tip penetrates the ground. This discrepancy comes from the flexibility of the boom. As a result, the exact moment of contact between foot tip and ground is unknown. Contact is estimated to occur, when $\boldsymbol{x}_f$ approximates zero in $\mathcal{W}_z$. The link stays in contact as long as $\boldsymbol{x}_f$ in $\mathcal{W}_z < 0$. The leg is most compressed, when $\texttt{MAX}(\tau_{j,2} - \tau_{j,1})$.[5] This is indicated in the Figure by red markers. After the compression phase, thrust is applied. During maximum thrust, the base acceleration in $\mathcal{W}_z$ is maximal, as indicated in yellow. The foot tip position is to the right of the CoM during thrust. This results in a forward motion of the hopper leftwards. Afterwards, the liftoff phase begins. The foot tip is close above the ground before it is retracted. During retraction and repositioning, the base experiences the greatest acceleration downwards (marked in purple), indicating falling.

[5]This is derived by the coupled kinematics of the hopper, see Figure 3.1

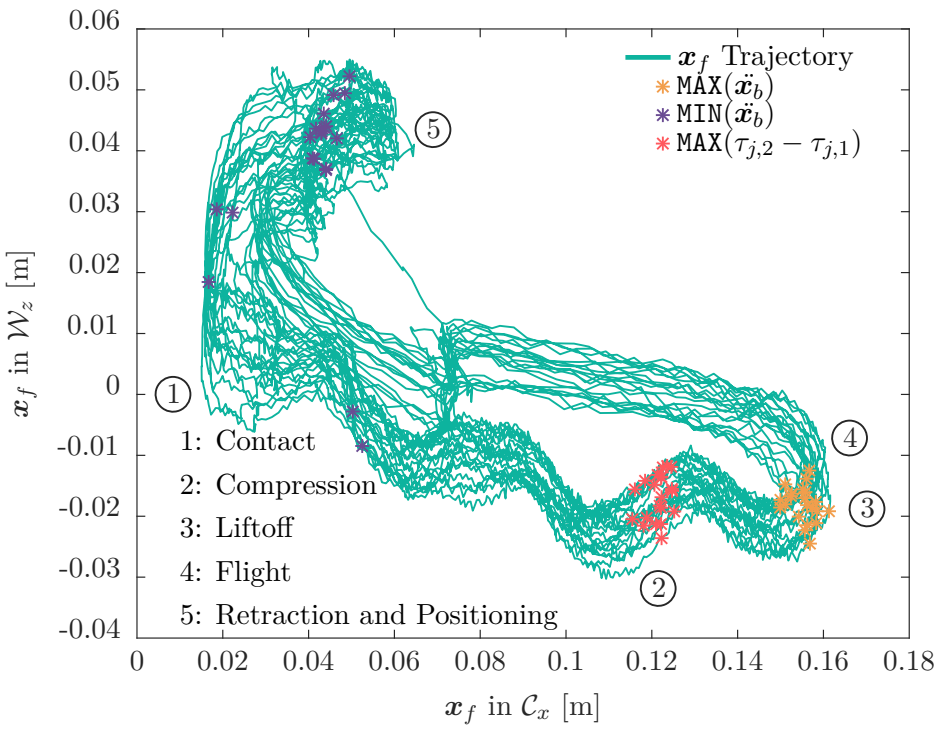


Figure 3.13: Phases of one-leg locomotion leftwards in the foot tip trajectory

3.4 Optimization for Energy Efficient Locomotion

In the following, the optimization for CoT is laid out and evaluated. The optimization is done in simulation and on the hardware.

3.4.1 Comparison of Simulation and Hardware

Transferring simulation results to the actual robot requires an accurate simulation model. To determine how well the simulation model performs, the approximated $\texttt{traj}_0(t)$ is replayed in simulation and hardware, and the relevant states and signals are compared.

Figure 3.14 compares the motor position signals $\boldsymbol{q}_a$ of simulation and hardware. The desired $\texttt{traj}_0(t)$ cannot be realized on hardware. There is a significant discrepancy between $q_{d,a,1}$ and $q_{a,1}$. Joint 1 experiences a the hightest load during impact and compression. During this time, the motor runs into its torque limits and cannot realize the desired trajectory. Internal friction, on the other hand, prevents it from being driven back. This effect is helpful for conserving energy. Figure 3.15 shows high frequency oscillations of the links on hardware, that are not present in simulation. In terms of the basic frequencies and amplitudes of link and motor signals, there are few variances between hardware and simulation. The accelerations of the base in $\mathcal{B}_x$ and $\mathcal{B}_z$ direction are shown in Figures 3.16 and 3.17. It has been noticed that the boom is quite flexible when it is under stress during locomotion. This is the cause of the high frequency oscillations, mostly in $\mathcal{B}_z$, that are not present in simulation.

Small variations of the hardware have a noticeable impact on the behavior of the robot. For instance, the foot tip broke after the demonstration part and has been changed for optimization (cf. Figure 3.18). The new foot tip is stiffer and larger. This negatively impacts the performance of the demonstrated trajectory. The new foot tip is slipping on the floor, as indicated by the scattering in Figure 3.19.

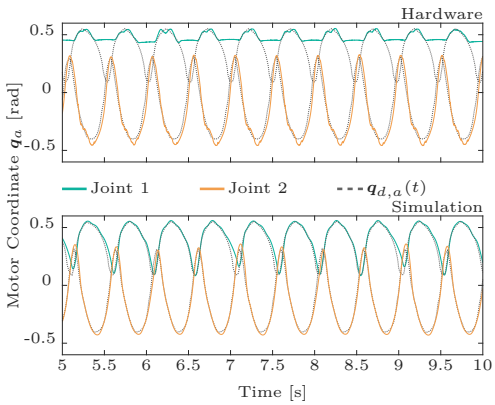


Figure 3.14: Comparison of motor signals between simulation and hardware

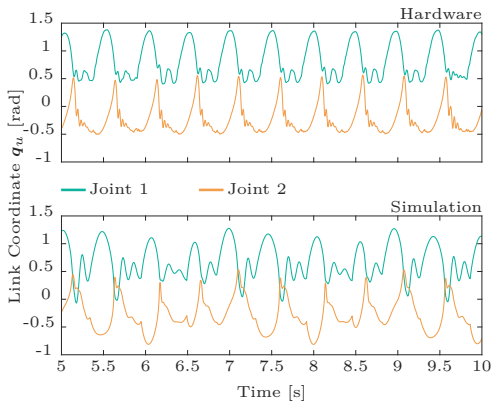


Figure 3.15: Comparison of link signals between simulation and hardware

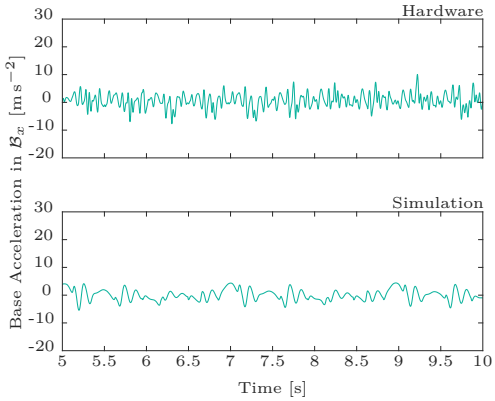


Figure 3.16: Base acceleration in $\mathcal{B}_x$ in simulation and hardware

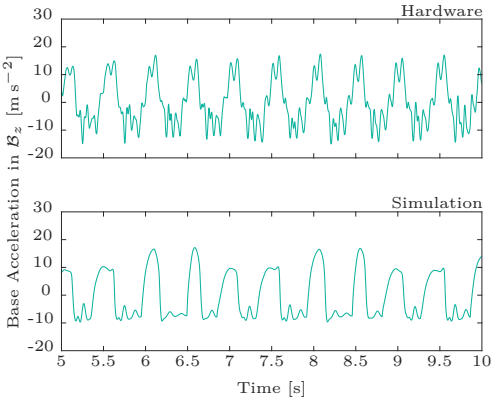


Figure 3.17: Base acceleration in $\mathcal{B}_z$ in simulation and hardware

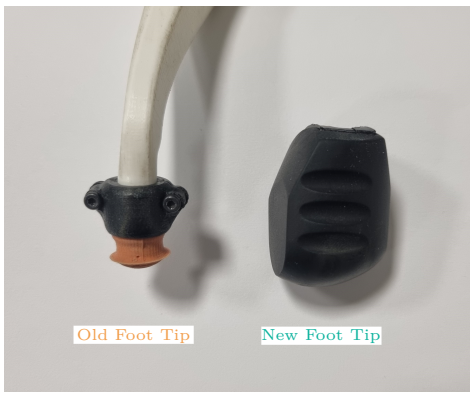


Figure 3.18: Old and new foot tip of the hopper

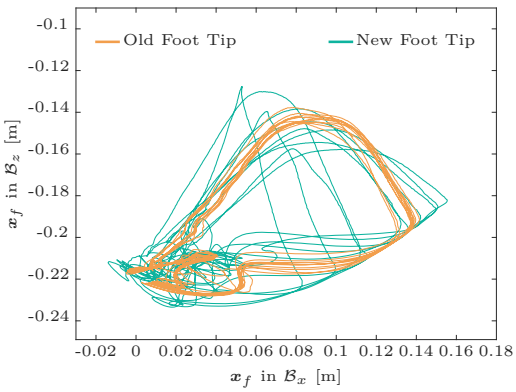


Figure 3.19: Scattering in the link-side foot tip motion with the new foot tip

3.4.2 Parameters and Cost Function

There are 18 parameters in the Fourier series (cf. equation (3.7)). They would take too long to optimize, especially on hardware. Therefore, the parameter space is reduced to six parameters, that are also more in intuitive to interpret. The initial function is scaled by A_1 and A_2 and offset by d_1 and d_2:

$$\begin{aligned} \texttt{traj}(t, \Theta) &= \begin{pmatrix} A_1 \\ A_2 \end{pmatrix} \texttt{traj}_0(t, \omega, \varphi) + \begin{pmatrix} d_1 \\ d_2 \end{pmatrix} , && (3.8) \\ \Theta &= \{A_1,\, A_2,\, d_1,\, d_2,\, \omega,\, \varphi\} . && (3.9) \end{aligned}$$

Further the period ω and phase φ are varied. The offset parameters directly influence the foot tip position $\boldsymbol{x}_f$, while the shape of the foot tip trajectory in Cartesian space is deformed by the phase parameter. The initial parameters and their corresponding limits in brackets are shown in Table 3.3. Broader restrictions have harsher negative effects on the hardware, such as collisions of the base with the ground.

Table 3.3: Optimization parameters within their respective bounds

Gain on joint 1 trajectory:	A_1	[0.8..1.2]
Gain on joint 2 trajectory:	A_2	[0.8..1.2]
Offset of joint 1:	d_1	[−0.1..0.1] rad
Offset of joint 2:	d_2	[−0.1..0.1] rad
Period of the rhythmic trajectory, corresponding to ω:	T	[0.4..0.6] s
Phase parameter:	φ	[0.0..0.24] s

As stated in Section 2.1, the CoT depends on the velocity of the system. This is considered in the optimization. The velocity of the robot with optimized parameters is supposed to match the velocity during demonstration. Therefore, the CoT is enhanced by a weight on deviations from the velocity v_0, corresponding to $\texttt{traj}_0(t)$. The designed cost function for the optimization is:

$$\mathrm{CoT}_{v0} = \mathrm{CoT}\; k^{-1} \underset{k}{\texttt{SAT}} \left(\left(\frac{v - v_0}{2\, v_0} \right)^2 \right) . \tag{3.10}$$

The second term is saturated to $k = 0.01$ at the lower end. Consequently, the cost function cannot be zero. With v close to v_0 and the saturation in effect: $\mathrm{CoT} = \mathrm{CoT}_{v0}$. Thus, the CoT can be optimized further. If the leg hopps backwards the CoT becomes negative. This is harmful to the hardware. The carbon torsion springs have lesser strength when being compressed, contrary to being pulled, which is considered in the kinematic design of the hopper. Therefore, negative CoT values are punished by a factor of -10.

The cost function CoT_{v0} is evaluated over an episode to limit evaluation noise. Each episode has a different set of parameters, determined by the optimization algorithm. At the beginning of an episode, the hopper does $n_0 = 5$ jumps with the given set of parameters. During this transient, the overall system has enough time to adapt to the new dynamics imposed by the new trajectory. Then the cost function is evaluated after $n = 20$ jumps. After the episode, the traversed distance and elapsed time are evaluated and from these values the mean velocity v is calculated. The parameter T determines the duration of a jump, it varies with the parameters in Θ. After each episode, the operator can either start a new one, or reset the hopper, if it is near its bounds of movement.[6]

For the energy calculation the mechanical output energy of the motors is used. For SEAs it is calculated by:

$$E = \int_0^{n\,T} \tau_{j,1}\,\dot{q}_{a,1} + \tau_{j,2}\,\dot{q}_{a,2}\;dt\;. \tag{3.11}$$

This has the advantage, that it makes the results comparable over different systems. In contrast, effects like motor efficiency and friction can be considered by using the electrical energy of the motors.

The Tree-structured Parzen Estimator (TPE) algorithm is implemented with the Optuna framework[7] [31] to perform the optimization (cf. Section 2.3.3). Five episodes are used for bootstrapping.

[6]This calculation is timed with the help of the Stateflow® package in MATLAB® Simulink™.
[7]The code is presented in the Appendix Section C.

3.4.3 Optimization Results

Three different optimizations are performed, with 50 episodes each. An optimization for CoT and for CoT_{v0} (as defined in equation (3.10)) on the hardware. The new foot tip is used. Additionally, CoT_{v0} is optimized in simulation. The corresponding parameter sets are called Θ_{CoT}, $\Theta_{CoT_{v0}}$ and Θ_{Sim}. They are listed in Table 3.4. Figure 3.21 illustrates how the parameters have changed from their initial state. The tendency for Θ_{Sim} parameters is often contrary to the parameters of the hardware optimization. For instance, in simulation, ω is increased and A_1 is decreased, however on hardware, the opposite occurs.

Table 3.4: Optimized parameters of the desired motor trajectory

	A_1	A_2	d_1 [rad]	d_2 [rad]	ω [Hz]	φ [s]
Θ_0	1.000	1.000	0.000	0.000	2.038	0.195
Θ_{CoT}	1.105	0.850	0.011	−0.048	1.910	0.210
$\Theta_{CoT_{v0}}$	1.034	1.032	0.019	−0.052	1.958	0.190
Θ_{Sim}	0.858	1.106	0.096	−0.020	2.185	0.177

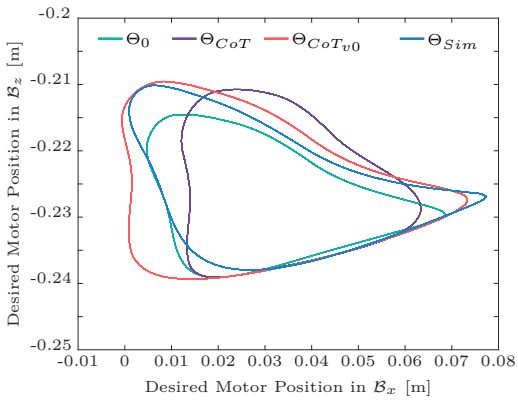


Figure 3.20: Initial and optimized motor trajectories in $\mathcal{B}$

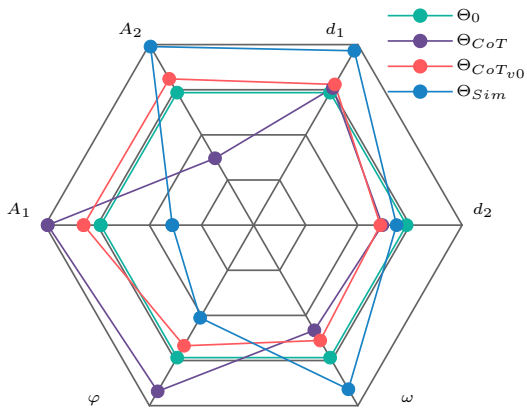


Figure 3.21: Spider graph of the tendency of the optimized parameters

The related motor trajectories are displayed in Cartesian space in Figure 3.20. It is noteworthy, that the trajectory of Θ_{CoT} is smaller than the others. Less motor movement is favorable to conserving energy.

According to the Optuna framework[8], the most sensitive parameter for improving CoT is φ (cf. Figure 3.23). The phase parameter shapes the trajectory in Cartesian space and can cause the hopper to move backwards, which is punished in the cost function. If the forward speed of the hopper is taken into account, with CoT_{v0}, the offset parameter d_1 has the biggest impact on the cost function (cf. Figure 3.24). It directly influences the foot position and therefore the forward speed, if the foot is placed e.g. further behind the CoM of the robot. The parameter importance for CoT_{v0} in simulation roughly matches the hardware, with a greater relevance of ω (cf. Figure 3.25). The evolution of the CoT over 50 trials is shown in Figure 3.22. Clearly, the room for improvement is marginal.

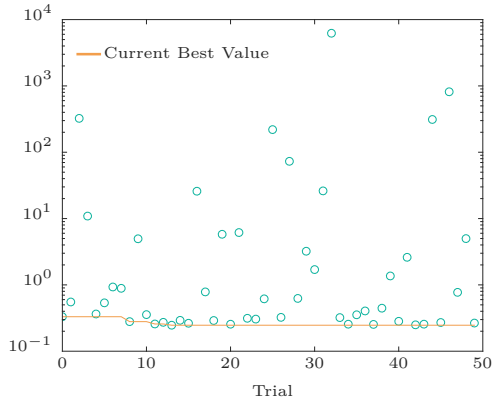


Figure 3.22: Study history of the optimization for CoT_{v0}

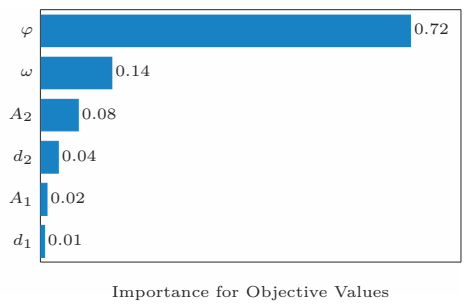


Figure 3.23: Parameter importance for CoT on hardware

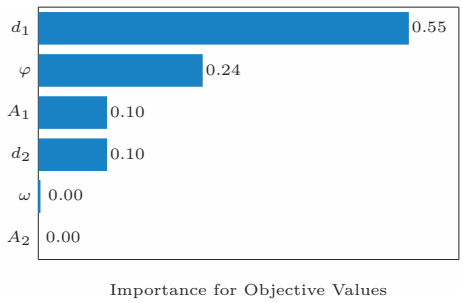


Figure 3.24: Parameter importance for CoT_{v0} on hardware

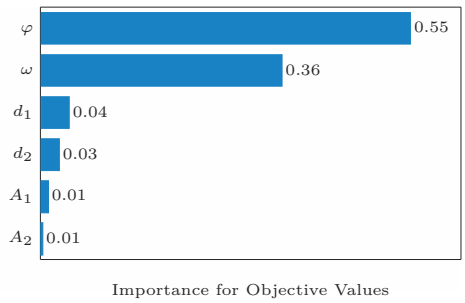


Figure 3.25: Parameter importance for CoT_{v0} in simulation

[8]The command `optuna-dashboard()` was used to evaluate the results.

The optimized parameters are evaluated by their CoT and forward velocity v over five episodes, as depicted in Figure 3.26. Their minimal and maximal values are used to calculate the corresponding uncertainty in Table 3.5. In addition, the mean foot tip clearance ($\max \boldsymbol{x}_f$ in $\mathcal{W}_z$) is assessed. These values were calculated with help of the boom position. Consequently, due to the boom's flexibility, these values are not exact. However, they can be compared in relation to other. The CoT as well as v deteriorates with the new foot tip. The optimization for CoT_{v0}, counteracts that. The optimization on hardware significantly improved the CoT over the initial value with the new foot tip. The optimization in simulation, performs poorly. Hence, the sim to real transfer fails, even though the accuracy of the model looked promising in Section 3.4.1. Little differences in the hardware parameters cause noticeable changes in performance. The optimization only for CoT does not show much improvement over CoT_{v0}. However, v is slower and foot tip clearance is reduced.

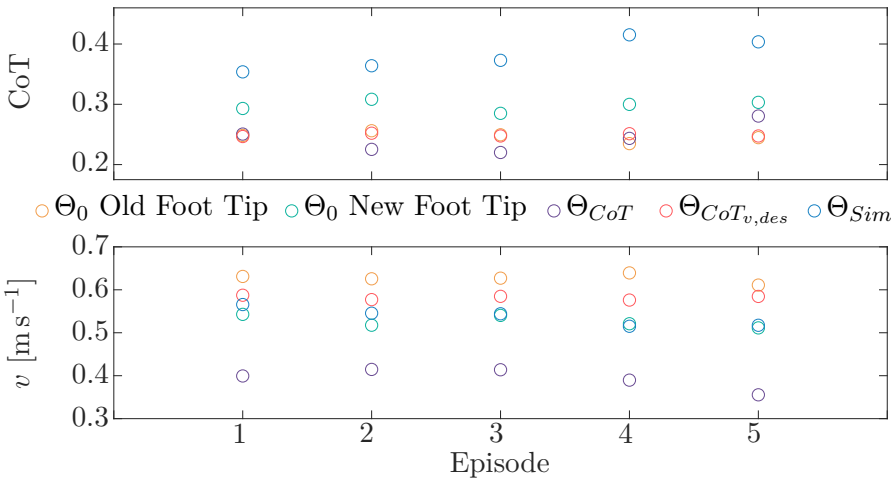


Figure 3.26: Evaluation of the optimized parameters over five episodes

Taken together, further room for improvement over the initial values is slim. As proposed, if the natural harmonics of the hopper are exploited, locomotion is energy efficient. The performance of the operator within the teleoperation framework shows good results. Optimization, on the other hand, can be used to maintain performance with hardware or environmental changes.

Table 3.5: Performance of the optimized parameters

	CoT	v [m s^{-1}]	foot tip clearance [m]
Θ_0 old foot tip	0.2467 ± 0.0214	0.6268 ± 0.0283	0.048
Θ_0 new foot tip	0.2979 ± 0.0233	0.5266 ± 0.0315	0.051
Θ_{CoT}	0.2440 ± 0.0604	0.3946 ± 0.0591	0.047
$\Theta_{CoT_{v0}}$	0.2489 ± 0.0055	0.5820 ± 0.0112	0.056
Θ_{Sim}	0.3819 ± 0.0614	0.5376 ± 0.0506	0.039

3.5 Balancing through Reinforcement Learning

In the following, the rotation of the base about γ is unlocked. This allows the hopper to tip over. Stabilizing the locomotion with lateral movement is important to apply the control strategies to legged robots that move in the world without constraints. Raibert (1986) has shown, that these types of hoppers can be stabilized by dynamic balance [1]. This control strategy adjusts the foot tip position in flight phase and manipulates the base angle γ in stance phase.

In this thesis locomotion is accomplished by replaying a desired trajectory, in contrast to Raibert's control algorithm. A reinforcement learning algorithm is trained to find a policy that prevents tipping over by adding an offset signal to the trajectory.

3.5.1 Definition of Action-space, Observation-space, and Reward Function

The Soft Actor-Critic (SAC) algorithm (see Section 2.3.2) was chosen and implemented with the stable-baselines3 library[9] [24]. The algorithm is running at 66.6 Hz and receiving information about the joint positions and velocities, the velocity and acceleration of the hopper's base and the desired feedforward trajectory as well as the base height, tipping angle and its derivative. The terminal state is sent, when the base of the hopper touches the ground. Then the episode is finished and the simulation as well as the feedforward trajectory are reset. The algorithm adds

[9]The code is presented in the Appendix Section D.

a varying offset of ± 0.2 to the desired trajectory. The observation space $\mathcal{O}$ and action space $\mathcal{A}$ is defined by:

$$\mathcal{O} = \{\texttt{traj}(t),\, \boldsymbol{q}_u,\, \dot{\boldsymbol{q}}_u,\, \boldsymbol{q}_a,\, \dot{\boldsymbol{q}}_a,\, x_{b,z},\, \dot{\boldsymbol{x}}_b,\, \ddot{\boldsymbol{x}}_b,\, \gamma,\, \dot{\gamma},\, \texttt{TERMINAL}\}\ , \tag{3.12}$$

$$\mathcal{A} = \{q_{RL,1},\, q_{RL,2}\}\ , \tag{3.13}$$

$$\boldsymbol{q}_{d,a} = \texttt{traj}(t) + \boldsymbol{q}_{RL}\ . \tag{3.14}$$

The algorithm is rewarded when the base is upright and thus $\gamma = 0$. The reward decays with a Gaussian bell function:

$$R = \frac{1}{\sigma\sqrt{2\,\pi}} \exp\left(-\frac{1}{2}\left(\frac{\gamma - \gamma_d}{\sigma}\right)^2\right)\ . \tag{3.15}$$

Where $\gamma_d = 0$ denotes the desired base angle and $\sigma = 0.4$ denotes the variance. The cumulative reward grows, the longer the hopper stays upright.

3.5.2 Evaluation in Simulation

The parameters Θ of $\texttt{traj}(t)$ are tuned to achieve a more stable feedforward trajectory. To this end $d_2 = -0.3$ shifts the foot tip closer to the hopper's CoM. The other parameter are Θ_0. The trajectory is started at $t = 0.75\,\mathrm{s}$. Thereby the hopper achieves 2-3 jumps before tipping over, without the reinforcement learning algorithm. After 100 thousand iterations, the hopper reaches half a round. After 500 thousand, several rounds were covered. The learning is stopped after one million iterations and five hours of training.

Figure 3.27 shows the trajectory and tipping angle γ with the learned deterministic policy. At start, the hopper is reset. Then the hopper is moving for 20 s without tipping over. It is covering a distance of 1.9 m. Several seconds after the reinforcement learning policy is turned off, the hopper is tipping over, as indicated by the deterioration of γ. Only staying upright is rewarded, therefore the forward velocity with $0.1\,\mathrm{m\,s^{-1}}$ is significantly slower than with the demonstrated trajectory and locked γ.

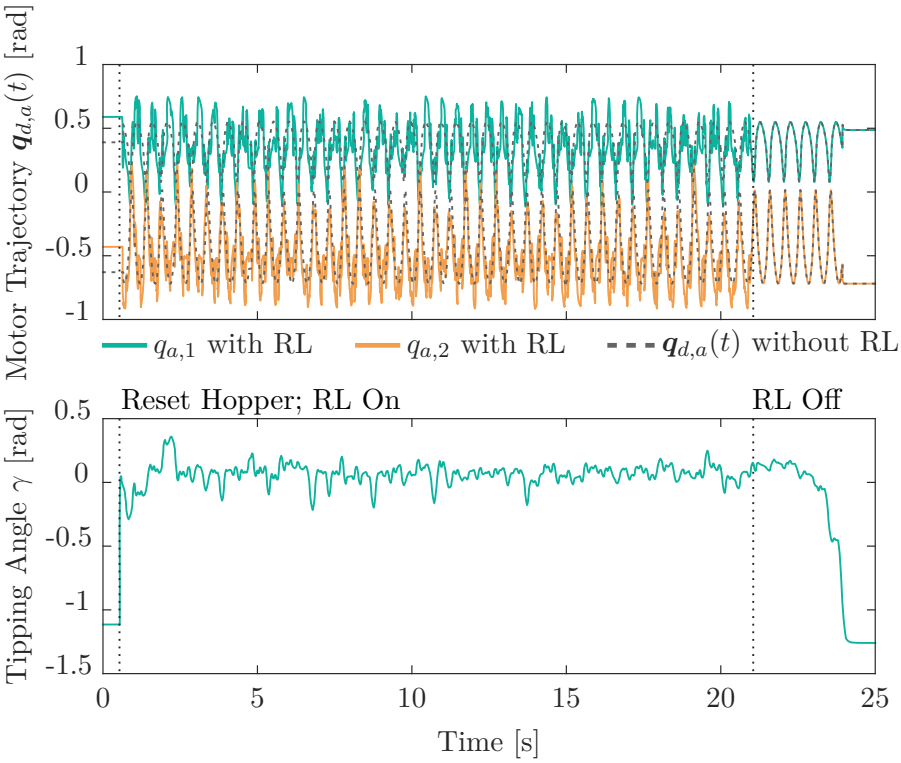


Figure 3.27: Balancing of the hopper in simulation with reinforcement learning

3.5.3 Transfer to Hardware

For the hardware implementation, the backend for the numerical calculations is switched to the machine learning framework JAX from Google [36]. This improves learning performance in simulation. The hopper was able to cover several rounds after 10 minutes of learning in real time simulation. The used hyperparameters are in Section E in the Appendix. The agent was trained in simulation without parameter variation, disturbances, and variation of the initial state after reset. Continuing the training on hardware with the policy and replay buffer from the simulation showed poor performance. Therefore, the policy is learned on the hardware from zero. At start, the replay buffer is loaded with data from 600 steps with randomly distributed actions. During training, the current policy and replay buffer is saved every ten thousand steps and after manual termination. This enables to continue learning.

Human interaction is needed to reset the hopper. To limit an unnecessary prolonging of the learning process, the reset process is partly automated. The episode is terminated when γ has reached a threshold of 40°. Then the trajectory is stopped. After it is picked up, which is detected by measuring the hopper's height, the leg is automatically moved in a defined configuration. The operator places the hopper on its foot tip and the episode is started when the joints of the hopper are loaded and $\boldsymbol{\tau}_j$ has reached a threshold. The action signal from the reinforcement policy $\boldsymbol{q}_{RL}$ is noisy. This is harmful to the motors. During training an action filter with a corner frequency of 10 Hz is used to smoothen the signal. The learning process is facilitated by increasing the friction in γ by tightening the clamp.

The learning progress is shown in Figure 3.28. After 300 episodes, the return tends to rise. The initial conditions on how the hopper is placed after reset vary. This prolongs the learning progress as the agent has to adapt. After the most common initial conditions are learned and respected in the policy, the learning progress accelerates and the hopper tips over less frequently. The learning process was stopped after five hours of training and 950 episodes. The agent favored hitting the knee on the ground, which is harmful to the hardware. The resulting deterministic policy is not capable of fully stabilizing the hopper. Nevertheless, the hopper stays significantly longer upright with this policy. Figure 3.29 compares the behavior of the hopper with and without reinforcement learning. The trajectory is stopped, when the robot tipped over.

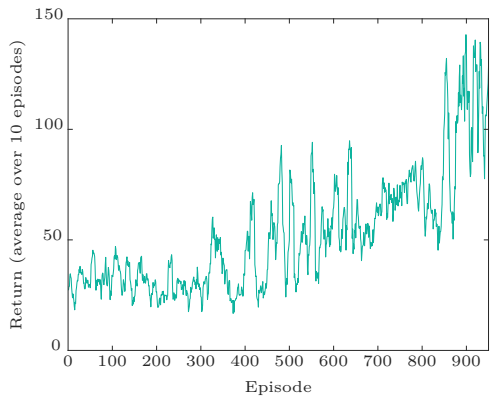


Figure 3.28: Return over episodes during learning on hardware

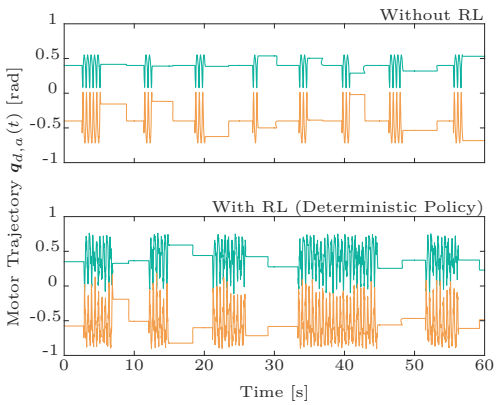


Figure 3.29: Evaluation of the resulting deterministic policy.

4 Extension to Multi-Legged Systems: C-Runner

This chapter aims to transfer the imitation approach of Chapter 3 to the biped C-Runner of the German Aerospace Center (DLR) [5] in simulation. Instead of jumping, a walking gate is generated by teleoperation.

4.1 Setup and Simulation Model

The C-Runner is a biped with six Degrees of Freedom (DoF) (cf. Figure 4.1). The hip, knee and ankle joints are elastically actuated by Series Elastic Actuators (SEAs). Contrary to the hopper of Chapter 3, the C-Runner has no kinematic couplings. Its basic specifications are given in Table 4.1.

Table 4.1: Model specifications of the DLR C-Runner

Motor inertia b:	1.6	$\mathrm{kg\,m^2}$
Spring stiffness $\boldsymbol{K}$:	$\mathtt{diag}(450, 564, 423, 450, 564, 423)$	$\mathrm{N\,m\,rad^{-1}}$
Robot mass m:	69	kg
Simulation frequency:	1	kHz

Figure 4.4 shows the frames and coordinates of the robot. The joints 1, 2 and 3 actuate left leg and joints 4, 5 and 6 the right leg. Equally to the hopper, the base frame $\mathcal{B}$ is located in joint 1, frame $\mathcal{C}$ is located in the Center of Mass (CoM) and $\boldsymbol{x}_f$ denotes the foot position relative to $\mathcal{B}$. In contrast, $\boldsymbol{x}_{f,left}$ and $\boldsymbol{x}_{f,left}$ do not denote the foot tips, but the positions of joint 3 and joint 6 respectively. This has the advantage, that the same inverse kinematics function from the hopper the can be used to calculate $\boldsymbol{x}_f$, when its output is transformed by equation (3.1).

Again, the Gazebo framework [33] is used for the simulation (see, Figure 4.2). It allows a simulation in real-time. To make walking by teleoperation easier, the rotation γ of the robot's base about $\mathcal{W}$ can be locked. This is only possible in simulation, contrary to the real robot. The operator controls the two feet of the biped, with two haptic input devices. The Anarkik3D® Falcon devices [37] are used, as an low-budget alternative to the omega.3 device by force dimension®. The setup is shown in Figure 4.3.

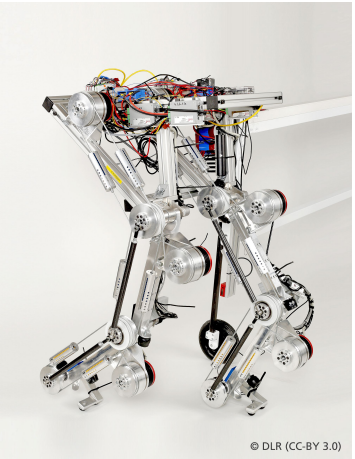


Figure 4.1: The DLR C-Runner

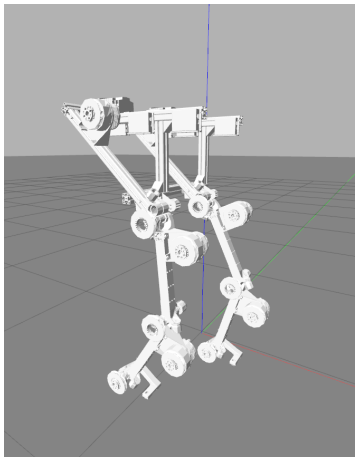


Figure 4.2: Simulation model of the DLR C-Runner

Figure 4.3: Setup for teleoperated walking with the DLR C-Runner in simulation

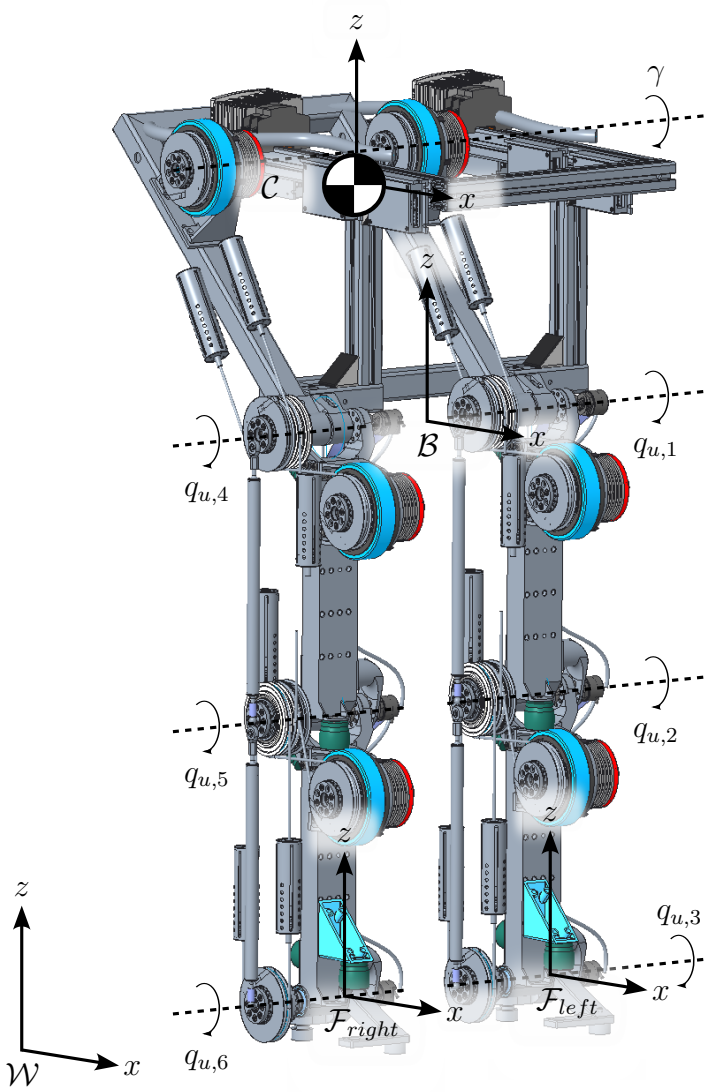


Figure 4.4: Kinematics of the DLR C-Runner

4.2 Input Processing and Control

The control of the C-Runner is similar to the hopper. However, the operator controls two legs $\boldsymbol{x}_{f,left}$ and $\boldsymbol{x}_{f,right}$, and the desired trajectory is realized on link-side. The corresponding joint positions q_1,q_2,q_4 and q_5 are calculated by saturating $\boldsymbol{x}_{d,o}$ and inverse kinematics (cf. equations (3.2) and (3.3)). Additionally, the ankles q_3 and q_6 are set to keep the foot parallel to the ground by calculating:

$$q_3 = -q_1 - q_2 - \gamma \ , \tag{4.1}$$

$$q_6 = -q_4 - q_5 - \gamma \ , \tag{4.2}$$

with γ denoting the rotation of the $\mathcal{B}$ relative to $\mathcal{W}$ about the axis depicted in Figure 4.4. The feature of motor overdrive, see equation (3.4), is omitted, as link-side positions are commanded.

The desired link-side trajectory $\boldsymbol{q}_{d,u}(t)$ is achieved by the Elastic Structure Preserving Impedance (ESπ) controller [35]. This controller shapes the link-side impedance of the robot. By introducing a coordinate transformation of the motor coordinates $\boldsymbol{q}_a$ into deflection coordinates $\bar{\boldsymbol{q}}_a$:

$$\bar{\boldsymbol{q}}_a = \boldsymbol{q}_a - \boldsymbol{K}^{-1}\,\boldsymbol{u}_u \ , \tag{4.3}$$

a new virtual control input on the link-side $\boldsymbol{u}_u$ is created, as depicted in Figure 4.5.

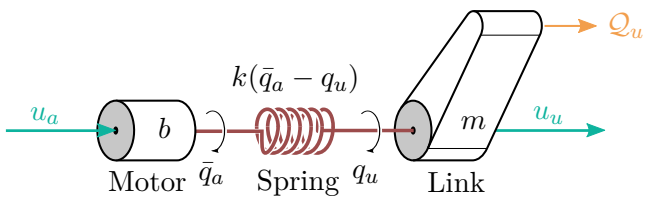


Figure 4.5: Virtual control inputs on a SEA

The control inputs are used to add a virtual spring and damper at link-side and a damper on motor-side by:

$$\boldsymbol{u}_u = -\boldsymbol{K}_{p,u}\,(\boldsymbol{q}_u - \boldsymbol{q}_{u,d}) - \boldsymbol{K}_{d,u}\,\dot{\boldsymbol{q}}_u \ , \tag{4.4}$$

$$\boldsymbol{u}_a = -\boldsymbol{K}_{d,a}\,\dot{\bar{\boldsymbol{q}}}_a \ . \tag{4.5}$$

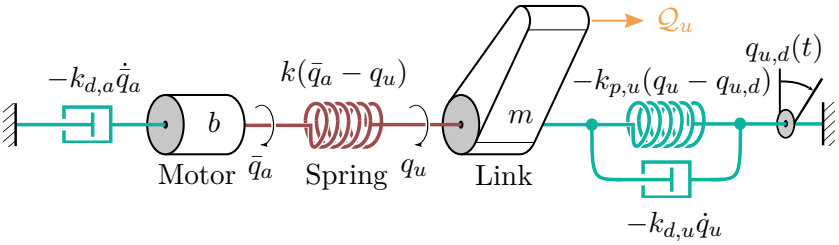


Figure 4.6: Principle sketch of a of a SEA controlled by the ESπ controller

The SEA is behaving like the principle sketch in Figure 4.6.

The positive, diagonal matrices $\boldsymbol{K}_{p,u}$ and $\boldsymbol{K}_{d,a}$ are $\in \mathbb{R}^{n\times n}$ for a robot with n joints. They shape the link-side stiffness and motor-side damping respectively. The links-side damping is set by $\boldsymbol{K}_{d,u}(\boldsymbol{q}_u) \in \mathbb{R}^{n\times n}$. It can be designed to be non-diagonal, configuration dependent and positive definite by considering the eigenvalues of $\boldsymbol{M}(\boldsymbol{q}_u)$ [38, Section 3.3]. Subsequently, the virtual control inputs are transformed back, to be utilized by the motors with the equation:

$$\boldsymbol{u} \;=\; \boldsymbol{u}_a + \boldsymbol{u}_u + \boldsymbol{B}\,\boldsymbol{K}^{-1}\,\ddot{\boldsymbol{u}}_u \;. \tag{4.6}$$

This control approach keeps the intrinsic elasticity of the system and shapes the systems dynamics minimally. Therefore, it has proven to be robust on highly compliant soft robots [39, 40]. Furthermore, it allows to shape the link-side stiffness directly, which is helpful for locomotion. The ESπ controller is implemented with the control parameters of Table 4.2.

Table 4.2: ESπ control parameters on the DLR C-Runner

Link-side stiffness $\boldsymbol{K}_{p,u}$:	$\texttt{diag}(600, 400, 80, 600, 400, 80)$	$\mathrm{N\,m\,rad^{-1}}$
Link-side damping $\boldsymbol{K}_{d,u}(\boldsymbol{q}_u)$:	Non-diagonal damping with a ratio of $\texttt{diag}(0.75, 0.5, 0.1, 0.75, 0.5, 0.1)$	$\mathrm{N\,m\,s\,rad^{-1}}$
Motor-side damping $\boldsymbol{K}_{d,a}$:	$0.6\,\sqrt{\boldsymbol{B}\,\boldsymbol{K}}$	$\mathrm{N\,m\,s\,rad^{-1}}$

4.3 Haptic Force Feedback

Teleoperated walking with an unlocked hip rotation γ is challenging because the robot is easily tipping over. Therefore, the haptic feedback is designed to support the operator in generating a stable gait, where the CoM does not leave the support polygon, i.e. the ground area, enclosed by the feet.

More precisely, the CoM is shifted above the forward foot in order to reduce the load of the backward foot before lifting it. This is achieved by introducing the distance s, which is the distance from $\boldsymbol{x}_f$ to the CoM in $\mathcal{B}_x$:

$$s \;=\; \mathcal{C}_{\mathcal{B}_x} - \texttt{MAX}\left(x_{f,\mathcal{B}_x,left},\; x_{f,\mathcal{B}_x,right}\right) \;. \tag{4.7}$$

The force feedback encourages the operator to position both feet to the left relative to $\mathcal{B}$, if the CoM is behind the forward foot. This shifts the base and therefore the CoM to the right. Figure 4.7 provides a graphical representation of s.

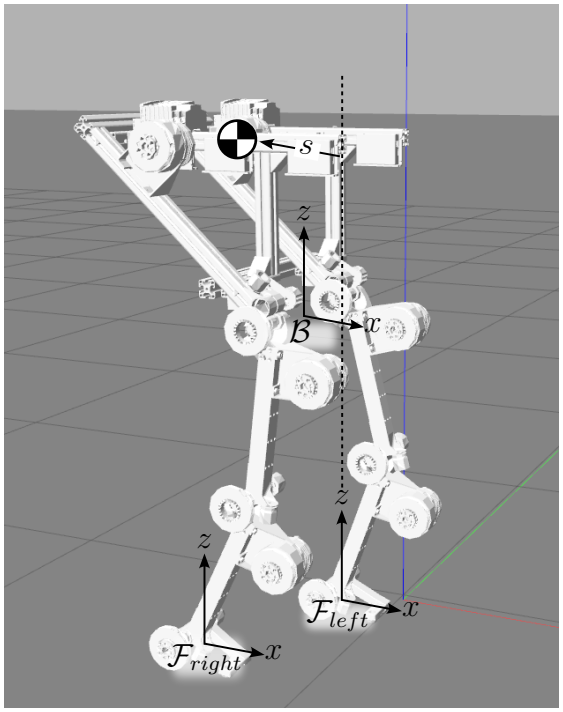


Figure 4.7: The distance s of from the forward foot to the CoM

Moreover, the operator gets an intuition on how much the feet are loaded, by using the vertical ground reaction force $f_{ext,\mathcal{F}_z}$ as force feedback in the z-axis of the

falcon haptic devices.[1] This enables the operator to prevent lifting a loaded foot, which causes tipping over. Further, a damping term is added for stability. Taken together, the equation for providing force feedback is:

$$\boldsymbol{f}_{\Omega,left} = \boldsymbol{K}_F \begin{pmatrix} s \\ f_{ext,\mathcal{F}_z,left} \end{pmatrix} - \boldsymbol{K}_D \, \dot{\boldsymbol{x}}_{d,o,left} \ , \tag{4.8}$$

$$\boldsymbol{f}_{\Omega,right} = \boldsymbol{K}_F \begin{pmatrix} s \\ f_{ext,\mathcal{F}_z,right} \end{pmatrix} - \boldsymbol{K}_D \, \dot{\boldsymbol{x}}_{d,o,right} \ . \tag{4.9}$$

4.4 Evaluation and Transferability to Hardware

Even though the force feedback helped stabilizing the robot with an unlocked hip rotation, not more than a few steps were possible without tipping over. However, fluent locomotion could be achieved with a locked hip. Figure 4.8 shows a series of snapshots during two steps.

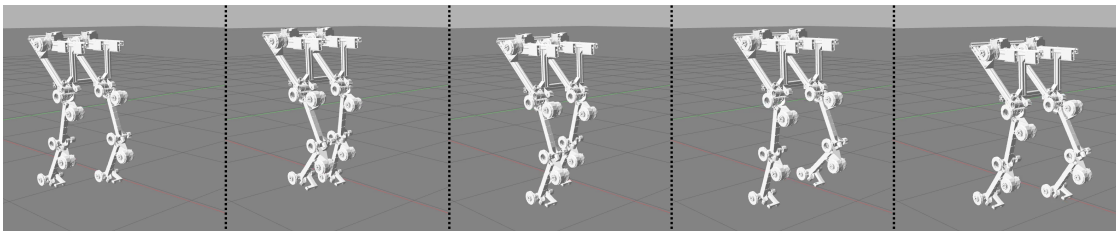


Figure 4.8: Slideshow of teleoperated walking on the DLR C-Runner

In addition, the trajectory of the base and feet in $\mathcal{W}$ is depicted in Figure 4.9. The wavelike motion of the base is identifiable, as well as the alternating steps.

One reason, why walking with a free hip failed, is the inaccuracy of the contact model of Gazebo. Accurate contact forces are hard to achieve in a real-time simulation. The ground friction is very low and the robot slips a lot on the floor. However, with some practice of the operator and tuning of control, input sensitivity and force feedback gains, a transferability on the real robot looks promising.

[1]These forces are measured at the real robot by a force sensor.

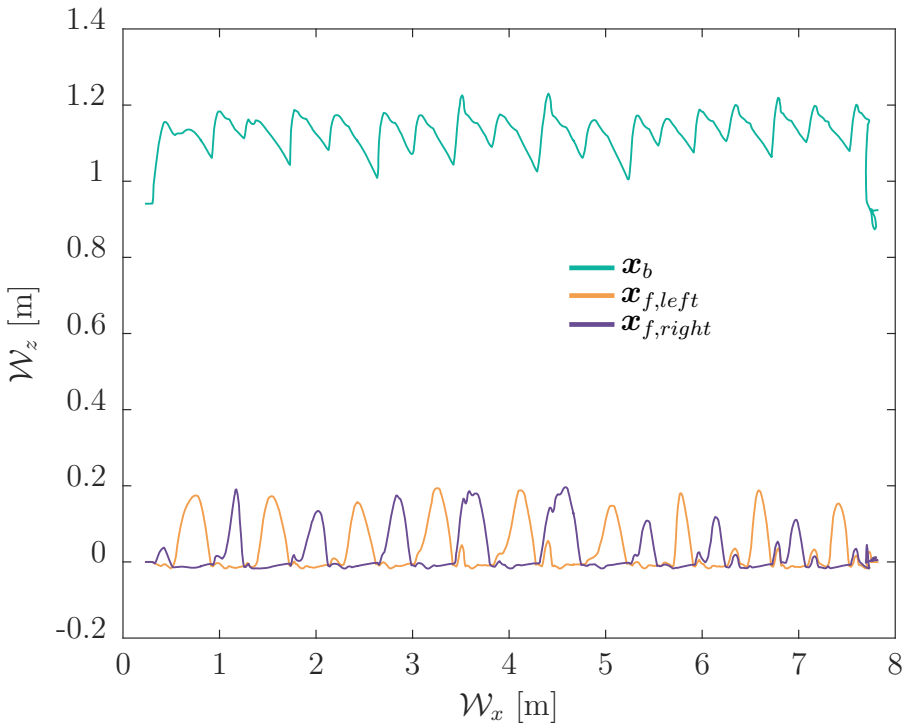


Figure 4.9: Base and foot trajectories during locomotion of the biped

5 Discussion and Outlook

The proposed teleoperation framework of controlling the hopper's foot tip while receiving the estimated external forces on the foot tip as feedback shows good performance. The generated trajectories achieve a CoT of 0.25 at $0.63\,\mathrm{m\,s^{-1}}$. This is a significant improvement over the SPEAR robot [4]. Despite being larger and heavier, this state-of-the-art robot is the most similar robot to the hopper, that has been found in literature. Figure 2.1 puts the hopper in relation to other legged robots.

Further room for improvement by BBO is slim. The energy efficiency, however, has been found to be significantly impacted by even small changes in the hardware. BBO on the hardware can be used to mitigate these effects. Despite an accurate simulation model, the sim to real transfer failed in both BBO and reinforcement learning. While the lateral movement could be fully stabilized in simulation by reinforcement learning, it only improved stability on the hardware. Variations in the initial conditions and policies, that favored harmful behavior to the hardware, have been the dominant challenges of applying reinforcement learning on the real robot. Last can be mitigated in the future, by manipulating the polar angle of the foot tip w.r.t. the base instead of adding an offset to the trajectory. That way, the policy cannot reduce foot tip clearance and provoke knee collisions. If the stabilization of lateral movement is successful on hardware, the policy can also consider forward velocity and CoT.

The teleoperation framework has successfully been applied to the biped DLR C-Runner in simulation. However, lateral stabilization must be improved in order to apply it on the real robot. In future research, the gaits of quadrupeds may be controlled with teleoperation by the use of virtual legs, where the legs operate in pairs. Contrary to bipeds, quadrupeds are less prone to tipping over.

References

[1] Marc H. Raibert. *Legged robots that balance*. The MIT Press series in artificial intelligence. MIT Press, Cambridge, Mass., 1986. ISBN 9780262181174.

[2] P. Gregorio, M. Ahmadi, and M. Buehler. Design, control, and energetics of an electrically actuated legged robot. *IEEE transactions on systems, man, and cybernetics. Part B, Cybernetics : a publication of the IEEE Systems, Man, and Cybernetics Society*, 27(4):626–634, 1997. ISSN 1083-4419. doi: 10.1109/3477.604106.

[3] M. Ahmadi and M. Buehler. The arl monopod ii running robot: control and energetics. In *Proceedings 1999 IEEE International Conference on Robotics and Automation (Cat. No.99CH36288C)*, volume 3, pages 1689–1694 vol.3, 1999. doi: 10.1109/ROBOT.1999.770352.

[4] Xin Liu, Anthony Rossi, and Ioannis Poulakakis. A switchable parallel elastic actuator and its application to leg design for running robots. *IEEE/ASME Transactions on Mechatronics*, 23(6):2681–2692, 2018. doi: 10.1109/TMECH.2018.2871670.

[5] Florian Loeffl, Alexander Werner, Dominic Lakatos, Jens Reinecke, Sebastian Wolf, Robert Burger, Thomas Gumpert, Florian Schmidt, Christian Ott, Markus Grebenstein, and Alin Albu-Schäffer. The dlr c-runner: Concept, design and experiments. In *2016 IEEE-RAS 16th International Conference on Humanoid Robots (Humanoids)*, pages 758–765. IEEE, 2016. ISBN 978-1-5090-4718-5. doi: 10.1109/HUMANOIDS.2016.7803359.

[6] Daniel Seidel, Milan Hermann, Thomas Gumpert, Florian C. Loeffl, and Alin Albu-Schäffer. Using elastically actuated legged robots in rough terrain:

Experiments with dlr quadruped bert. In *2020 IEEE Aerospace Conference*, pages 1–8, 2020. doi: 10.1109/AERO47225.2020.9172301.

[7] Johannes Englsberger, Christian Ott, and Alin Albu-Schäffer. Three-dimensional bipedal walking control based on divergent component of motion. *IEEE Transactions on Robotics*, 31(2):355–368, 2015. doi: 10.1109/TRO.2015.2405592.

[8] Alin Albu-Schäffer and Cosimo Della Santina. A review on nonlinear modes in conservative mechanical systems. *Annual Reviews in Control*, 50:49–71, 2020. ISSN 1367-5788. doi: 10.1016/j.arcontrol.2020.10.002. URL https://www.sciencedirect.com/science/article/pii/S1367578820300687.

[9] Alin Albu-Schäffer, Dominic Lakatos, and Stefano Stramigioli. Strict nonlinear normal modes of systems characterized by scalar functions on riemannian manifolds. *IEEE Robotics and Automation Letters*, 6(2):1910–1917, 2021. doi: 10.1109/LRA.2021.3061303.

[10] Dominic Lakatos, Gianluca Garofalo, Alexander Dietrich, and Alin Albu-Schäffer. Jumping control for compliantly actuated multilegged robots. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4562–4568. IEEE, 2014. ISBN 978-1-4799-3685-4. doi: 10.1109/ICRA.2014.6907525.

[11] Philipp Stratmann, Dominic Lakatos, Mehmet C. Ozparpucu, and Alin Albu-Schäffer. Legged elastic multibody systems: Adjusting limit cycles to close-to-optimal energy efficiency. *IEEE Robotics and Automation Letters*, 2(2): 436–443, 2017. doi: 10.1109/LRA.2016.2633580.

[12] Dominic Lakatos. *Multi-Dimensional Nonlinear Oscillation Control of Compliantly Actuated Robots*. PhD thesis, Technische Universität München, 2018. URL https://mediatum.ub.tum.de/1364441.

[13] Antonin Raffin, Daniel Seidel, Jens Kober, Alin Albu-Schäffer, João Silvério, and Freek Stulp. Learning to exploit elastic actuators for quadruped locomotion, 2022. URL http://arxiv.org/pdf/2209.07171v1.

[14] Dominic Lakatos, Florian Petit, and Alin Albu-Schäffer. Nonlinear oscillations for cyclic movements in human and robotic arms. *IEEE Transactions on Robotics*, 30(4):865–879, 2014. doi: 10.1109/TRO.2014.2308371.

[15] Denis Ćehajić, Pablo Budde gen. Dohmann, and Sandra Hirche. Estimating unknown object dynamics in human-robot manipulation tasks. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1730–1737, 2017. doi: 10.1109/ICRA.2017.7989204.

[16] Sangok Seok, Albert Wang, Meng Yee Chuah, David Otten, Jeffrey Lang, and Sangbae Kim. Design principles for highly efficient quadrupeds and implementation on the mit cheetah robot. In *2013 IEEE International Conference on Robotics and Automation*, pages 3307–3312, 2013. doi: 10.1109/ICRA.2013.6631038.

[17] G. Gabrielli and T. H. von Kármán. What price speed '?—-specific power required for propulsion of 'vehicles. *Mechanical Engineering*, 72(10), 1950.

[18] Xin Liu, Anthony Rossi, and Ioannis Poulakakis. Spear: A monopedal robot with switchable parallel elastic actuation. In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5142–5147, 2015. doi: 10.1109/IROS.2015.7354101.

[19] M. W. Spong. Modeling and control of elastic joint robots. *Journal of Dynamic Systems, Measurement, and Control*, 109(4):310–318, 1987. ISSN 0022-0434. doi: 10.1115/1.3143860.

[20] Bernd Henze. *Whole-Body Control for Multi-Contact Balancing of Humanoid Robots: Design and Experiments*, volume v.143 of *Springer Tracts in Advanced Robotics Ser.* Springer International Publishing AG, Cham, 2022. ISBN 978-3-030-87212-0. URL https://ebookcentral.proquest.com/lib/kxp/detail.action?docID=6798630.

[21] Jens Kober, Betty Mohler, and Jan Peters. Learning perceptual coupling for motor primitives. In *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 834–839, 2008. doi: 10.1109/IROS.2008.4650953.

[22] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[23] Welcome to spinning up in deep rl!, 2018. URL https://spinningup.openai.com/. Accessed: Feb. 10th, 2023.

[24] Antonin Raffin, Ashley Hill, Adam Gleave, Anssi Kanervisto, Maximilian Ernestus, and Noah Dormann. Stable-baselines3: Reliable reinforcement learning implementations. *Journal of Machine Learning Research*, 22(268): 1–8, 2021. URL http://jmlr.org/papers/v22/20-1364.html.

[25] Donald E Kirk. *Optimal control theory: an introduction*. Courier Corporation, 2004.

[26] David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. Deterministic policy gradient algorithms. In Eric P. Xing and Tony Jebara, editors, *Proceedings of the 31st International Conference on Machine Learning*, volume 32 of *Proceedings of Machine Learning Research*, pages 387–395, Bejing, China, 22–24 Jun 2014. PMLR. URL https://proceedings.mlr.press/v32/silver14.html.

[27] Boris Polyak and Anatoli Juditsky. Acceleration of stochastic approximation by averaging. *SIAM Journal on Control and Optimization*, 30:838–855, 07 1992. doi: 10.1137/0330046.

[28] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *CoRR*, abs/1801.01290, 2018. URL http://arxiv.org/abs/1801.01290.

[29] Leon Sievers, Johannes Pitz, and Berthold Bäuml. Learning purely tactile in-hand manipulation with a torque-controlled hand. In *2022 International Conference on Robotics and Automation (ICRA)*, pages 2745–2751, 2022. doi: 10.1109/ICRA46639.2022.9812093.

[30] Scott Fujimoto, Herke Hoof, and David Meger. Addressing function approximation error in actor-critic methods. In *International conference on machine learning*, pages 1587–1596. PMLR, 2018.

[31] Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. Optuna: A next-generation hyperparameter optimization framework. In *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 2623–2631, 2019.

[32] James Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. Algorithms for hyper-parameter optimization. In J. Shawe-Taylor, R. Zemel, P. Bartlett, F. Pereira, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 24. Curran Associates, Inc, 2011. URL https://proceedings.neurips.cc/paper/2011/file/86e8f7ab32cfd12577bc2619bc635690-Paper.pdf.

[33] N. Koenig and A. Howard. Design and use paradigms for gazebo, an open-source multi-robot simulator. In *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566)*, volume 3, pages 2149–2154 vol.3, 2004. doi: 10.1109/IROS.2004.1389727.

[34] Stefan Schaal, Jan Peters, Jun Nakanishi, and Auke Ijspeert. Control, planning, learning, and imitation with dynamic movement primitives. In *Workshop on Bilateral Paradigms on Humans and Humanoids: IEEE International Conference on Intelligent Robots and Systems (IROS 2003)*, pages 1–21, 2003.

[35] Manuel Keppler, Dominic Lakatos, Christian Ott, and Alin Albu-Schäffer. Elastic structure preserving impedance (espi) control for compliantly actuated robots. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5861–5868, 2018. doi: 10.1109/IROS.2018.8593415.

[36] James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, et al. Jax: Autograd and xla. *Astrophysics Source Code Library*, pages ascl–2111, 2021. URL http://github.com/google/jax.

[37] Steven Martin and Nick Hillier. Characterisation of the novint falcon haptic device for application as a robot manipulator. In *Australasian Conference on Robotics and Automation (ACRA)*, pages 291–292, 2009.

[38] Christian Ott. *Cartesian impedance control of redundant and flexible-joint robots*. Springer, 2008.

[39] Markus Grebenstein, Alin Albu-Schäffer, Thomas Bahls, Maxime Chalon, Oliver Eiberger, Werner Friedl, Robin Gruber, Sami Haddadin, Ulrich Hagn, Robert Haslinger, Hannes Höppner, Stefan Jörg, Mathias Nickl, Alexander Nothhelfer, Florian Petit, Josef Reill, Nikolaus Seitz, Thomas Wimböck, Sebastian Wolf, Tilo Wüsthoff, and Gerd Hirzinger. The dlr hand arm system. In *2011 IEEE International Conference on Robotics and Automation*, pages 3175–3182, 2011. doi: 10.1109/ICRA.2011.5980371.

[40] Manuel Keppler, Clara Raschel, David Wandinger, Andreas Stemmer, and Christian Ott. Robust stabilization of elastic joint robots by esp and pid control: Theory and experiments. *IEEE Robotics and Automation Letters*, 7 (3):8283–8290, 2022. doi: 10.1109/LRA.2022.3187277.

Appendix

A Inverse Kinematics and Saturation

```
%% Symbolic Invserse Kinematics Calculation
syms q1 q2 x z l1 l2

% forward kinematics for coupled kinematics
eq1 = x == l1*cos(q1) + l2*cos(q2);
eq2 = z == l1*sin(q1) + l2*sin(q2);

assume(l1,'positive')
assume(l2,'positive')
assume(x,'positive')
assume(z,'positive')

f = solve(eq1,eq2, q1,q2,'ReturnConditions',true)
eq_q1 = simplify(subs(f.q1,f.parameters(1:2),[0 0]))
eq_q2 = simplify(subs(f.q2,f.parameters(1:2),[0 0]))
```

```
function [q1, q2] = x2joint(x_in, link_length, configuration)
% generated from x2joint_sym.m
% input: x of TCP,  vector of link_lengths, desired configuration of knee
%        left or right

l1 = link_length(1); l2 = link_length(2);
x = x_in(1);     % see DOC for frame
z = x_in(2);

if sqrt(x^2+z^2)≥(l1+l2)
    error('x2joint is out of operation space! Function undefined!')
elseif x == 0 && z == 0
    error('x2joint is in sigurlarity!')
else
    temp_1 = (- l1^2 + 2*l1*l2 - l2^2 + x^2 + z^2)*(l1^2 + 2*l1*l2 + l2^2 ...
        - x^2 - z^2);
    temp_2 = (l1^2 + 2*l1*x - l2^2 + x^2 + z^2);
    temp_3 = (- l1^2 + l2^2 + 2*l2*x + x^2 + z^2);

    if configuration == 1
        q1   = 2*atan2((2*l1*z + (temp_1)^(1/2)),temp_2);
        q2   = 2*atan2((2*l2*z - (temp_1)^(1/2)),temp_3);
        % rotate for initial position (see DOC)
        q1 = q1 + pi/2;
        q2 = q2 + pi/2;
        % aviod discontinuity
        if q1 > pi
           q1 = q1 - 2*pi;
        end
    elseif configuration == 2
        q1 = 2*atan2((2*l1*z - (temp_1)^(1/2)),temp_2);
        q2 = 2*atan2((2*l2*z + (temp_1)^(1/2)),temp_3);
        % rotate for initial position (see DOC)
        q1 = q1 + pi/2;
        q2 = q2 + pi/2;
        % aviod discontinuity
        if q2 > pi
           q2 = q2 - 2*pi;
        end
    else
        error('select configuration 1 or 2')
    end
    % wrap to 2pi
    if q1 ≥ 2*pi
        q1 = q1 - 2*pi;
    end
    if q2 ≥ 2*pi
        q2 = q2 - 2*pi;
    end
end
```

```
function [x_sat, z_sat,alpha,r,r_sat,state] = ...
    sat_cartesian_to_op_space(x,z,r_saturation)
% saturate to op space (defined in polar coordinates)
% input: r_saturation = [lowerbound upperbound]
%        x and z according to {COM} frame (see DOC)

    % map to polar space
    r = (x^2 + z^2)^(1/2);

    % saturate radius to operational space
    r_sat = min(r_saturation(2), max(r_saturation(1), r));

    % if in saturation: state = 1
    if r_sat ≠ r
        state = 1;
    else
        state = 0;
    end

    % map back to cartesian space
    alpha = acos(x/r);
    if z<0
       alpha = -alpha;
    end
    z_sat = sin(alpha)*r_sat;
    x_sat = cos(alpha)*r_sat;

end
```

B Function Parameters of the Fitted Trajectory from Imitation

```
qda1 = General model Fourier4:

fit_s_mean_1(x) =  a0 + a1*cos(x*w) + b1*sin(x*w) +
            a2*cos(2*x*w) + b2*sin(2*x*w) + a3*cos(3*x*w)
            + b3*sin(3*x*w) + a4*cos(4*x*w) + b4*sin(4*x*w)
Coefficients (with 95% confidence bounds):
    a0 =      0.3905  (0.3897, 0.3913)
    a1 =     -0.1963  (-0.1971, -0.1955)
    b1 =     0.03698  (0.03547, 0.03848)
    a2 =    -0.07066  (-0.07111, -0.07021)
    b2 =   -0.005418  (-0.006441, -0.004395)
    a3 =    -0.03269  (-0.03309, -0.03229)
    b3 =   -0.004374  (-0.005178, -0.003569)
    a4 =    -0.01268  (-0.01314, -0.01221)
    b4 =   -0.003078  (-0.003636, -0.00252)
    w =       12.73  (12.7, 12.76)
```

```
qda2 = General model Fourier4:

fit_s_mean_2(x) =  a0 + a1*cos(x*w) + b1*sin(x*w) +
            a2*cos(2*x*w) + b2*sin(2*x*w) + a3*cos(3*x*w)
            + b3*sin(3*x*w) + a4*cos(4*x*w) + b4*sin(4*x*w)
Coefficients (with 95% confidence bounds):
    a0 =     -0.1428  (-0.1445, -0.1412)
    a1 =     -0.3334  (-0.3337, -0.3331)
    b1 =      0.0117  (0.005532, 0.01787)
    a2 =     0.07895  (0.07871, 0.0792)
    b2 =    -0.03212  (-0.03493, -0.02931)
    a3 =    -0.02601  (-0.027, -0.02501)
    b3 =     0.01959  (0.01788, 0.0213)
    a4 =    0.006339  (0.006025, 0.006652)
    b4 =   -0.007807  (-0.008493, -0.007121)
    w =       13.16  (13.08, 13.24)
```

C Code for Optimization with Optuna

```
import pickle as pkl
import time
import links_and_nodes as ln
import optuna
from optuna.samplers import TPESampler

N_TRIALS = 50
N_STARTUP_TRIALS = 5
SEED = 42

# Connect to ln
ln_client = ln.client("hoppy_optim")

# Initialize Opt Parameters
T =     ln_client.get_parameters("opt.T")
phase = ln_client.get_parameters("opt.phase")
d_1 =   ln_client.get_parameters("opt.d_1")
d_2 =   ln_client.get_parameters("opt.d_2")
A_1 =   ln_client.get_parameters("opt.A_1")
A_2 =   ln_client.get_parameters("opt.A_2")

# Initialize Episode Parameters
req_start = ln_client.get_parameters("opt.req_start")
n_0       = ln_client.get_parameters("opt.n_0")
n         = ln_client.get_parameters("opt.n")

study = optuna.create_study(
    sampler=TPESampler(n_startup_trials=N_STARTUP_TRIALS, ...
                                                multivariate=True, seed=SEED),
    storage=args.storage,
    study_name=args.study_name,
    load_if_exists=True,
    direction= "minimize",
)

try:
    for trial_idx in range(1, N_TRIALS + 1):

        # Initialize Trial
        trial = study.ask()

        # Set Test-Parameters
        T_param =   trial.suggest_float("T_param", 0.4, 0.6)
        phase_param = trial.suggest_float("phase_param", 0.0, 0.24)
        d_1_param = trial.suggest_float("d_1_param", -0.1, 0.1)
        d_2_param = trial.suggest_float("d_2_param", -0.1, 0.1)
```

```
        A_1_param = trial.suggest_float("A_1_param", 0.8, 1.2)
        A_2_param = trial.suggest_float("A_2_param", 0.8, 1.2)

        T.set_override("value", T_param)
        phase.set_override("value", phase_param)
        d_1.set_override("value", d_1_param)
        d_2.set_override("value", d_2_param)
        A_1.set_override("value", A_1_param)
        A_2.set_override("value", A_2_param)

        # Start Trial (trigger simulink chart)
        req_start.set_override("value", 1)
        time.sleep(0.1)
        req_start.set_override("value", 0)

        # Sleep time should be no lower than
        # transient+evaluation time
        #(faster for accelerated simulation)
        #T_sleep = n_0.value*T_param + n.value*T_param
        T_sleep = 1#(faster for accelerated simulation)
        time.sleep(T_sleep)

        # Evaluate Trial (trial stopps automatically, see Simulink)
        break_while = 1

        # Check for end_of_trial
        while break_while:
            end_of_trial = \
                ln_client.get_parameters("opt.end_of_trial")
            time.sleep(0.1)
            if end_of_trial.value == 1.0:
                # Reward
                CoT = \
                    ln_client.get_parameters("opt.CoT")
                CoT_v_des = \
                    ln_client.get_parameters("opt.CoT_v_des")
                distance_traveled = \
                    ln_client.get_parameters("opt.distance")
                velocity_mean = \
                    ln_client.get_parameters("opt.velocity")
                break_while = 0
            else:
                break_while = 1

        trial.set_user_attr("CoT", CoT.value)
        trial.set_user_attr("CoT_v_des", CoT_v_des.value)
        trial.set_user_attr("distance_traveled", \
            distance_traveled.value)
        trial.set_user_attr("velocity_mean", \
            velocity_mean.value)

```

```
        #reward = CoT.value
        reward = CoT_v_des.value
        study.tell(trial, reward)
        best_value = study.best_trial.value

# Disable robot for safety
except KeyboardInterrupt:
    pass

# Write report
study.trials_dataframe().to_csv("../logs/study_results.csv")
with open("../logs/study.pkl", "wb+") as f:
    pkl.dump(study, f)
```

D Code for Reinforcement Learning with stable-baselines3

```
import links_and_nodes as ln
import numpy as np
import time
import gym
from stable_baselines3 import SAC
from stable_baselines3.common.env_checker import check_env
from stable_baselines3.common.callbacks import CheckpointCallback

ln_client = ln.client("rl_client")
port_out = ln_client.publish("rl_cmd", "rl_cmd_from_python")
p_out = port_out.packet
port_in = ln_client.subscribe("rl_tele", "rl_tele_from_simulink")

class CustomEnv(gym.Env):
    def __init__(self):
        super().__init__()
        self.observation_space = \
            gym.spaces.Box(low=-np.inf, high=np.inf, shape=(18,), ...
                                                        dtype=np.float32)
        self.action_space = \
            gym.spaces.Box(low=-1, high=1, shape=(2,))

    def get_observation(self):
        telemetry = port_in.read()
        self.q_d = telemetry.q_d.copy()
        self.qu = telemetry.qu.copy()
        self.dqu = telemetry.dqu.copy()
        self.qa = telemetry.qa.copy()
        self.dqa = telemetry.dqa.copy()
        self.xb = telemetry.xb.copy()
        self.dxb = telemetry.dxb.copy()
        self.ddxb = telemetry.ddxb.copy()
        obs = np.concatenate(
            (
                self.q_d,
                self.qu,
                self.dqu,
                self.qa,
                self.dqa,
                self.xb,
                self.dxb,
                self.ddxb,
            )
        )
```

```
        terminate = telemetry.coll_flag
        reward = telemetry.reward_val
        return obs, terminate, reward

    def reset(self):
        # reset sim
        p_out.reset_from_rl = 1.0
        port_out.write()
        time.sleep(0.1)
        p_out.reset_from_rl = 0.0
        port_out.write()
        obs, terminate, reward = self.get_observation()
        while terminate == 1.0:
            time.sleep(0.01)
            obs, terminate, reward = self.get_observation()
        obs, terminate, reward = self.get_observation()
        return obs

    def step(self, action):
        p_out.q_d_rl = action
        port_out.write()
        obs, terminate, reward = self.get_observation()
        if terminate == 1.0:
            done = True
        else:
            done = False
        info = {}
        return obs, reward, done, info

check_env(CustomEnv(), warn=True)
env = CustomEnv()

# Save a checkpoint every 1000 steps
checkpoint_callback = CheckpointCallback(
  save_freq=10000,
  save_path="./logs/",
  name_prefix="sac_hoppy",
  save_replay_buffer=True,
  save_vecnormalize=True,
)
```

E Hyperparameters for SAC on Hardware

```yaml
HoppyRLEnv-v0: &droq
  env_wrapper:
    - rl_zoo3.wrappers.HistoryWrapper:
        horizon: 2
    - custom_envs.filter_wrappers.ActionFilterWrapper:
        sampling_rate: 66.6
        lowcut: 0
        highcut: 10

  n_timesteps: !!float 10e4
  policy: 'MlpPolicy'
  qf_learning_rate: !!float 1e-3
  train_freq: 1
  gradient_steps: 10
  policy_delay: 10
  learning_starts: 600
  # use_sde_at_warmup: True
  # use_sde: True
  # top_quantiles_to_drop_per_net: 2
  policy_kwargs: "dict(layer_norm=True, dropout_rate=0.01)"
```