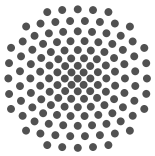


Universität Stuttgart

Vorgelegt an der Universität Stuttgart
Institut für Modellierung und Simulation Biomechanischer Systeme
Computational Biophysics and Biorobotics

Über die Regelung muskelgetriebener Systeme: ein hierarchischer und geometriebasierter Ansatz

—

On the control of muscle-actuated systems: a hierarchical and geometry-based approach

Vorgelegt von
Johannes Raphael Walter
aus Korb

Hauptberichter:
Prof. Dr. rer. nat. Syn Schmitt, Universität Stuttgart

Mitberichter:
Prof. Thor Besier, PhD, Universität Auckland

Tag der mündlichen Prüfung: 15. Dezember 2021

Von der *Fakultät 2: Bau- und Umweltingenieurswissenschaften* und dem *Stuttgarter Zentrum für Simulationswissenschaften* der Universität Stuttgart zur Erlangung der Würde eines Doktor-Ingenieurs (Dr.-Ing.) genehmigte Abhandlung

September 2022
⁂
CBB-001-2021

Selbständigkeitserklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbständig verfasst habe, dass ich keine anderen als die angegebenen Quellen benutzt und alle wörtlich oder sinngemäß aus anderen Werken übernommenen Aussagen als solche gekennzeichnet habe, dass die eingereichte Arbeit weder vollständig noch in wesentlichen Teilen Gegenstand eines anderen Prüfungsverfahrens gewesen ist und dass das elektronische Exemplar mit den anderen Exemplaren übereinstimmt. Teile der Arbeit wurden bereits vorab publiziert. Es handelt sich dabei um den hierarchischen Regelungsansatz, der bereits im Rahmen der Internationalen Graduiertenkollegs in der Öffentlichkeit diskutiert wurde und deshalb eine zeitnahe Publikation in einem Fachjournal notwendig war (Walter et al., 2021a,b).

Johannes Raphael Walter
Stuttgart
September 2022

Preface

I am thankful for all that brought me here, first and foremost for my family and friends who offer me the best in life.

Apart from that, I guess I can only bow humbly before the benevolent waves of fate; meeting the right people at the right time; mindlessly making right decisions; living in a world that has more wonders than explanations.

When enrolling at the University of Stuttgart I could have never imagined to leave it as a doctor. But soon my interest in mathematical models and the fascination of dynamical systems sparked. I have to thank the University of Stuttgart and all professors involved for providing the teaching of cybernetics and all my fellow students for making this a great time.

With rising expertise in the field of control—and a rising need to plug the whole in a student's wallet—I started a research assistant job in the wonderful world of biomechanics under the guidance of Prof. Dr. rer. nat. Syn Schmitt and Dipl.-Ing. Tille Karoline Rupp. The idea to apply my theoretical mathematical knowledge to such a most fascinating natural system felt like the right way—and it seems it was. Thanks to the ambition of Prof. Dr. rer. nat. Syn Schmitt I could continue my work in biomechanics to eventually start my PhD project under his guidance.

This PhD project was funded by the DFG as part of the International Research and Training Group 'Soft Tissue Robotics' (GRK 2198/1), which would not have been possible without the hard work of Prof. Oliver Röhrle. I therefore thank him and the DFG as an institution for the generous funding, but also the other professors, postdocs and students in Germany and New Zealand. Especially the Summer Schools that alternated between Stuttgart and Auckland allowed us students to fruitfully work together and to widen our own horizons in academia and across the globe.

As my work continued and the design of the hierarchical control architecture sprouted, the discussions with my professor Prof. Dr. rer. nat Syn Schmitt and with my colleagues Dr. rer. nat. Michael Günther, Dipl.-Gwl Simon Wolfen, Dipl.-Phys. Maria Hammer, M.Sc. Patrick Lerge and Dr. Daniel F. B. Häufle where of invaluable help for shaping the architectures current form. I appreciate Prof. Dr. rer. nat. Syn Schmitt for providing such a great working atmosphere in his research group for Computational Biophysics and Biorobotics—with such a great coffee.

Johannes Raphael Walter
Stuttgart
September 2022

Kurzfassung

Computersimulationen sind heutzutage eine leistungsfähige wissenschaftliche Methode um Hypothesen unter simulierten Bedingungen zu überprüfen. Dennoch scheinen biologische Bewegungen von mehrgelenkigen Systemen mit einer Vielzahl von Muskeln das Ergebnis von neuronalen Kommandos zu sein, die zu komplex sind um algorithmisch implementiert zu werden. Daher ist die Vielfalt, sowie die Komplexität von *in-silico* synthetisierten, muskelgetriebenen Bewegungen noch immer gering. Ein Schlüsselproblem zur Regelung biologischer Bewegung ist es eine Verbindung zwischen einer konzeptionellen Idee der Bewegung und der Bereitstellung von Muskelstimulationen herzustellen. Dies kann sich als schwierig erweisen, da in biologischen Bewegungen die Anzahl der Muskeln größer ist als die Dimension des konzeptionellen Raums der Bewegungsidee, bspw. der mechanischen Freiheitsgraden (FHG) des Skelettsystems.

In dieser Dissertation wird eine mathematische Formulierung einer hierarchischen Regelungsarchitektur vorgestellt, die eine solche Verbindung herstellt und die dazu ausgelegt ist eine Vielzahl von dreidimensionalen, muskelgetriebenen Bewegungen zu synthetisieren. Die Funktionsfähigkeit der Regelungsarchitektur ist anhand von verschiedenen menschlichen Bewegungsaufgaben demonstriert. Dies beinhaltet Simulationen von einem aufrechtem Stand, von einer Einstiegsbewegung in ein Fahrzeug, um ergonomische Rückschlüsse von einer virtuellen Designänderung zu ziehen, und von einem Sturz in eine Badewanne, um die Aufklärung eines Kriminalfalles zu unterstützen. Das zur Bewegungssynthese verwendete dreidimensionale digitale Menschmodell (DMM) besteht aus 20 Gelenk FHG und 36 Hill-Typ Muskel-Sehnen Einheiten (MSE). Das DMM ist erdähnlicher Gravitation ausgesetzt und die Füße interagieren mit dem Boden durch reversible Haft- und Gleitreibungskontakte. Die Regelungsarchitektur liefert kontinuierliche Stimulationen für alle MSE, basierend auf einer konzeptionellen Formulierung der Bewegungsaufgabe in den Koordinaten der Gelenkwinkel, der Gelenkmomente, der Positionen der Gliedmaßen oder in anderen konzeptionellen Koordinaten. Die Hierarchie der Regelungsarchitektur besteht aus drei Ebenen, der 'Konzeptionsebene', der 'Transformationsebene' und der 'Strukturebene'. In der 'Konzeptionsebene' wird die Bewegungsaufgabe in den konzeptionellen Koordinaten der Winkel, der Momente oder der Positionen formuliert und geregelt. Die Ausgangsgröße des konzeptionellen Reglers wird in einen Bewegungsplan für die Gelenkwinkel transformiert und bildet die Eingangsgröße für zwei Gelenkwinkelregler in der 'Transformationsebene'. Die 'Transformationsebene' kommuniziert mit den biologischen Strukturen in der 'Strukturebene', indem sie zum einen direkte Stimulationen für die MSE bereitstellt und zum anderen weitere Eingangssignale für strukturelle MSE Regler liefert. Dabei wird die Redundanz zwischen den MSE Stimulationen und den Gelenkwinkeln aufgelöst. Hierzu werden die Charakteristiken der modellierten biophysikalischen Strukturen, die Hebelarme der Muskeln, die Steifigkeitsverhältnisse innerhalb des Muskelmodells und die Längen-Stimulationsabhängigkeit der Aktivierungsdynamik, zu Nutze gemacht. Die von den MSE über ihre Hebelarme generierten Gelenkmomente beschleunigen die Körpersegmente und, indem die konzeptionellen Koordinaten an die Regler in der 'Konzeptionsebene' zurückgeführt werden, wird der hierarchische Regelkreis geschlossen. Die präsentierte Regelungsarchitektur erlaubt es damit eine konzeptionelle Bewegungsaufgabe direkt in Stimulationssignale der MSE zu übersetzen. Mit diesem Ansatz wird das Problem der Bewegungsplanung erleichtert, da bspw. nur das mechanische System in der konzeptionellen Planung betrachtet werden muss. Da zudem die Auflösung der Muskel-Gelenk-Redundanz nicht eindeutig ist, verbleibt zur Regelung eine 'ungeregelte Mannigfaltigkeit', mit der die Kokontraktion aller Muskeln an dem selben Gelenk genau so angepasst werden kann, dass sie nicht mit der Erfüllung der Bewegungsaufgabe in Konflikt steht.

Die Ergebnisse dieser Dissertation sind vielversprechend bezüglich der Anwendung der Regelungsarchitektur für die Synthese von dynamischen und komplexen muskelgetriebenen Bewegungen, auch für robotische Systeme die mit künstlichen Muskeln ausgestattet sind. Die internen Zustände des muskuloskelettalen Models sind zu weiterführenden Analysen geeignet, wie z.B. zur Evaluation der Ergonomie oder zur Abschätzung gesundheitlicher Auswirkungen der Bewegung.

Abstract

Simulation technologies have become a powerful tool in science for hypothesis testing, as more and more complex systems can be translated in a mathematical formulation that can be implemented in a simulation environment. Still, biological multi-joint, multi-muscle movements seem to be the outcome of neural commands too complex to be algorithmically represented. Therefore, the variety of muscle-driven movement tasks synthesised *in-silico* is still narrow, and so is their complexity. A key problem for biological motor control is to establish a link between a conceptional idea of a movement and the generation of a set of muscle-stimulating signals. This is particularly difficult as in biological motion the number of muscles is typically larger than the dimension of the conceptional space, e.g. of the body's mechanical degrees of freedom (DoFs).

A mathematical formulation that provides this link is presented in this dissertation in the form of a layered, hierarchical control architecture, which is meant to synthesise a wide range of complex 3-dimensional muscle-driven movements. The operativeness of the architecture is demonstrated by applying it to human movement tasks of different type, including human-like upright stance, the synthetisation of a car ingress motion to enable pre-production ergonomics evaluations and of a falling movement to assist in solving a criminal case. The 3-dimensional digital human model (DHM) deployed consists of 20 angular DoFs and 36 Hill-type muscle-tendon units (MTUs). The DHM is exposed to gravity while its feet contact the ground via reversible stick-slip interactions. The architecture continuously stimulates all MTUs based on a conceptional task formulation in terms of joint angles, joint torques, limb positions, or of other higher-level coordinates. It consists of the three layers, the 'conceptional layer', the 'transformational layer' and the 'structural layer'. In the 'conceptional layer', the movement task is formulated and controlled in the high-level coordinate spaces of angles, torques, or positions. The output of the high-level controller is transformed to provide a 'postural plan' that is fed to two mid-level joint controllers in the 'transformational layer'. The 'transformational layer' communicates with the biophysical structures in the 'structural layer' by providing, firstly, direct MTU stimulation contributions and, secondly, input signals for low-level MTU controllers. Thereby, the redundancy of the MTU stimulations with respect to the joint angles is resolved, a link is established, by exploiting some properties of the biophysical structures modelled, namely, the muscle moment arms, the stiffness relations within the muscle model, and the length-stimulation relation of the muscle activation dynamics. In the 'structural layer', each MTU is stimulated individually by a low-level length controller that models the mono-synaptic reflex, where the muscle-spindles' efferent outputs are fed back via α-neurons to innervate the whole muscle. The resulting joint torques that are generated by the MTUs via their moment arms accelerate the segments of the DHM and by feeding back the conceptional coordinates to the high-level controllers in the 'conceptional layer' the hierarchical feedback loop is closed.

The present control architecture, thus, allows the straightforward feeding of conceptional movement task formulations to MTUs. With this approach, the problem of movement planning is eased, as e.g. solely the mechanical system has to be considered in the conceptional plan. Additionally, as resolving the muscle-joint redundancy is not unique, additional DoFs on an 'uncontrolled manifold' can be exploited for control to synergistically adjust the co-contraction of all MTUs that act on the same joint while not interfering with the movement task. The results are promising regarding the application of the architecture to more dynamic and complex movements, with also the mechanical and muscular dimensionality enhanced to, e.g., implementing bi-articular muscles or examining diverse body plans (animal morphologies). It seems conceivable to deploy the architecture as part of a movement system that also incorporates nonlinear dynamics of biological sensors, or let it be part of learning processes and other sophisticated hypotheses on biological motor control.

Contents

List of Figures

List of Tables

List of Symbols

Sets and groups

$\mathbb{R}$	Set of real numbers.
$\mathbb{R}_1$	Set of real numbers, bounded by the intervall $[u_{\min}, 1]$, see (2.37).
$SO(3)$	Group of special orthogonal matrices, see (2.2).
$so(3)$	Vector space of all skew-symmetric matrices, see (2.5).
$SE(3)$	Group of special Euclidean matrices, see (2.12).
$se(3)$	Generalisation of $so(3)$, see (2.18).

Rigid body mechanics

$A, B, C, ...$	Rigid body labels are written in capital letters.
$\mathcal{A}, \mathcal{B}, \mathcal{C}, ...$	Coordinate frame labels are written in capital calligraphic letters.
$\boldsymbol{G}^{\mathcal{AB}} \in SE(3)$	Homogeneous representation of the rigid body transformation from $\mathcal{B}$ to $\mathcal{A}$, see (2.11).
$\boldsymbol{R}^{\mathcal{AB}} \in SO(3)$	Rotation matrix between the two frames $\mathcal{A}$ and $\mathcal{B}$.
$\boldsymbol{\omega}^{\mathcal{AB}} \in \mathbb{R}^3$, $\hat{\boldsymbol{\Omega}}^{\mathcal{AB}} \in so(3)$	Rotation axis or angular velocity component, see (2.5), (2.8) and (2.17).
$\boldsymbol{\psi}^{\mathcal{AB}}, \boldsymbol{\phi}^{\mathcal{AB}}$	Rotational coordinates, see (2.9).
$\boldsymbol{\chi}^{\mathcal{AB}}$	Coordinate representation of $\boldsymbol{G}$, see (2.16).
$\hat{\boldsymbol{V}}^{\mathcal{A}}_{\mathcal{AB}} \in se(3)$	Spatial velocity of frame $\mathcal{B}$, relative to frame $\mathcal{A}$ (subscripts), viewed from frame $\mathcal{A}$ (superscript) see (2.23).
$\boldsymbol{\xi}^{\mathcal{AB}} \in \mathbb{R}^6$, $\hat{\boldsymbol{\Xi}}^{\mathcal{AB}} \in SE(3)$	Coordinate representation and homogeneous representation of a twist, see (2.17).
$\hat{\boldsymbol{A}}^{\mathcal{AB}} \in \mathbb{R}^{3\times3}$	Homogeneous representation of rigid body acceleration, see (2.29).
$\hat{\boldsymbol{\mathcal{J}}}^{\mathcal{AB}}$, $\hat{\boldsymbol{\mathcal{P}}}^{\mathcal{AB}}$, $\hat{\boldsymbol{\Phi}}^{\mathcal{AB}} \in \mathbb{R}^{3\times3}$	Inertia, momentum and force matrices, see (2.32).
$\boldsymbol{q} \in \mathbb{R}^{n_{\text{DoF}}}$	Vector of generalised DoFs, see (2.33).
$\boldsymbol{\tau} \in \mathbb{R}^{n_{\text{DoF}}}$	Vector of generalised forces, see (2.33).

$\boldsymbol{\tau}^{\mathrm{MTU}}$, $\boldsymbol{\tau}^{\mathrm{lmt}}$ and $\boldsymbol{\tau}^{\mathrm{bsh}} \in \mathbb{R}^{n_{\mathrm{DoF}}}$	Vectors MTU, limitation and visco-elastic (bushing) joint torques, see (2.56), (2.57) and (2.58).

Muscle tendon units

$\gamma \in \mathbb{R}_1$	Concentration of Ca^{2+} ions in the sarcoplasma of a muscle, see (2.38).
$a \in \mathbb{R}_1$	Current activity of a MTU, see (2.39).
$u \in \mathbb{R}_1$	Current stimulation of a MTU, see (2.38).
l^{MTU}, l^{CE} and $l^{\mathrm{SEE}} \in \mathbb{R}$	Length of a MTU, CE and SEE, see (2.40).
f^{MTU}, f^{CE}, ... and $f^{\mathrm{SDE}} \in \mathbb{R}$	Forces of a MTU, CE, PEE, SEE and SDE, see (2.41).
R	Moment arm matrix, see (2.55).

Hierarchical control architecture

$\delta^{\mathrm{a/e}}_{\mathrm{SL}}$, $\delta^{\mathrm{a/e}}_{\mathrm{TL}}$ and $\delta^{\mathrm{a/e}}_{\mathrm{CL}}$	Vectors of afferent and efferent neural sensor delays of the structural, transforlational and coneptional layer, see (3.2).
$\boldsymbol{\lambda} \in \mathbb{R}^{n_{\mathrm{MTU}}}$	Vector of desired CE lengths, see (2.51) and (3.3).
$\boldsymbol{\theta} \in \mathbb{R}^{n_\theta}$ and $\boldsymbol{\theta}^{\mathrm{des}} \in \mathbb{R}^{n_\theta}$	Vectors of the 'MTU-actuated' joint angles and the 'postural plan' as a vector of desired joint angles, see (3.7).
$J^{\lambda\theta} \in \mathbb{R}^{n_{\mathrm{MTU}} \times n_\theta}$	Angle-length Jacobian matrix, see (3.9).
$J^{u\lambda} \in \mathbb{R}^{n_{\mathrm{MTU}} \times n_{\mathrm{MTU}}}$	Length-stimulation, see (3.26).
$J^{u\theta} \in \mathbb{R}^{n_{\mathrm{MTU}} \times n_\theta}$	Angle-stimulation Jacobian matrices, see (3.25).
$J^{\theta\tau} \in \mathbb{R}^{n_\theta \times n_\theta}$	Torque-angle Jacobian matrix, see (3.36).
$J^{\theta\chi} \in \mathbb{R}^{n_{\mathrm{MTU}} \times n_\theta}$	Position-angle Jacobian matrix, see (3.42).

List of Acronyms

Part I

Introduction and Preliminaries

Introduction 1

Animate nature is rife with intelligent design. This stretches from the growth of plants over locomotion of animals towards an individual human writing a doctoral dissertation about a concept that potentially describes some aspects of how his body is moving. A key feature of such animate systems is that they seem to violate the physics of an equivalent inanimate system, e.g. by growing roots or by jumping in the air. Of course, physical laws are pristine, as these animate systems are capable of exchanging energy with their environment, e.g. by absorbing matter or by the exertion forces. When control structures are present that regulate the type and amount of energy exchanged, an animate system is able to manipulate its dynamics to follow a desired plan. The control structures in nature emerged from evolution and thereby, the physical structure of an individual system is closely interweaved with the controlling functions of the organism. Terrestrial locomotion of animals, for example, is the outcome of the synergistic activation of a multitude of muscles, which themselves make up a big part of the animal's body. By this design, there exists no simple blueprint to study. Deciphering a natural system's function by its structure can be considered a big challenge for science, if not one of the biggest, when talking about the human brain. Different sub-genres of science have therefore emerged to face this challenge, each from their own perspective and with certain questions in mind. The desire to understand the principles of biological motion, in particular, has a long history that spreads across many fields of research in science and robotics (Wiener, 1948; Full and Koditschek, 1999; Holmes et al., 2006; Pratt and Pratt, 1998; Park, 2001), to aid improved medical care, to construct improved robotic systems, or for the sake of scientific curiosity. One of the most challenging tasks hereby is to decipher the organisation of control within the nervous system, as among others approached by Kiehn (2016), Tresch et al. (1999), and Herzfeld and Shadmehr (2014). For low-level structures in the spinal cord, including α- and γ-motor neurons, as well as for the muscles themselves, there is already strong biological evidence to have significant impact on movement control (Bizzi et al., 1992; van Soest and Bobbert, 1993; Hulliger et al., 1989; Brändle et al., 2020). Neglecting those structures in computational modelling studies can even more lead to misleading conclusions (Pinter et al., 2012). In the scenario of complex movements, higher centres of the *central nervous system* (CNS) take an important role in planning and execution of a movement task (Martin, 2005; Doya, 2000; Gao et al., 1996). For successful execution, internal models of the body's biophysical properties, including its neural wiring, and the external world, as needed, are present in any stable feedback control mechanism, be them explicitly learned or implicitly encoded in the structural embodiment (Wolpert and Kawato, 1998). A basic characteristic of muscle-driven, biological structures is the redundancy of neural commands to realise the kinematics of one specific movement

task in a space of fewer *degrees of freedom* (DoFs) (Wolpert, 1997; Bernstein, 1967). For this, selecting an appropriate combination of neural signals is traditionally perceived to be a tricky thing (Latash et al., 2002).

In this doctoral dissertation, an attempt is made to understand some principles of biological motor control. This is done by an abstract system description in terms of dynamical mathematical models and by the construction of a hierarchical control architecture that manipulates the system's mechanical DoFs, i.e. it controls the system's posture. Based on a conceptional plan, the control architecture eventually produces robust and stable muscle actuating signals, such that the muscle exerted forces accelerate the body segments to follow the conceptional plan. The biological feature of redundancy is thereby exploited for transforming the conceptional plan from the low-dimensional task space into the high-dimensional space of muscle stimulations. The muscle-joint redundancy even opens a manifold of control signals to fulfil several criteria at once, like maintaining a posture with different levels of co-contraction. The conceptional planning in the task space has the advantage of a reduced processing effort due to a lower number of controlled state variables to consider and control parameters to deal with. The subsequent transformation of the task-fulfilling signals into the high-dimensional space of muscle stimulations is, in the here presented control architecture of biological movement, achieved by geometric, Jacobian-based transformations between task space coordinates, e.g. joint angles or limb positions, and actuator variables, e.g. muscle lengths or stimulations (Pellionisz and Llinás, 1985)[1]. In the architecture presented, these transformations resolve the redundancy by knowledge about the structural characteristics, i.e. the body's morphology.

Using this principle, the hierarchical control architecture is designed with three layers of control: the conceptional layer, the transformational layer and the structural layer. In the structural layer, the relevant biophysical structures are modelled. This includes the bones, the muscles and a simple low-level spinal cord feedback mechanism that is known as the mono-synaptic reflex (Kistemaker et al. (2006); Bayer et al. (2017); Stollenmaier et al. (2020); Stollenmaier (2021)). Most of the dynamics from within the structural layer form the basis of the remaining control architecture. The transformational layer provides the input to the structural layer by means of muscle actuating signals and other inputs for the mono-synaptic reflex model. This input is based on a postural plan in terms of joint angles that is locally (mid-level) controlled and transformed into the high dimensional space of muscle stimulations. The conceptional layer produces such a postural plan as the result of yet another (high-level) control concept and a subsequent coordinate transformation. All geometric transformations are expressed as Jacobian matrices in closed-form and with this it becomes clear exactly which of the body's properties are exploited for control. The required information depends on the conceptional layer's coordinate space but always includes the muscle's moment arms and muscle-tendon stiffness relations in the transformational layer. By re-implementing this body knowledge in the control architecture, the morphological intelligence (Ghazi-Zahedi, 2019) of the closed loop system is increased.

The control architecture is used to synthesise various movement examples of a surrogate *digital human model* (DHM). Even though the DHM only has two muscles modelled for each of its angular DoFs, according to Schmitt et al. (2019), it is equipped with the most fundamental structures that are present in any biological, muscle-driven movement system. Beside demonstrating the isolated control of joint angles and limb positions, a torque controller synthesises upright standing of the DHM and the complex movements of a car-ingress and a fall into a bathtub are performed. The control architecture presented here, does not explain the complexity of its real biological pendant by far. However, some of the findings presented in this thesis may still serve as a starting point for further investigations. Even if

[1] Despite some critique on the mathematical denotations (Arbib and Amari, 1985) in the work of Pellionisz and Llinás (1985), their work is still a suitable tool in the computational analysis of movement organisation (Habas et al., 2020).

the biological validity of the method presented here is not given, it can already be used for the synthesis of various *in-Silico*, muscle-driven motions as a solution of forward dynamics simulations.

1.1 Structure of this doctoral dissertation

In Pt. I, this introduction is given and all preliminary model descriptions are presented. This includes an introduction to the theory of rigid body dynamics and a presentation of the used muscle model, its activation dynamics and a model of the mono-synaptic reflex.

In Pt. II, the hierarchical control architecture is introduced and mathematically designed. The three layers, the structural layer, the transformational layer and the conceptional layer are all derived in their mathematical formulations. It is explained how desired states of i) joint angles, ii) joint torques, iii) limb positions or iv) limb forces are transformed into a synergistic set of muscle actuating signals that lead to the body's movements towards the desired states.

In Pt. III, the DHM is presented and various simulations are performed to demonstrate the functionality of the control architecture. This includes simulations for desired joint angles of an arm, a leg and the trunk. Also of the position controlled movements of the foot and the hand and of an autonomous, torque-controlled upright standing of the DHM. Subsequently the two more complex and combined movements of a car-ingress and a falling motion are presented to give an impression of the versatility of the architecture.

In Pt. IV, the work of this dissertation is critically discussed and an optimistic outlook for the architecture's future steps is given.

Preliminary mathematical model descriptions 2

To perform forward dynamics simulations of biophysical systems, mathematical models are required as a basis for algorithmic implementations. For this, it is not only the mathematical models of certain subsystems that are of interest but also the exchange of information between those subsystems. In computational modelling, exchange of information thereby describes the communication of the subsystems and represents biophysical processes, such as neural transmission of electrical signals along axons and transaction of muscle forces along tendons to bones. The control structure, in particular, forms a closed loop feedback unit, which interacts with the biomechanical structures of the body through sensor signals and muscle stimulations. Therefore, it is conceivable that, to produce coordinated movements, the controller benefits of having embedded some information about the body's peripheral structures and the properties of the muscles.

In this chapter, basic mathematical descriptions of the subsystems are given that are used later in the design of the control architecture in Pt. II. Therefore in Sec. 2.1 the dynamics of rigid body systems are investigated in a mostly general way. Rigid body dynamics are used mainly for the control of positions in Sec. 3.3.3 and additionally provides the reader with knowledge of the general description of the skeletal system. In Sec. 2.2 the muscle model that is used for the actuation of the skeletal system is introduced. Secs. 2.3 and 2.4 introduce the force laws of joint limitations, of other tissue forces and of contacts with the environment. An overview of the subsystems and their interconnections is displayed in Fig. 2.1.

2.1 Rigid body dynamics

In this section, the mathematical notation of rigid body dynamics is introduced. Rigid bodies, linked to each other, form the skeletal structure of the biomechanical models that are used later. By forces that are applied to the bones, the rigid bodies exchange energy with their surroundings and are accelerated. The forces, e.g. provided by muscles or by contacts with the environment, form the input to the rigid body system. The movement of the bones can then be described by differential equations that capture the development of the internal system states by enforcing the laws of physics (inertia, gravitation, Coriolis,...). For this, a *Lagrangian* formulation of the equations of motion is used that has the relative DoFs (joint angles) as system states. The forces of, e.g. the muscles, are therefore calculated to be equivalent and generalized forces (joint torques) of the DoFs. As these equations of motions can become tremendously complex for systems of higher orders, an algorithmic implementation that automatically generates the equations of motion can be deployed as,

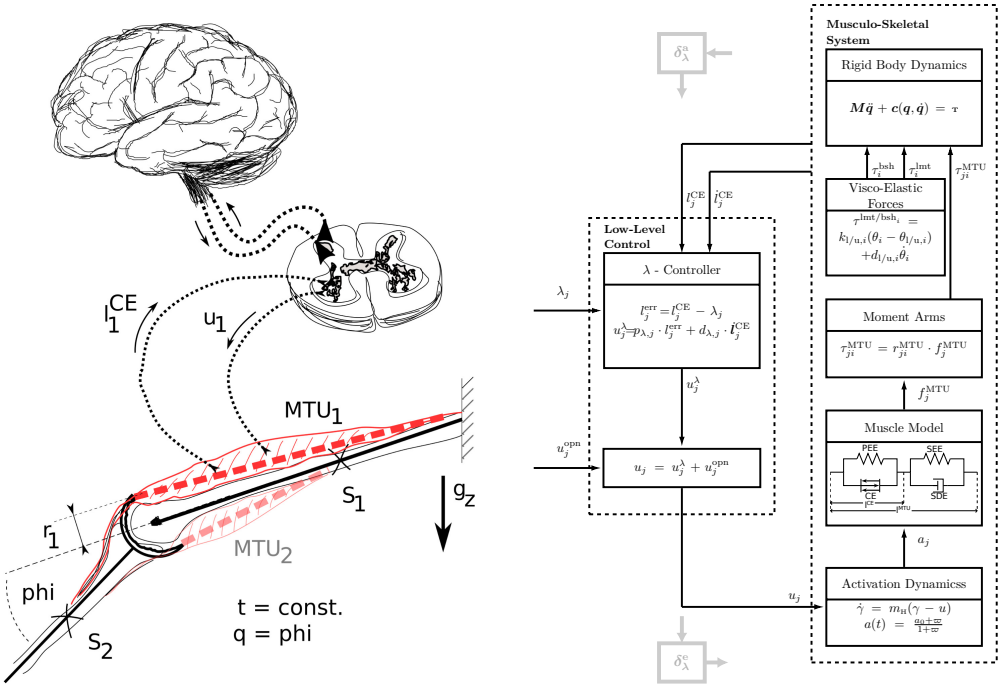


Figure 2.1: Left: An *elementary biological drive* includes the minimum structural elements required to study biological movement from a systems biophysics perspective (Schmitt et al., 2019). The body segments are connected by a joint and the muscles exert forces that move the joint's DoF. The muscles are commanded by synergistic control signals u_j that originate from the spinal cord and from higher centres of the CNS. Proprioceptive sensory organs from within the muscles feed back signals to the spinal cord, where low-level, neural connections, i.e. the monosynaptic-reflex, closes the reflex loop by contributing to u_j.
Right: Schematic block diagram that models the biophysical process of movement generation. The muscles are modelled as *muscle-tendon units* (MTUs) (Haeufle et al., 2014b; Günther et al., 2018; Hammer et al., 2019) and their resulting joint torques feed the *equations of motion* of the mechanical skeletal system. The monosynaptic-reflex is modelled by means of a *proportional-derivative* (PD) controller of the muscle fibre length of a MTU (Bayer et al., 2017; Kistemaker et al., 2006; Günther and Ruder, 2003). The inputs that control the movement of the musculo-skeletal system are values of MTU stimulations u_j^{opn} and of desired fibre lengths λ_j for $j = 1...n_{\text{MTU}}$.

e.g. described by Legnani et al. (1996b,a). This method utilizes the sophisticated notion of screw motions as a basis, which consists of a rotational motion around an axis plus a translational motion parallel to that axis.

The goal of this section is to have a closed formulation of the *Lagrangian* equations of motion that describes the mechanical movement of an open-chain, ramified skeletal system. Such an open-chain, ramified system consists of a base body, i.e. the pelvis body, to which other bodies are linked via joints in a ramified way without containing closed kinematic chains. In Sec. 2.1.1 some of the basic notations of rigid bodies and coordinate frames are given. In Sec. 2.1.3, coordinate transformations in between these frames are introduced in a *homogeneous representation* that makes use of 4×4 matrices (Murray et al., 1994; Legnani et al., 1996b,b; Hartenberg and Denavit, 1955). The use of 4×4 matrices simultaneously takes account of translations and rotations in between different coordinate frames. In Sec. 2.1.2 a representation of rotations that uses *exponential coordinates* is introduced that describes each arbitrary rotation in 3D space by a rotation around a single rotation axis plus an angle. In Appendix A.1 further approaches for describing rotations by *quaternions* or *Euler/Cardan conventions* are introduced (Sarabandi and Thomas, 2019; Wu et al., 2002, 2005; Sinclair et al., 2012). With the use of rigid body transformations in their homogeneous representation, a screw motion is described in Sec. 2.1.4 and it's velocity is derived in Sec. 2.1.5. By additionally providing transformations and compositions of rigid body acceleration and of other dynamical features (in Sec. 2.1.6), the equations of motion for the full skeletal system are derived in Sec. 2.1.7.

2.1.1 Rigid bodies, frames and points

A rigid body is defined by the property that it cannot be deformed. As a consequence, the distances between all points on the body remain constant during all movements and for all forces that are applied to the body. Each point in space (attached to a rigid body) and thereby the whole movement of the body must be described from a specific frame that is either attached to a moving body or to an inertial world frame. The movement of the whole body can thereby be described by an equivalent movement of the body's *center of mass* (COM), when considering the body's mass and inertia properties.

To clarify the notation, rigid bodies are written by capital letters, i.e. by A, B, or C. The *inertial* 'world' body is called W. On each rigid body, multiple frames may be defined, which are written by capital calligraphic letters, i.e. $\mathcal{A}_i$ is the i-th frame fixed to the rigid body A; if no index i is given, the frame is located at the body's COM. The inertial 'world' frame is called $\mathcal{W}$ The position of a point $\boldsymbol{p} \in \mathbb{R}^3$ that is fixed to a rigid body can only be described from a respective frame. For example, the symbol $\boldsymbol{p}_A^{\mathcal{A}}$ describes the 3D *coordinates* $\boldsymbol{p}_A^{\mathcal{A}} = \begin{bmatrix} x_{p_A}^{\mathcal{A}} & y_{p_A}^{\mathcal{A}} & z_{p_A}^{\mathcal{A}} \end{bmatrix}^T \in \mathbb{R}^3$ of a point that is fixed to the body A (subscript), as viewed from the frame $\mathcal{A}$ (superscript). The symbol $\boldsymbol{p}_A^{\mathcal{B}}$ describes the position of the same point, viewed from the frame $\mathcal{B}$. In 3D-space, different frames may have a *translational offset* and a different *orientation*. The translation of $\mathcal{B}$ in $\mathcal{A}$ is described by the point $\boldsymbol{t}^{\mathcal{AB}} \in \mathbb{R}^3$ of the origin of $\mathcal{B}$ viewed from $\mathcal{A}$. The rotation is described by a rotation matrix $\boldsymbol{R}^{\mathcal{AB}} \in SO(3) \subset \mathbb{R}^{3\times3}$.

2.1.2 3D rotations, exponential coordinates and helical angles

There are several ways of representing the rotation in 3D-space of a frame relative to another. Following the *Euler/Cardan convention* (Henze, 2002; Murray et al., 1994), a rotation in 3D space can be described by three individual rotations around the Cartesian coordinate axes. Hereby three parameters are used for the parametrisation of the rotation, namely the rotation angles around the chosen axes. Using only three parameters for rotations is prone to so-called *Gimbal-locks*, i.e. mathematical singularities at which the rotation cannot be well-described. Even if such singularities can never be eliminated with only

three parameters (Murray et al. (1994)), they may be detected and avoided (Henze (2002)). Other representations of rotations use four parameters, for example *quaternions* (Sarabandi and Thomas, 2019), or the representation by *exponential coordinates* (Murray et al., 1994). While more details on the Euler/Cardan convention and quaternions are given in Appendix A.1, the representation by exponential coordinates is introduced in the following.

Mathematically, a rotation that transforms the coordinates of a point $\boldsymbol{p}_B^{\mathcal{B}}$ to the coordinates of the frame $\mathcal{A}$ can be described by a rotation matrix

$$\boldsymbol{R}^{\mathcal{AB}} \in SO(3) \subset \mathbb{R}^{3\times 3}. \tag{2.1}$$

The coordinates of $\boldsymbol{p}_B^{\mathcal{A}}$ are simply given by the matrix-vector product

$$\boldsymbol{p}_B^{\mathcal{A}} = \boldsymbol{R}^{\mathcal{AB}} \cdot \boldsymbol{p}_B^{\mathcal{B}}.$$

The set $SO(3)$ is a subset of $\mathbb{R}^{3\times 3}$ and forms the multiplicative group of *special orthogonal* matrices. It contains all orthogonal matrices $\boldsymbol{R}^T = \boldsymbol{R}^{-1}$ with $\det(\boldsymbol{R}) = +1$:

$$SO(3) := \left\{\boldsymbol{R} \in \mathbb{R}^{3\times 3} : \boldsymbol{R}\boldsymbol{R}^T = \boldsymbol{I}_3,\, \det \boldsymbol{R} = +1\right\} \subset \mathbb{R}^{3\times 3} \tag{2.2}$$

All valid rotation matrices $\boldsymbol{R}$ belong to this group (Murray et al., 1994). The inverse rotation is applied by a transposed rotation matrix, and additionally, the composition rule applies (Murray et al., 1994), i.e.

$$\boldsymbol{R}^{\mathcal{BA}} = \boldsymbol{R}^{\mathcal{AB}^T} \qquad \boldsymbol{R}^{\mathcal{AC}} = \boldsymbol{R}^{\mathcal{AB}} \cdot \boldsymbol{R}^{\mathcal{BC}}.$$

The representation of exponential coordinates describes each rotation around exactly one axis $\boldsymbol{\omega} \in \mathbb{R}^3$ along the scalar angle $\theta \in \mathbb{R}$. Although mathematically complex at first glance, this representation is closely connected to the notion of screw motions, which is given later. The basic idea that lies behind the representation of a single axis and angle is formulated in Euler's theorem, which is stated here. Its proof can be found in Murray et al. (1994).

Theorem 2.1.1 (Euler) *Any rotation $\boldsymbol{R} \in SO(3)$ is equivalent to a rotation about a fixed axis $\boldsymbol{\omega} \in \mathbb{R}^3$ through an angle $\theta \in [0, 2\pi)$.*

This means, that for each rotation in 3D-space a single axis $\boldsymbol{\omega} \in \mathbb{R}^3$ can be found together with a scalar rotation angle $\theta \in [0, 2\pi)$ that executes the rotation. This is a first step towards describing any rigid body motion by a screw motion, which consists of a rotational motion around an axis plus a translational motion parallel to that axis. Therefore in this section, the notation for 'screw-like' rotations are laid out and introduced. This includes the notion of *exponential coordinates* for rotations and the corresponding representation of a rotation matrix $\boldsymbol{R} \in \mathrm{SO}(3)$ using a matrix exponential. Later, in Sec. 2.1.4, these findings will be generalised to also include the translational part of a rigid body transformation $G \in \mathrm{SE}(3)$ for a complete screw motion.

For deriving a rotation matrix $\boldsymbol{R} \in \mathrm{SO}(3)$ based on a single axis $\boldsymbol{\omega} \in \mathbb{R}^3$ and angle $\theta \in \mathbb{R}$ a *thought experiment* of a dynamic rotational movement is performed. Therefore, assume $\boldsymbol{p}$ to be a point on a body that rotates about the (fixed) axis $\boldsymbol{\omega}$ with *constant unit velocity*. The velocity $\boldsymbol{\nu}_p = \dot{\boldsymbol{p}} \in \mathbb{R}^3$ of the point $\boldsymbol{p}$ then is

$$\begin{aligned} \dot{\boldsymbol{p}}(t) &= \boldsymbol{\omega} \times \boldsymbol{p}(t) \\ &= \hat{\boldsymbol{\Omega}}\boldsymbol{p}(t), \end{aligned} \tag{2.3}$$

where $\hat{\boldsymbol{\Omega}} \in \mathbb{R}^{3\times 3}$ is a matrix that corresponds to the axis $\boldsymbol{\omega} \in \mathbb{R}^3$, which will be clarified shortly below in (2.5). The general solution of the differential equation (2.3) is obtained by the initial condition $\boldsymbol{p}(t = 0)$ and the matrix exponential of the angular velocity matrix

$$\boldsymbol{p}(t) = e^{\hat{\boldsymbol{\Omega}}\boldsymbol{t}}\boldsymbol{p}(0).$$

The matrix exponential $e^{\hat{\boldsymbol{\Omega}}\boldsymbol{t}}$ thus acts as a rotation matrix from $\boldsymbol{p}(0)$ to $\boldsymbol{p}(t)$. When rotating with unit velocity, i.e. $|\boldsymbol{\omega}| = 1$, for an amount of $t = \theta$ units of time, the respective rotation matrix is

$$\boldsymbol{R}(\boldsymbol{\omega}, \theta) = e^{\hat{\boldsymbol{\Omega}}\boldsymbol{\theta}}. \tag{2.4}$$

The matrix $\hat{\boldsymbol{\Omega}}$ is thereby the skew-symmetric matrix of the axis vector $\boldsymbol{\omega}$ that is obtained by the the $\wedge$-operator (wedge). The $\wedge$-operator transforms a vector $\boldsymbol{\omega} = \begin{bmatrix}\omega_1 & \omega_2 & \omega_3\end{bmatrix} \in \mathbb{R}^3$ into a skew symmetric matrix $(\boldsymbol{\omega})^\wedge = \hat{\boldsymbol{\Omega}} \in so(3) \subset \mathbb{R}^{3\times 3}$ by

$$(\boldsymbol{\omega})^\wedge = \hat{\boldsymbol{\Omega}} := \begin{bmatrix} 0 & -\omega_3 & \omega_2 \\ \omega_3 & 0 & -\omega_1 \\ -\omega_2 & \omega_1 & 0 \end{bmatrix} \in so(3).$$

The vector space $so(3)$ contains all 3×3 skew-symmetric matrices:

$$so(3) = \{\boldsymbol{A} \in \mathbb{R}^{3\times 3} : \boldsymbol{A}^T = -\boldsymbol{A}\}. \tag{2.5}$$

The inverse of this operator is called the $\vee$-operator (vee) and it transforms a skew-symmetric matrix from $so(3)$ into a vector in $\mathbb{R}^3$:

$$(\hat{\boldsymbol{\Omega}})^\vee = \boldsymbol{\omega} \in \mathbb{R}^3.$$

A skew symmetric matrix is closely related to the cross product, as already foretold in (2.3). This can be easily seen by calculating

$$\boldsymbol{a} \times \boldsymbol{b} = (\boldsymbol{a})^\wedge \cdot \boldsymbol{b}, \quad \text{for} \quad a,\, b \in \mathbb{R}^3.$$

The somewhat unintuitive notion of a matrix exponential as a rotation matrix in (2.4) is bolstered with a straightforward and feasible way of calculation. While details on the derivation are omitted here (see Murray et al. (1994) Chpt. 2.2 for details), the resulting equation, which is also known as *Rodrigues' formula*, is directly given:

$$\boldsymbol{R} = \boldsymbol{e}^{\hat{\boldsymbol{\Omega}}\theta} = \boldsymbol{I}_3 + \hat{\boldsymbol{\Omega}}\sin(\theta) + \hat{\boldsymbol{\Omega}}^2(1 - \cos(\theta)), \tag{2.6}$$

with $\hat{\boldsymbol{\Omega}}^2 = \boldsymbol{\omega}\boldsymbol{\omega}^T - \|\boldsymbol{\omega}\|^2\boldsymbol{I}_3$.

To obtain the axis $\boldsymbol{\omega} \in \mathbb{R}^3$ and the rotation angle $\theta \in \mathbb{R}$ from a rotation matrix $\boldsymbol{R} \in SO(3)$, on the other hand, the rotation matrix $\boldsymbol{R}$ is calculated according to (2.6) and further analysed by investigating its trace and the off-diagonal elements $r_{i,j}$ (Murray et al., 1994). The rotation angle $\theta \in \mathbb{R}$ is then given by

$$\theta = \cos^{-1}\left(\frac{\text{trace}(\boldsymbol{R}) - 1}{2}\right) \tag{2.7}$$

and the axis $\boldsymbol{\omega} \in \mathbb{R}^3$ for $\theta \neq 0$ is

$$\boldsymbol{\omega} = \frac{1}{2 \cdot \sin(\theta)} \begin{bmatrix} r_{32} - r_{23} \\ r_{13} - r_{31} \\ r_{21} - r_{12} \end{bmatrix}. \tag{2.8}$$

The components of the vector $\boldsymbol{\psi} := \theta\boldsymbol{\omega} \in \mathbb{R}^3$ given by equations (2.8) and (2.7) are called the *exponential coordinates*, also known as *helical-angles*. In the later chapters of this thesis, namely for the position control of a limb, the angular coordinates given by

$$\boldsymbol{\phi} := \frac{1}{2} \begin{bmatrix} r_{32} - r_{23} \\ r_{13} - r_{31} \\ r_{21} - r_{12} \end{bmatrix} \tag{2.9}$$

are used (Ibuki et al., 2014). Note, that the rotation angle θ is ambiguous for $\theta \pm 2\pi n$ and the axis $\boldsymbol{\omega}$ is not well defined for $\theta = 0$, i.e. there is a *singularity* for $\boldsymbol{R} = \boldsymbol{I}_3$. Using the vector $\theta\boldsymbol{\phi}$ avoids this singularity at the cost of losing mathematical accuracy.

Using exponential coordinates for rotation parametrisation angles in biomechanical studies has the advantages of stable mathematics that avoid a *Gimbal-lock*, which is known from other representations. On the other hand, the representation of rotations using an *Euler/-Cardan convention* may be the most straightforward to understand and are considered to have the most intuitive relation to physiological angles in biomechanical studies. This may also be why the *International Society of Biomechanics* (ISB) recommends the usage of *Euler* or *Cardan* angles, although in 3D not even the clinical definitions are consistent (Sarabandi and Thomas, 2019; Wu et al., 2002, 2005; Sinclair et al., 2012). *Quaternions* are closely connected to the exponential coordinates, as both can be converted into the other. There are different ways to define the quaternions, though, of which two are given in Appendix A.1 (Sarabandi and Thomas, 2019; Sinclair et al., 2012).

2.1.3 Homogeneous representation of rigid body transformations

A rigid body transformation is used to change the reference frame in which points are described. Given the point $\boldsymbol{p}_B^{\mathcal{B}}$ on body B in the frame $\mathcal{B}$, it is described in the frame $\mathcal{A}$ by

$$\boldsymbol{p}_B^{\mathcal{A}} = \boldsymbol{R}^{\mathcal{AB}} \cdot \boldsymbol{p}_B^{\mathcal{B}} + \boldsymbol{t}^{\mathcal{AB}}.$$

The original point is rotated into the coordinates of $\mathcal{A}$ by the rotation matrix $\boldsymbol{R}^{\mathcal{AB}} \in SO(3)$ (see (2.1)) and the translational offset $\boldsymbol{t}^{\mathcal{AB}} \in \mathbb{R}^3$ is added. The same result is obtained from the matrix equation

$$\left[\begin{array}{c} \boldsymbol{p}_B^{\mathcal{A}} \\ \hline 1 \end{array}\right] = \underbrace{\left[\begin{array}{ccc|c} & \boldsymbol{R}^{\mathcal{AB}} & & \boldsymbol{t}^{\mathcal{AB}} \\ \hline 0 & 0 & 0 & 1 \end{array}\right]}_{:=\boldsymbol{G}^{\mathcal{AB}}} \cdot \left[\begin{array}{c} \boldsymbol{p}_B^{\mathcal{B}} \\ \hline 1 \end{array}\right]. \tag{2.10}$$

Exactly this 4×4 matrix $\boldsymbol{G}^{\mathcal{AB}}$ is called the *homogeneous representation* of the rigid body transformation that relates the frames $\mathcal{A}$ and $\mathcal{B}$ (Murray et al., 1994; Legnani et al., 1996b,b; Hartenberg and Denavit, 1955):

$$\boldsymbol{G}^{\mathcal{AB}} := \left[\begin{array}{ccc|c} & \boldsymbol{R}^{\mathcal{AB}} & & \boldsymbol{t}^{\mathcal{AB}} \\ \hline 0 & 0 & 0 & 1 \end{array}\right] \in SE(3) \subset \mathbb{R}^{4\times 4}. \tag{2.11}$$

It belongs to the *special Euclidean* group, defined by

$$SE(3) := \mathbb{R}^3 \times SO(3), \tag{2.12}$$

with $SO(3)$ defined in (2.2). All valid homogeneous representations $\boldsymbol{G}$ of rigid body transformations belong to this group. The use of such a homogeneous representation requires to redefine the notion of a point by appending a 1 in the fourth dimension. Throughout this thesis, for such an appended number no special symbol is introduced, as it should be apparent from the context, whether the calculation requires 3 or 4 dimension. The transformation of the reference frame of a point $\boldsymbol{p}_b$ according to (2.10) is thereby written in the simple form of

$$\boldsymbol{p}_B^{\mathcal{A}} = \boldsymbol{G}^{\mathcal{AB}} \boldsymbol{p}_B^{\mathcal{B}}. \tag{2.13}$$

A vector in 3D space is the result of subtracting the coordinates of two points. Within the appended dimension, the 1s cancel each other out. That is, a vector $\boldsymbol{v}$ is appended by 0 in

the fourth dimension and a transformation according to (2.10) yields a transformation by pure rotation

$$\boldsymbol{v}^{\mathcal{A}} = \boldsymbol{R}^{\mathcal{AB}} \cdot \boldsymbol{v}^{\mathcal{B}}. \tag{2.14}$$

The two most important properties for such homogeneous representations $\boldsymbol{G} \in SE(3)$ are its *inversion* and *composition rule*:

$$\boldsymbol{G}^{\mathcal{AB}-1} = \boldsymbol{G}^{\mathcal{BA}}. \qquad \boldsymbol{G}^{\mathcal{AC}} = \boldsymbol{G}^{\mathcal{AB}} \cdot \boldsymbol{G}^{\mathcal{BC}}. \tag{2.15}$$

In most cases, the homogeneous transformation matrices are easy to obtain with respect to the world frame $\mathcal{W}$, like $\boldsymbol{G}^{\mathcal{WA}}$. A point, however, is mostly given relative to the origin of the rigid body frame to which it is fixed. To change the reference of a point $\boldsymbol{p}_B^{\mathcal{B}}$ to $\mathcal{A}$ on body A the following calculation, which combines (2.15), is really useful in practice:

$$\begin{aligned} \boldsymbol{p}_B^{\mathcal{A}} &= \boldsymbol{G}^{\mathcal{WA}-1} \cdot \boldsymbol{G}^{\mathcal{WB}} \cdot \boldsymbol{p}_B^{\mathcal{B}} \\ &= \boldsymbol{G}^{\mathcal{AW}} \cdot \boldsymbol{G}^{\mathcal{WB}} \cdot \boldsymbol{p}_B^{\mathcal{B}} \\ &= \boldsymbol{G}^{\mathcal{AB}} \cdot \boldsymbol{p}_B^{\mathcal{B}}. \end{aligned}$$

When the point $\boldsymbol{p}_B$ is placed at the origin of $\mathcal{B}$, i.e. $\boldsymbol{p}_B^{\mathcal{B}} = \begin{bmatrix} 0 & 0 & 0 & 1 \end{bmatrix}^T$, with (2.13) the point in $\mathcal{A}$ represents exactly the translational offset between both frames

$$\boldsymbol{p}_B^{\mathcal{A}} = \boldsymbol{t}^{\mathcal{AB}}, \text{ for } \boldsymbol{p}_B^{\mathcal{B}} = \begin{bmatrix} 0 & 0 & 0 & 1 \end{bmatrix}^T.$$

The information that is stored in a 4×4 homogeneous transformation $\boldsymbol{G}^{\mathcal{AB}}$ consists of a rotation matrix $\boldsymbol{R}^{\mathcal{AB}}$ and a translational offset $\boldsymbol{t}^{\mathcal{AB}}$. This can be written in vector form by stacking the translational and rotational coordinates

$$\boldsymbol{\chi}^{\mathcal{AB}} := \begin{bmatrix} \boldsymbol{t}^{\mathcal{AB}} \\ \hline \boldsymbol{\phi}^{\mathcal{AB}} \end{bmatrix} \in \mathbb{R}^6, \tag{2.16}$$

with $\boldsymbol{\phi} \in \mathbb{R}^3$ known from (2.9). Such a vector will form the control error of a limb's position later in Sec. 3.3.3 (see also Ibuki et al. (2014)). In the following section an alternative parametrisation of a transformation using exponential coordinates is presented.

The homogeneous representation of transformations allows a convenient handling of relative rigid body mechanics and its usage is of big importance throughout this thesis as it forms the basis of the mechanical system's description.

2.1.4 Rigid body twists, exponential coordinates and screw motions

In this section it is derived that any rigid body motion, from its initial to its final position, can be described by a *screw motion*. That is, the final configuration of a body in space corresponds to a mapping of the body's initial configuration by a rotation about an axis combined with a translation parallel to that axis. To comprehend the notion of such a screw motion, rigid body *twists* are introduced. A twist represents the combined angular and linear velocities $\boldsymbol{\omega} \in \mathbb{R}^3$ and $\boldsymbol{\nu} \in \mathbb{R}^3$ of a rigid body. The velocity of a rigid body is covered in Sec. 2.1.5 in more detail and for now the decomposition in $\boldsymbol{\omega}$ and $\boldsymbol{\nu}$ is sufficient. It is later seen that twists correspond to spatial velocities of rigid bodies and that any screw motion can be associated with a twist.

Twists can be used to describe the relative motion of a rigid body in a parametrisation that already fits to the idea of a screw motion. A twist can either be written in homogeneous representation in a 4×4 matrix from the set $se(3) \subset \mathbb{R}^{4\times4}$ or as vector in $\mathbb{R}^6$. The $\wedge$-operator (wedge), generalized for $\mathbb{R}^6$, transforms a twist from it's coordinate representation to it's

homogeneous representation. Given a twist $\boldsymbol{\xi} \in \mathbb{R}^6$ with the angular velocity component $\boldsymbol{\omega} \in \mathbb{R}^3$ and the linear velocity component $\boldsymbol{\nu} \in \mathbb{R}^3$, it's coordinate representation in $\mathbb{R}^6$ and it's homogeneous representation in $se(3) \subset \mathbb{R}^{4\times4}$ are

$$\boldsymbol{\xi} = \left[\begin{array}{c} \boldsymbol{\nu} \\ \hline \boldsymbol{\omega} \end{array}\right] \in \mathbb{R}^6 \qquad (\boldsymbol{\xi})^\wedge = \hat{\boldsymbol{\Xi}} := \left[\begin{array}{ccc|c} & \hat{\boldsymbol{\Omega}} & & \boldsymbol{\nu} \\ \hline 0 & 0 & 0 & 0 \end{array}\right] \in se(3). \tag{2.17}$$

The set $se(3)$ is the generalization of $so(3)$ from (2.5) to also incorporate the linear velocity component $\boldsymbol{\nu}$:

$$se(3) := \{(\boldsymbol{\nu}, \hat{\boldsymbol{\Omega}}) : \boldsymbol{\nu} \in \mathbb{R}^3, \hat{\boldsymbol{\Omega}} \in so(3)\}. \tag{2.18}$$

As a comprehensive introduction, an analogous *thought experiment* as in Section 2.1.2 is performed. Consider the movement of a point $\boldsymbol{p}$ that is fixed to a rigid body that undergoes a *pure rotation with unit velocity* around the axis $\boldsymbol{\omega} \in \mathbb{R}^3$, with $|\boldsymbol{\omega}| = 1$. With a given point $\boldsymbol{q}$ on the axis, the velocity of the point $\boldsymbol{p}(t)$ is then

$$\dot{\boldsymbol{p}}(t) = \boldsymbol{\omega} \times (\boldsymbol{p}(t) - \boldsymbol{q}). \tag{2.19}$$

This can be expressed in a homogeneous twist representation by the 4×4 matrix

$$\hat{\boldsymbol{\Xi}} = \left[\begin{array}{ccc|c} & \hat{\boldsymbol{\Omega}} & & \boldsymbol{\nu} \\ \hline 0 & 0 & 0 & 0 \end{array}\right]. \tag{2.20}$$

With $\boldsymbol{v} = -\boldsymbol{\omega} \times \boldsymbol{q} \in \mathbb{R}^3$, equation (2.19) can be rewritten to

$$\left[\begin{array}{c} \dot{\boldsymbol{p}} \\ \hline 0 \end{array}\right] = \left[\begin{array}{ccc|c} & \hat{\boldsymbol{\Omega}} & & -\boldsymbol{\omega} \times \boldsymbol{q} \\ \hline 0 & 0 & 0 & 0 \end{array}\right] \cdot \left[\begin{array}{c} \boldsymbol{p} \\ \hline 0 \end{array}\right] = \hat{\boldsymbol{\Xi}} \cdot \left[\begin{array}{c} \boldsymbol{p} \\ \hline 0 \end{array}\right]$$
$$\Rightarrow \dot{\boldsymbol{p}} = \hat{\boldsymbol{\Xi}}\boldsymbol{p}.$$

This is a differential equation and its general solution is obtained by the matrix exponential of $\hat{\boldsymbol{\Xi}}$:

$$\boldsymbol{p}(t) = e^{\hat{\boldsymbol{\Xi}}t}\boldsymbol{p}(0).$$

The parameter t can be exchanged for θ if the rotation occurs about the angle θ instead of unit velocity as in the example from above. Still, this example shows that the exponential of a twist can be used to transform the initial point to its final position. In a similar way, the final configuration in terms of a rigid body transformation $\boldsymbol{G}^{\mathcal{AB}}$ can be expressed using an exponential of a twist and the initial configuration:

$$\boldsymbol{G}(\theta) = e^{\hat{\boldsymbol{\Xi}}\theta}\boldsymbol{G}(0). \tag{2.21}$$

From this follows that in fact every rigid body transformation $\boldsymbol{G}$ can be expressed as the exponential of some twist $\hat{\boldsymbol{\Xi}}\theta$,

$$\boldsymbol{G} = e^{\hat{\boldsymbol{\Xi}}\theta}.$$

The rigid body movement, at any time t, is therefore said to be generated by the instantaneous twist $\hat{\boldsymbol{\Xi}}(t)\theta(t) \in se(3)$. The vector $\boldsymbol{\xi}\theta \in \mathbb{R}^6$ is referred to as the *exponential coordinates* for the rigid body transformation $\boldsymbol{G}$ (Murray et al., 1994).

The exponential of a twist $\hat{\boldsymbol{\Xi}}\theta \in se(3)$, to obtain $G \in SE(3)$, is in fact the generalization of the matrix exponential of the axis $\boldsymbol{\omega} \in so(3)$ to obtain a rotation matrix $R = e^{\hat{\boldsymbol{\Omega}}\theta} \in SO(3)$

(compare to Section 2.1.2). The calculation of $\boldsymbol{G}$ as a matrix exponential of a twist $\boldsymbol{\xi}$ is presented in Murray et al. (1994) and includes the use of *Rodrigues' formula* (2.6).

In practice, often the twist coordinates $\boldsymbol{\nu}$ and $\boldsymbol{\omega}$ are known a-priori from the anatomy of the system. For the movement of a 1D, rotational revolute joint, for example, the axis $\boldsymbol{\omega}$ is exactly the rotation axis of the joint. The corresponding calculation of the matrix exponential of the twist that generates this motion takes a simple form. Refer to Murray et al. (1994) or Legnani et al. (1996b,a) for the closed-form representation of a free motion and of the simplifications for special joints.

Having the notion of twists introduced, they are now related to a *screw motion*. The twist with coordinates $\boldsymbol{\omega}$ and $\boldsymbol{\nu}$ generates a movement by rotation around $\boldsymbol{\omega}$ and translation along $\boldsymbol{\nu}$. The motion of a screw, i.e. rotation and parallel translation, seems to be a special case for arbitrary rigid body motions, but in fact it is not. According to Kumar (2014), this is one of the most fundamental results in spatial kinematics. It is stated in a theorem that is usually attributed to Chasles (1830), although Mozzi and Cauchy are credited with earlier results that are similar, e.g. (Mozzi, 1763).

Theorem 2.1.2 (Chasles) *Every rigid body motion can be realized by a rotation about an axis combined with a translation parallel to that axis (Murray et al., 1994).*

That is, for every rigid body motion, a generating twist in the form of (2.20) can be found, where $\boldsymbol{\omega}$ describes the rotation axis and $\boldsymbol{\nu}$ the translation parallel to that axis. A proof of Theorem 2.1.4 and a relation of the exponential twist coordinates to the parameters of a screw is given in Murray et al. (1994). The consequence of this theorem is that the used notation of homogeneous representations is eligible for describing arbitrary relative rigid body motions.

2.1.5 Rigid body velocities and velocity transformations

The notion of twists from Sec. 2.1.4 is strongly connected to the velocity of a rigid body. As a twist describes an instantaneous screw motion, a twist is in fact a representation of a body's velocity. In this section some more details on the velocity of rigid bodies are introduced. This includes laws for the transformation of velocities between different frames using rigid body transformations $\boldsymbol{G}$ from (2.11).

Assume $\boldsymbol{p}_B^{\mathcal{B}}$ to be a point attached to the rigid body B, viewed from the body frame $\mathcal{B}$. With the rigid body transformation $\boldsymbol{G}^{\mathcal{AB}}$ the reference can be changed to the frame $\mathcal{A}$ on body A, as already known from (2.13). A representation of the velocity of this point relative to the frame $\mathcal{A}$, and viewed from frame $\mathcal{A}$, can be constructed by differentiation

$$\begin{aligned} \boldsymbol{\nu}_{\mathcal{A}p_B}^{\mathcal{A}} &:= \dot{\boldsymbol{p}}_B^{\mathcal{A}} = \dot{\boldsymbol{G}}^{\mathcal{AB}} \cdot \boldsymbol{p}_B^{\mathcal{B}} = \underbrace{\dot{\boldsymbol{G}}^{\mathcal{AB}} \cdot \boldsymbol{G}^{\mathcal{AB}\,-1}}_{:=\hat{\boldsymbol{V}}_{\mathcal{AB}}^{\mathcal{A}}} \cdot \boldsymbol{p}_B^{\mathcal{A}} \\ \boldsymbol{\nu}_{\mathcal{A}p_B}^{\mathcal{A}} &= \hat{\boldsymbol{V}}_{\mathcal{AB}}^{\mathcal{A}} \cdot \boldsymbol{p}_B^{\mathcal{A}} \end{aligned} \tag{2.22}$$

It is hereby seen that the relative velocity of a point can be expressed in the *homogeneous representation*

$$\begin{aligned} \hat{\boldsymbol{V}}_{\mathcal{AB}}^{\mathcal{A}} &= \dot{\boldsymbol{G}}^{\mathcal{AB}} \boldsymbol{G}^{\mathcal{AB}\,-1} = \left[\begin{array}{ccc|c} & \dot{\boldsymbol{R}}^{\mathcal{AB}} \boldsymbol{R}^{\mathcal{AB}\,T} & & -\dot{\boldsymbol{R}}^{\mathcal{AB}} \boldsymbol{R}^{\mathcal{AB}\,T} \boldsymbol{t}^{\mathcal{AB}} + \dot{\boldsymbol{t}}^{\mathcal{AB}} \\ \hline 0 & 0 & 0 & 0 \end{array}\right] \\ &:= \left[\begin{array}{ccc|c} & \hat{\boldsymbol{\Omega}}_{\mathcal{AB}}^{\mathcal{A}} & & \boldsymbol{\nu}_{\mathcal{AB}}^{\mathcal{A}} \\ \hline 0 & 0 & 0 & 0 \end{array}\right] \in se(3) \subset \mathbb{R}^{4\times 4} \end{aligned} \tag{2.23}$$

that has the form of a twist (2.20). The *coordinate representation* is

$$\boldsymbol{v}_{\mathcal{AB}}^{\mathcal{A}} = \left[\begin{array}{c} -\dot{\boldsymbol{R}}^{\mathcal{AB}}\boldsymbol{R}^{\mathcal{AB}T}\boldsymbol{t}^{\mathcal{AB}} + \dot{\boldsymbol{t}}^{\mathcal{AB}} \\ \hline \left(\dot{\boldsymbol{R}}^{\mathcal{AB}}\boldsymbol{R}^{\mathcal{AB}T}\right)^{\vee} \end{array}\right] = \left[\begin{array}{c} \boldsymbol{\nu}_{\mathcal{AB}}^{\mathcal{A}} \\ \hline \boldsymbol{\omega}_{\mathcal{AB}}^{\mathcal{A}} \end{array}\right] \in \mathbb{R}^6. \tag{2.24}$$

To clarify the notation and meaning of this definition: $\boldsymbol{\nu}_{\mathcal{A}p_B}^{\mathcal{A}}$ in (2.24) is the velocity of the point $\boldsymbol{p}_B$ relative to the frame $\mathcal{A}$ (subscript) as viewed from the frame $\mathcal{A}$ (superscript). This is the *spatial velocity* of a point on a body B, as viewed from the spatial frame $\mathcal{A}$. The velocity $\hat{\boldsymbol{V}}_{\mathcal{AB}}^{\mathcal{A}}$ is therefore also known as the *spatial velocity* of $\mathcal{B}$ in $\mathcal{A}$. Another representation, the *body velocity*, is presented in Appendix A.2.

Often the velocities of rigid bodies are expressed relative to the inertial world frame, i.e. $\mathcal{A}\hat{=}\mathcal{W}$ in the formulations above. In this case, the relative velocity of the three rigid bodies A, B and C, with their respective frames $\mathcal{A}$, $\mathcal{B}$ and $\mathcal{C}$, can be easily calculated (Legnani et al., 1996b):

$$\hat{\boldsymbol{V}}_{\mathcal{AC}}^{\mathcal{A}} = \hat{\boldsymbol{V}}_{\mathcal{AB}}^{\mathcal{A}} + \hat{\boldsymbol{V}}_{\mathcal{BC}}^{\mathcal{A}}. \tag{2.25}$$

In some special cases, the relative velocities of $\mathcal{B}$ and $\mathcal{C}$ are not known with respect to the frame $\mathcal{A}$, i.e. $\boldsymbol{v}_{\mathcal{BC}}^{\mathcal{A}}$ is unknown, but with respect to the frame $\mathcal{B}$. By changing the reference frame

$$\hat{\boldsymbol{V}}_{\mathcal{AB}}^{\mathcal{A}} = \boldsymbol{G}^{\mathcal{AB}}\hat{\boldsymbol{V}}_{\mathcal{AB}}^{\mathcal{B}}\boldsymbol{G}^{\mathcal{AB}-1} \qquad \text{and} \qquad \hat{\boldsymbol{V}}_{\mathcal{AB}}^{\mathcal{B}} = \boldsymbol{G}^{\mathcal{AB}-1}\hat{\boldsymbol{V}}_{\mathcal{AB}}^{\mathcal{A}}\boldsymbol{G}^{\mathcal{AB}}. \tag{2.26}$$

, the expression of the velocity in $\mathcal{A}$ is obtained by (see also Appendix A.2 or Murray et al. (1994); Legnani et al. (1996b,a)):

$$\hat{\boldsymbol{V}}_{\mathcal{AC}}^{\mathcal{A}} = \hat{\boldsymbol{V}}_{\mathcal{AB}}^{\mathcal{A}} + \boldsymbol{G}^{\mathcal{AB}} \cdot \hat{\boldsymbol{V}}_{\mathcal{BC}}^{\mathcal{B}} \cdot \boldsymbol{G}^{\mathcal{AB}-1} \tag{2.27}$$

As a final remark for this section, the notation of relative rigid body velocities are related to a screw motion (see also Murray et al. (1994) for details). Therefore reconsider, that any movement can be represented by a screw motion that relates the initial relative configuration $\boldsymbol{G}^{\mathcal{AB}}(0)$ of the frames $\mathcal{A}$ and $\mathcal{B}$ to its final configuration $\boldsymbol{G}^{\mathcal{AB}}(\theta)$ by the exponential of the constant twist $\hat{\boldsymbol{\Xi}}\theta$ as stated in (2.21). Plugging (2.21) into the velocity definition (2.23) yields

$$\begin{aligned} \hat{\boldsymbol{V}}_{\mathcal{AB}}^{\mathcal{A}} &= \dot{\boldsymbol{G}}^{\mathcal{AB}}\boldsymbol{G}^{\mathcal{AB}-1} \\ &= \frac{\mathrm{d}}{\mathrm{d}t}\left(e^{\hat{\boldsymbol{\Xi}}^{\mathcal{AB}}\theta}\boldsymbol{G}^{\mathcal{AB}}(0)\right)\left(e^{\hat{\boldsymbol{\Xi}}^{\mathcal{AB}}\theta}\boldsymbol{G}^{\mathcal{AB}}(0)\right)^{-1} \\ &= \left(\hat{\boldsymbol{\Xi}}\dot{\theta}e^{\hat{\boldsymbol{\Xi}}^{\mathcal{AB}}\theta}\boldsymbol{G}^{\mathcal{AB}}(0)\right)\left(\boldsymbol{G}^{\mathcal{AB}-1}(0)e^{-\hat{\boldsymbol{\Xi}}^{\mathcal{AB}}\theta}\right) \\ &= \hat{\boldsymbol{\Xi}}^{\mathcal{AB}}\dot{\theta}. \end{aligned} \tag{2.28}$$

Note, that here the identity $\frac{\mathrm{d}}{\mathrm{d}t}(e^{\hat{\boldsymbol{\Xi}}^{\mathcal{AB}}\theta}) = \hat{\boldsymbol{\Xi}}\dot{\theta}e^{\hat{\boldsymbol{\Xi}}^{\mathcal{AB}}\theta}$ is used (Murray et al., 1994). In (2.28) it is clearly seen that the spatial velocity $\hat{\boldsymbol{V}}_{\mathcal{AB}}^{\mathcal{A}}$ of a body in fact is generated by the twist $\hat{\boldsymbol{\Xi}}^{\mathcal{AB}}$.

$$\hat{\boldsymbol{V}}_{\mathcal{AB}}^{\mathcal{A}} = \hat{\boldsymbol{\Xi}}^{\mathcal{AB}}\dot{\theta}.$$

As every twist can be described by a screw motion, this velocity corresponds to a screw motion. The notion of relative rigid body motion is used later in the design of a position controller. In order to evaluate the control performance, the positional error $\boldsymbol{\chi} \in \mathbb{R}^6$ (2.16) and the spatial velocity coordinates $\boldsymbol{v}$ (2.24) are used.

2.1.6 Rigid body acceleration and other dynamics variables

To derive the equations of motion, the homogeneous representations in 4×4-matrices of the *acceleration*, the *inertia* and the *momentum* of a rigid body are needed. Most of the mathematical derivations are omitted here and it is referred to Legnani et al. (1996b,a) and Murray et al. (1994).

The acceleration of a rigid body can be written in a homogeneous representation in the form of a 4×4 matrix, similar to the homogeneous representations of positions and velocities (twists) from above. For a point $\boldsymbol{p}_B^{\mathcal{A}}$, attached to the the body B and viewed from the (inertial) frame $\mathcal{A}$, the relative rigid body accelerations $\ddot{\boldsymbol{p}}_B$ fulfils

$$\ddot{\boldsymbol{p}}_B^{\mathcal{A}} = \hat{\boldsymbol{A}}_{\mathcal{AB}}^{\mathcal{A}} \boldsymbol{p}_B^{\mathcal{A}},$$

where

$$\hat{\boldsymbol{A}}_{\mathcal{AB}}^{\mathcal{A}} := \dot{\hat{\boldsymbol{V}}}_{\mathcal{AB}}^{\mathcal{A}} + \hat{\boldsymbol{V}}_{\mathcal{AB}}^{\mathcal{A}^2}.$$

The homogeneous representation of a body's acceleration $\hat{\boldsymbol{A}} \in \mathbb{R}^{4\times4}$ writes

$$\hat{\boldsymbol{A}} = \left[\begin{array}{ccc|c} & \dot{\hat{\boldsymbol{\Omega}}} + \hat{\boldsymbol{\Omega}}^2 & & \boldsymbol{\alpha} \\ \hline 0 & 0 & 0 & 0 \end{array}\right] \in \mathbb{R}^{4\times4}. \tag{2.29}$$

The change of the reference frame for rigid body accelerations follows

$$\hat{\boldsymbol{A}}^{\mathcal{A}} = \boldsymbol{G}^{\mathcal{AB}} \hat{\boldsymbol{A}}^{\mathcal{B}} \boldsymbol{G}^{\mathcal{AB}^{-1}}. \tag{2.30}$$

The relative motion of the three frames $\mathcal{B}$, $\mathcal{C}$ and $\mathcal{D}$, viewed from $\mathcal{A}$ can be written by

$$\hat{\boldsymbol{A}}_{\mathcal{BD}}^{\mathcal{A}} = \hat{\boldsymbol{A}}_{\mathcal{BC}}^{\mathcal{A}} + \hat{\boldsymbol{A}}_{\mathcal{CD}}^{\mathcal{A}} + 2\hat{\boldsymbol{V}}_{\mathcal{BC}}^{\mathcal{A}} \hat{\boldsymbol{V}}_{\mathcal{CD}}^{\mathcal{A}}. \tag{2.31}$$

Similar transformation laws as (2.26) and (2.30) hold for the homogeneous representations of the *inertia matrix* $\boldsymbol{J} \in \mathbb{R}^{4\times4}$, *momentum matrix* $\boldsymbol{\mathcal{P}}$ and the force matrix $\boldsymbol{\Phi}$:

$$\hat{\boldsymbol{\mathcal{J}}}_B^{\mathcal{A}} = \boldsymbol{G}^{\mathcal{AB}} \hat{\boldsymbol{\mathcal{J}}}_B^{\mathcal{B}} \boldsymbol{G}^{\mathcal{AB}^{-1}} \qquad \hat{\boldsymbol{\mathcal{P}}}_B^{\mathcal{A}} = \boldsymbol{G}^{\mathcal{AB}} \hat{\boldsymbol{\mathcal{P}}}_B^{\mathcal{B}} \boldsymbol{G}^{\mathcal{AB}^{-1}}$$
$$\hat{\boldsymbol{\Phi}}_B^{\mathcal{A}} = \boldsymbol{G}^{\mathcal{AB}} \hat{\boldsymbol{\Phi}}_B^{\mathcal{B}} \boldsymbol{G}^{\mathcal{AB}^{-1}}. \tag{2.32}$$

As already mentioned above, the derivation and further details on these dynamical variables are mostly omitted here. A full comprehensive theory is given by Legnani et al. (1996b,a); Murray et al. (1994); Lynch and Park (2017). It is important to note that the reference of all these dynamical system matrices can be changed by a rigid body transformation $\boldsymbol{G}$. The rigid body transformation $\boldsymbol{G}$ itself is calculated based on the system's state and includes knowledge on the 'anatomy' of the system. This means, the joint types of the model restrict the movement and simplifies the calculation of $\boldsymbol{G}$ of two adjacent bodies.

2.1.7 Lagrange equations of motion

Using the findings from above, the equations of motion for a whole *open-chain* system can be obtained. This is even possible in an algorithmic implementation as described by Legnani et al. (1996b,a) that follows a *Lagrangian* formalism for deriving the equations of motion. This formalism is based on the well-known formula

$$\frac{\mathrm{d}}{\mathrm{d}t} \frac{\partial \mathcal{L}}{\partial \dot{q}_i} + \frac{\partial \mathcal{L}}{\partial q_i} = \mathrm{T}_i,$$

where $\mathcal{L} := \mathcal{T} - \mathcal{V}$ is the Lagrangian function of kinetic and potential energy, q_i is a generalised degree of freedom and T_i is a generalised force or torque directly acting on q_i.

The energy terms of this equation can be expressed for an open chain of rigid bodies by the homogeneous representation matrices that where introduced in the previous sections

$$\mathcal{T}_i = \frac{1}{2}\text{trace}(\hat{\boldsymbol{V}}_{\mathcal{W}\mathcal{I}}^{\mathcal{W}}\boldsymbol{\mathcal{J}}_I^{\mathcal{W}}\hat{\boldsymbol{V}}_{\mathcal{W}\mathcal{I}}^{\mathcal{W}}) \qquad \mathcal{V}_i = \text{trace}(-\boldsymbol{H}_g^{\mathcal{W}}\boldsymbol{\mathcal{J}}_I^{\mathcal{W}}).$$

With further calculations (see Legnani et al. (1996a) and Henze (2002) for details), it is possible to express the dynamics of the whole open chain system by the matrix equation

$$\boldsymbol{M}\ddot{\boldsymbol{q}} + \boldsymbol{c}(\boldsymbol{q}, \dot{\boldsymbol{q}}) = \boldsymbol{\tau} \in \mathbb{R}^{n_{\text{DoF}}}, \tag{2.33}$$

where $\boldsymbol{\tau} \in \mathbb{R}^{n_{\text{DoF}}}$ is a vector of generalized forces that act directly on the DoFs $\boldsymbol{q} \in \mathbb{R}^{n_{\text{DoF}}}$. This equation (2.33) is the *equation of motion* of the whole rigid body system. The matrix equation for deriving $\boldsymbol{M}$ and $\boldsymbol{c}$ are given in Appendix A.3 (A.29)-(A.31). The whole system (2.33) has the dimension n_{DoF} and the respective matrix entries are summed up by the individuals of the homogeneous representations for the single bodies. These matrices are obtained by the motion composition rules

$$\begin{aligned}
\boldsymbol{G}^{\mathcal{W}\mathcal{N}} &= \boldsymbol{G}^{\mathcal{W}\mathcal{M}}\boldsymbol{G}^{\mathcal{M}\mathcal{N}} && (2.34)\\
\hat{\boldsymbol{V}}_{\mathcal{W}\mathcal{N}}^{\mathcal{W}} &= \hat{\boldsymbol{V}}_{\mathcal{W}\mathcal{M}}^{\mathcal{W}} + \hat{\boldsymbol{V}}_{\mathcal{M}\mathcal{N}}^{\mathcal{W}} && (2.35)\\
\boldsymbol{H}_{\mathcal{W}\mathcal{N}}^{\mathcal{W}} &= \boldsymbol{H}_{\mathcal{W}\mathcal{M}}^{\mathcal{W}} + \boldsymbol{H}_{\mathcal{M}\mathcal{N}}^{\mathcal{W}} + 2\boldsymbol{H}_{\mathcal{W}\mathcal{M}}^{\mathcal{W}}\boldsymbol{H}_{\mathcal{M}\mathcal{N}}^{\mathcal{W}}, && (2.36)
\end{aligned}$$

which are based on equations (2.15), (2.25) and (2.31) (Legnani et al., 1996b,a; Murray et al., 1994). Starting at a body $\mathcal{M}$ with given $\boldsymbol{G}^{\mathcal{W}\mathcal{M}}$, and knowing the transformation to the following body $\boldsymbol{G}^{\mathcal{M}\mathcal{N}}$, the frame $\mathcal{N}$ can be described in the frame $\mathcal{W}$. In practice, this information is usually present, as the relative transformation $\boldsymbol{G}^{\mathcal{M}\mathcal{N}}$ is based on the type of the connecting joint between the bodies M and N. Using successive calculations, the whole structure of an open-chain system of rigid bodies can be reached by calculation. With the equations for deriving the system matrices, given in Appendix A.3, the matrix $\boldsymbol{M}$ and the vector $\boldsymbol{c}$ on the left side of equation (2.33) can be calculated based on the current state $\boldsymbol{q}(t)$ and $\dot{\boldsymbol{q}}(t)$.

The right side of (2.33) is formed by the vector of generalised forces $\boldsymbol{\tau}$. These are external forces that act on the DoFs of the system and include forces from the modelled muscles $\boldsymbol{\tau}^{\text{mtu}}$ (see Sec. 2.2), of joint limitations and of visco-elastic elements $\boldsymbol{\tau}^{lmt}$, $\boldsymbol{\tau}^{\text{bsh}}$ (see Sec 2.3) and of contacts with the environment $\boldsymbol{\tau}^{\text{cnt}}$ (see Sec 2.4). In the Appendix Chpt. D.2 the homogeneous representations of rigid body transformations $\boldsymbol{G}$, twists $\hat{\boldsymbol{\Xi}}$ and velocities $\hat{\boldsymbol{V}}$ are correlated to the notation used in the simulation software `demoa`.

2.2 Muscle Model

Muscles are biological actuators and responsible for the movement of animals. While there are different types of muscles in a biological individual, the only focus in this work lies on skeletal muscles. Those muscles are attached to the bones by tendons and generate the movement of the whole body in the present mathematical system and in real life. Skeletal muscles consist of fascicles, which themselves consist of muscle fibres. The fibres are specialized cells with the cytoplasmic proteins *myosin* and *actin* forming the consecutively arranged *sarcomeres*. These sarcomeres are excitable by pools of motor neurons from within the spinal cord. The neural excitation that arrives at a neuromuscular junction of a muscle releases a specific neurotransmitter, which subsequently leads to a release of Ca^{2+} ions in the sarcoplasm. With a rising Ca^{2+} ion concentration more myosin-binding sites are exposed to actin, which allows actin and myosin to engage in binding and contracting the muscle. The result is a contracting force of the muscle belly, which is transduced to the bones via the passive elastic properties of the tendons (Hill, 1949; Huxley, 1969; Martonosi, 2000; Mörl et al., 2012). The effect of this force on an embraced joint can be determined by

the geometric implementation of the muscle, i.e. by calculating the joint torque with the muscle's moment arm.

Within this section, mathematical model descriptions of different parts of the muscular system are presented. In this formulation each muscle is modelled from a macroscopic viewpoint. This means that all muscle fibres and sarcomeres are combined to a single contracting unit, i.e. the multiple stimulations of individual neuromuscular junctions are combined into one scalar stimulation signal. By this, each muscle can be modelled individually as a one dimensional, scalar, dynamical system of second order. The dynamic relation of tension between active muscle fibres and passive tendons of a muscle is modelled by means of a macroscopic *muscle-tendon unit* (MTU). These active and passive, mechanical tissue-properties of a MTU are modelled in a non-linear differential equation of first order, the *contraction dynamics*. As a result the total force of a MTU is obtained that depends on its length and biochemical *activity* (see Sec.2.2.2 and Haeufle et al. (2014a)). The state of activity of a MTU depends on the concentration of free $Ca2^+$ ions in the sarcoplasm. The *activation-dynamics*, i.e. the response of free CA^{2+} ions to neural *stimulation*, are modelled by another differential equation (see Sec.2.2.1 and Rockenfeller and Günther (2018)). The stimulation signals by which the MTUs are commanded originate from the spinal cord and higher centres of the CNS. Within the spinal cord, proprioceptive feedback circuits transmit afferent sensor signals back to stimulate the MTU, e.g. the *mono-synaptic reflex*. This feedback loop of the mono-synaptic reflex with afferent feedback from the muscle spindle is modelled as a low-level PD-controller (see Sec. 2.2.3 and Bayer et al. (2017)). In Sec. 2.2.4 the matrix of moment arms is presented that converts MTU forces into joint torques (Hammer et al., 2019). In Sec. 2.3 and 2.4 further modelled torque contributions of passive, visco-elastic tissue and of contacts with the environment are presented.

2.2.1 Activation dynamics

The model of activation dynamics from Rockenfeller and Günther (2018) describes the electrochemical process of transforming the electrical neural signal into the chemical Ca2+ ion concentration in the sarcoplasm of the muscle. This model is an enhanced version of what had originally been developed by Hatze (1977). Therefore, the normalised signal variable $u(t) \in \mathbb{R}_1$ is introduced as a value of total neural excitation of all neuromuscular junctions of a muscle. The state variable $\gamma(t) \in \mathbb{R}_1$ is the Ca^{2+} ion concentration in the sarcoplasm and $a(t) \in \mathbb{R}_1$ is the activity of a MTU. The set $\mathbb{R}_1$ is defined as a normalisation of $\mathbb{R}$ to

$$\mathbb{R}_1 := \{u \in \mathbb{R} : u_0 \leq u < 1\}, \tag{2.37}$$

where $u_0 > 0$ is a minimal base stimulation level of the muscles. To model the activation dynamics, firstly the total neural stimulation signal $u(t)$ that arrives at the neuromuscular junctions serves as input for a first order differential equation with the state variable of the normalised Ca^{2+} ion concentration $\gamma(t)$:

$$\dot{\gamma} = M_{\mathrm{H}}(u - \gamma). \tag{2.38}$$

In a second step, the activity $a(t)$ of the MTU is obtained by the non-linear relationship

$$a(\gamma, l^{\mathrm{CE}}) = \frac{a_0 + \vartheta(\gamma, l^{\mathrm{CE}})}{1 + \vartheta(\gamma, l^{\mathrm{CE}})}, \tag{2.39}$$

with $\vartheta = (\gamma \cdot \varpi(l^{\mathrm{CE}}))^{\nu}$, where $\varpi(l^{\mathrm{CE}}) = \varpi_{\mathrm{opt}} \cdot \frac{l^{\mathrm{CE}}}{l^{\mathrm{CE}}_{\mathrm{opt}}} = \gamma_{\max} \cdot \rho_0 \cdot \frac{l^{\mathrm{CE}}}{l^{\mathrm{CE}}_{\mathrm{opt}}}$.

The activation dynamics in (2.38) are a first-order differential equation which is further non-linearly scaled to the activity state of the muscle (2.39). The dynamical behaviour of how $a(t)$ follows $u(t)$ is comprehensively displayed by Bayer et al. (2017).

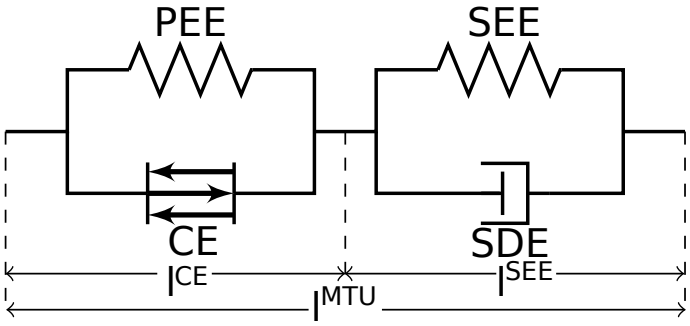


Figure 2.2: Diagram of a *muscle-tendon unit* (MTU) with *contractile element* (CE), *parallel elastic element* (PEE), *serial elastic element* (SEE) and *serial damping element* (SDE) (Haeufle et al., 2014a). The CE and PEE together form a model of the muscle belly and the SEE and SDE of the tendons that connect the muscle to the bones.

2.2.2 Hill-type muscle-tendon-units

Hill-type muscle models are macroscopic muscle models that aim to have an equivalent mechanical effect on the skeletal system compared to a biological muscle. This idea is originally based on Hill (1938). Hill-type muscle models typically consist of several functional elements that correspond to the biophysical structures of a muscle (see Fig 2.2 for the used version (Haeufle et al., 2014a; Mörl et al., 2012; Günther et al., 2007)). Most prominently, the *contractile element* (CE) models the active properties of the muscle belly and includes length and velocity dependencies of actin and myosin bindings within a sarcomere. The *parallel elastic element* (PEE) and *serial elastic element* (SEE) represent the passive properties of the muscle-belly tissue and the tendon, when externally stretched. The *serial damping element* (SDE) introduces dynamical properties of the strain of a tendon. Commonly, the tendons at both sides of a muscle are folded to one side for an equivalent representation in the MTU. With this, the total length l^{MTU} of a MTU is the sum of the lengths of the CE (muscle belly, l^{CE}) and of the SEE (tendons, l^{SEE}):

$$l^{\mathrm{MTU}} = l^{\mathrm{CE}} + l^{\mathrm{SEE}}. \tag{2.40}$$

Given the current lengths $l^{\mathrm{CE}}(t)$ and $l^{\mathrm{SEE}}(t)$, and thereby $l^{\mathrm{MTU}}(t)$, as well as the velocities $\dot{l}^{\mathrm{CE}}$, $\dot{l}^{\mathrm{SEE}}$ and $\dot{l}^{\mathrm{MTU}}$ of the single elements and of the whole MTU, respectively, each element, and thereby the whole MTU, exerts a force. These individual forces are linked by the force equilibrium

$$f^{\mathrm{MTU}} = f^{\mathrm{CE}} + f^{\mathrm{PEE}} = f^{\mathrm{SEE}} + f^{\mathrm{SDE}}, \tag{2.41}$$

The individual parameter dependencies are

$$\begin{aligned} f^{\mathrm{CE}} &= f^{\mathrm{CE}}(l^{\mathrm{CE}}, \dot{l}^{\mathrm{CE}}, a), && (2.42)\\ f^{\mathrm{PEE}} &= f^{\mathrm{PEE}}(l^{\mathrm{CE}}), && (2.43)\\ f^{\mathrm{SEE}} &= f^{\mathrm{SEE}}(l^{\mathrm{SEE}}), && (2.44)\\ f^{\mathrm{SDE}} &= f^{\mathrm{SDE}}(l^{\mathrm{CE}}, \dot{l}^{\mathrm{CE}}, \dot{l}^{\mathrm{MTU}}, a) && (2.45) \end{aligned}$$

and thereby $f^{\mathrm{MTU}} = f^{\mathrm{MTU}}(l^{\mathrm{CE}}, \dot{l}^{\mathrm{CE}}, \dot{l}^{\mathrm{MTU}}, a)$.

All of the force laws (2.42)-(2.45) are algebraic functions of the muscle's state. They are parametrised by specific properties that are connected to the biophysical behaviour of the modelled muscle (Haeufle et al., 2014a). The length of a muscle $l^{\mathrm{CE}}(t)$ additionally follows a non-linear, first-order differential equation, the so-called *contraction dynamics* $\dot{l}^{\mathrm{CE}} = f(l^{\mathrm{CE}}, ...)$.

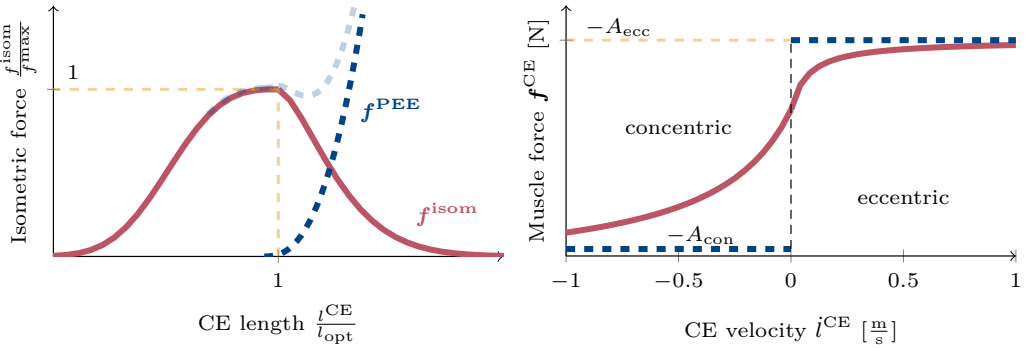


FIGURE 2.3: Left: Force-length relation of a CE (red) and a PEE (blue, dashed). At its optimal length $l^{\text{CE}} = l_{\text{opt}}$, a CE can exert its maximum isometric force. When the MTU is stretched beyond its optimal length, the non-linear, passive elastic force of the PEE rises and the combined force increases. Right: Force-velocity relation of a CE for the concentric and eccentric branch according to (2.47). The hyperbolas' asymptotes are given by $A_{\text{dir}} = A_{\text{rel,dir}} \cdot f^{\text{CE}}_{\text{max}}$ (dashed lines).

The following subsections give a short summary of the force laws (2.42)-(2.45) of the MTU's elements. For more details on the MTU model and an in-depth explanation of the contraction dynamics please refer to Haeufle et al. (2014a). For a list of MTU parameters that are used in the model later refer to Appendix C.

2.2.2.1 The contractile element (CE)

The CE models the active fibres of a muscle, where motor proteins are responsible for actin and myosin bindings that develop the binding force $f^{\text{CE}}(l^{\text{CE}}, \dot{l}^{\text{CE}}, a)$. This force is dependent on the current length of the muscle fibres, $f^{\text{CE}}(l^{\text{CE}}, ...)$, their concentric or eccentric velocity, $f^{\text{CE}}(\dot{l}^{\text{CE}}, ...)$ and of the current state of activity, $f^{\text{CE}}(a, ...)$. Therefore, the force-length and the force-velocity relation form a important correlations for the CE.

The force-length relation can easily be motivated by picturing consecutively arranged actin and myosin proteins that have a larger overlap at a certain configuration of the fibre's lengths and thereby the whole muscle belly exerts a force dependent on the fibre length.

Experimental measurements have shown that the *isometric force*[1] f^{isom} of a muscle belly shows a bell-shaped length dependency that is modelled according to Haeufle et al. (2014a) by two exponential functions of the form

$$f^{\text{isom}}(l^{\text{CE}}) = e^{\left\{ -\left| \frac{\frac{l^{\text{CE}}}{l^{\text{CE}}_{\text{opt}}} - 1}{\Delta W_{\text{limb}}} \right|^{\nu^{\text{CE}}_{\text{limb}}} \right\}}. \tag{2.46}$$

The parameter l_{opt} hereby describes the optimal length of the CE, where the maximal isometric force can occur. The parameters ΔW_{limb} and $\nu^{\text{CE}}_{\text{limb}}$ describe the shape of the ascending (subscript limb = asc) and of the descending (subscript limb = des) branches of the bell-shaped exponential function (2.46). This length dependency is displayed in Fig. 2.3 (left) in the normalized coordinates of $f^{\text{isom}}/f^{\text{max}}(l^{\text{CE}}/l_{\text{opt}})$.

[1] An isometric force is a force exerted by the muscle fibres at a constant length.

The force-velocity relation describes experimentally observed phenomena of a velocity dependence of the force of a muscle. The force of a muscle can thereby increase with an increasing eccentric (stretching) velocity of the muscle's fibres. For increasing concentric (shortening) velocities, the muscle force typically decreases compared to isometric conditions (Katz, 1939; Joyce et al., 1969; Till et al., 2008). Both cases (subscripts dir = ecc for $\dot{l}^{\mathrm{CE}} > 0$ and dir = con for $\dot{l}^{\mathrm{CE}} \leq 0$) can be modelled by hyperbolic functions with a different set of parameters. The force velocity is comprehensively outlined by Haeufle et al. (2014a). Taking account of the velocity dependency and the length dependency (2.46), the force of the CE follows

$$f_{\mathrm{dir}}^{\mathrm{CE}}(l^{\mathrm{CE}}, \dot{l}^{\mathrm{CE}}, a) = f_{\mathrm{max}}^{\mathrm{CE}} \cdot \left(\frac{a \cdot f^{\mathrm{isom}}(l^{\mathrm{CE}}) + A_{\mathrm{rel,dir}}(l^{\mathrm{CE}}, a)}{1 - \frac{\dot{l}^{\mathrm{CE}}}{B_{\mathrm{rel,dir}}(l^{\mathrm{CE}}, a) \cdot l_{\mathrm{opt}}^{\mathrm{CE}}}} - A_{\mathrm{rel,dir}}(l^{\mathrm{CE}}, a) \right), \tag{2.47}$$

where a is the current activation of the CE according to (2.39), and $A_{\mathrm{rel,dir}}$ and $B_{\mathrm{rel,dir}}$ are the *Hill-parameters* that define the shape of the force-velocity relation for concentric and eccentric length changes, respectively. To have a continuity at $\dot{l}^{\mathrm{CE}} = 0$ the velocity constrain $\frac{\partial f_{\mathrm{ecc}}^{\mathrm{CE}}}{\partial \dot{l}^{\mathrm{CE}}} = S_e \cdot \frac{\partial f_{\mathrm{con}}^{\mathrm{CE}}}{\partial \dot{l}^{\mathrm{CE}}}$, parametrized by S_e and the force constrain $f^{\mathrm{CE}} \to F_e a f^{\mathrm{isom}} f_{\mathrm{max}}^{\mathrm{CE}}$ for $\dot{l}^{\mathrm{CE}} \to +\infty$ are used and the *concentric Hill*-parameters are defined as

$$\begin{aligned}
A_{\mathrm{rel,con}}(l^{\mathrm{CE}}, a) &= A_{\mathrm{rel},0} \cdot L_{A_{\mathrm{rel}}}(l^{\mathrm{CE}}) \cdot Q_{A_{\mathrm{rel}}}(a) \\
&\text{with} \quad L_{A_{\mathrm{rel}}}(l^{\mathrm{CE}}) = \begin{cases} 1, & l^{\mathrm{CE}} < l_{\mathrm{opt}} \\ f^{\mathrm{isom}}(l^{\mathrm{CE}}) & l^{\mathrm{CE}} \geq l_{\mathrm{opt}} \end{cases} \\
&\text{and} \quad Q_{A_{\mathrm{rel}}}(a) = \frac{1}{4} \cdot (3 + 4 \cdot a), \\
B_{\mathrm{rel,con}}(l^{\mathrm{CE}}, a) &= B_{\mathrm{rel},0} \cdot L_{B_{\mathrm{rel}}}(l^{\mathrm{CE}}) \cdot Q_{B_{\mathrm{rel}}}(a) \\
&\text{with} \quad L_{B_{\mathrm{rel}}}(l^{\mathrm{CE}}) = 1 \\
&\text{and} \quad Q_{B_{\mathrm{rel}}} = \frac{1}{7} \cdot (3 + 4 \cdot a)
\end{aligned}$$

and the *eccentric Hill*-parameters as

$$\begin{aligned}
A_{\mathrm{rel,ecc}} &= -F_e a f^{\mathrm{isom}} \qquad \text{and} \\
B_{\mathrm{rel,ecc}} &= \frac{B_{\mathrm{rel,con}}(1 - F_e)}{S_e \left(1 + \frac{A_{\mathrm{rel,c}}}{a f^{\mathrm{isom}}}\right)}.
\end{aligned}$$

With (2.47), for given length l^{CE}, velocity $\dot{l}^{\mathrm{CE}}$ and activation a of a CE, its force f^{CE} is determined. In Fig. 2.3 (right), the velocity dependent CE force (2.47) is exemplarily displayed.

2.2.2.2 The parallel elastic element (PEE)

The PEE describes the passive tissue properties of the muscle belly when externally stretched. The PEE force is modelled by the non-linear spring-like law

$$f^{\mathrm{PEE}}(l^{\mathrm{CE}}) = \begin{cases} 0 & , l^{\mathrm{CE}} < l_0^{\mathrm{PEE}} \\ k^{\mathrm{PEE}}(l^{\mathrm{CE}} - l_0^{\mathrm{PEE}})^{\nu^{\mathrm{PEE}}} & , l^{\mathrm{CE}} \geq l_0^{\mathrm{PEE}} \end{cases}, \tag{2.48}$$

with $k^{\mathrm{PEE}} = f^{\mathrm{PEE}} \frac{f_{\mathrm{max}}^{\mathrm{CE}}}{(l_{\mathrm{opt}}^{\mathrm{CE}}(\Delta W_{\mathrm{des}} + 1 - l_0^{\mathrm{PEE}}))^{\nu^{\mathrm{PEE}}}}$.

In Fig. 2.3 (left), the length dependent PEE force is displayed alongside with the isometric CE force (2.46).

2.2.2.3 The serial elastic element (SEE)

The SEE of a MTU describes the force of the aponeuroses and the tendons that connect the muscle belly (CE) to the bones. Below its tendon slack length l_0^{SEE}, the SEE force is zero ($f^{\mathrm{SEE}} = 0$). Shortly after the transition ($l_0^{\mathrm{SEE}} < l^{\mathrm{SEE}} < l_{\mathrm{nll}}^{\mathrm{SEE}}$), the tendon force rises non-linearly until the tendon length reaches $l_{\mathrm{nll}}^{\mathrm{SEE}}$. After that the force of the SEE is linearly dependent on its length l^{SEE}:

$$f^{\mathrm{SEE}}(l^{\mathrm{SEE}}) = \begin{cases} 0 & , l^{\mathrm{SEE}} \leq l_0^{\mathrm{SEE}} \\ k_{\mathrm{nl}}^{\mathrm{SEE}}(l^{\mathrm{SEE}} - l_0^{\mathrm{SEE}})^{\nu^{\mathrm{SEE}}} & , l^{\mathrm{SEE}} < l_{\mathrm{nll}}^{\mathrm{SEE}} \\ \Delta f_0^{\mathrm{SEE}} + k_l^{\mathrm{SEE}}(l^{\mathrm{SEE}} - l_{\mathrm{nll}}^{\mathrm{SEE}}) & , l^{\mathrm{SEE}} \geq l_{\mathrm{nll}}^{\mathrm{SEE}} \end{cases}. \tag{2.49}$$

2.2.2.4 The serial damping element (SDE)

The SDE is a non-linear, viscous damping element of the MTU model, dependent on the current force f^{MTU} of the whole muscle and on the contraction velocities $\dot{l}^{\mathrm{CE}}$ and $\dot{l}^{\mathrm{MTU}}$. The force of the SDE is modelled by

$$f^{\mathrm{SDE}}(l^{\mathrm{CE}}, \dot{l}^{\mathrm{CE}}, \dot{l}^{\mathrm{MTU}}) = \\ D_{\max}^{\mathrm{SDE}} \cdot \left((1 - R^{\mathrm{SDE}}) \frac{f^{\mathrm{CE}}(l^{\mathrm{CE}}, \dot{l}^{\mathrm{CE}}, a_k) + f^{\mathrm{PEE}}(l^{\mathrm{CE}})}{f_{\max}^{\mathrm{CE}}} + R^{\mathrm{SDE}} \right) \cdot \left(\dot{l}^{\mathrm{MTU}} - \dot{l}^{\mathrm{CE}} \right), \tag{2.50}$$

with $D_{\max}^{\mathrm{SDE}} = D^{\mathrm{SDE}} \cdot \frac{f_{\max}^{\mathrm{CE}} \cdot A_{\mathrm{rel}}}{l_{\mathrm{opt}}^{\mathrm{CE}} \cdot B_{\mathrm{rel}}}$.

2.2.3 Low-level control of muscle length

For approaching possibilities to control a muscle, a deeper look in the physiology of a muscle, the affected proprioceptive sensor organs and the neural circuitry in the spinal cord is needed. In this section, a principle of controlling the length of a muscle is investigated that is based on the mono-synaptic reflex, in which sensor signals of the muscle spindle are fed back to the pool of α-motoneurons that activate the whole muscle. This most basic principle also forms the basis of the control architecture that is designed later, as the architecture will provide the required input signals for this low-level length controller.

A muscle consists of different types of muscle fibres, where in this work the distinction between *extra-fusal* and *intra-fusal* fibres will suffice. *Extra-fusal* fibres are the 'main' muscle fibres that, when activated by the pool of α-motoneurons, contract the whole muscle belly (CE). *Intra-fusal* muscle fibres are connected in parallel to the *extra-fusal* fibres but are activated by the pool of γ-motoneurons and additionally contain the *muscle-spindle* sensor organs (Mileusnic et al., 2006). *Muscle-spindles* are stretch sensitive sensor organs that, when stretched by a contraction of the *intra-fusal* fibres, transmit a neural excitation to the pool of α-motoneurons. This eventually leads to a contraction of the *extra-fusal* fibres, which reduces the strain of the *intra-fusal* fibres and thereby lowers the afferent firing of the muscle-spindles. Thus, the length of the whole muscle can be adjusted by an activation of the *intra-fusal* fibres by the γ-motoneurons. This is the most simple neural connection of afferent sensor signals to the excitation of muscle fibres, also known as the *mono-synaptic reflex* (Haus, 2014; Feldman, 1974; Matthews, 1959). A simplified neural circuitry of this mono-synaptic reflex is illustrated in Fig. 2.4. In a real biological body, additional neurons, such as *inhibitory interneurons* or *Renshaw cells* are connected to the mono-synaptic reflex in the spinal cord. In this thesis, those neurons and cells are not explicitly considered.

Based on the physiology of the neural feedback loop described above, mathematical formulations have emerged with the goal of synthesizing muscle-driven motions. A most simple implementation of this feedback control mechanism is known in the literature as λ-controller, which has proven to be a practicable approach for the forward-dynamics synthesis of biological movements (Kistemaker et al., 2006; Bayer et al., 2017; Günther and Ruder,

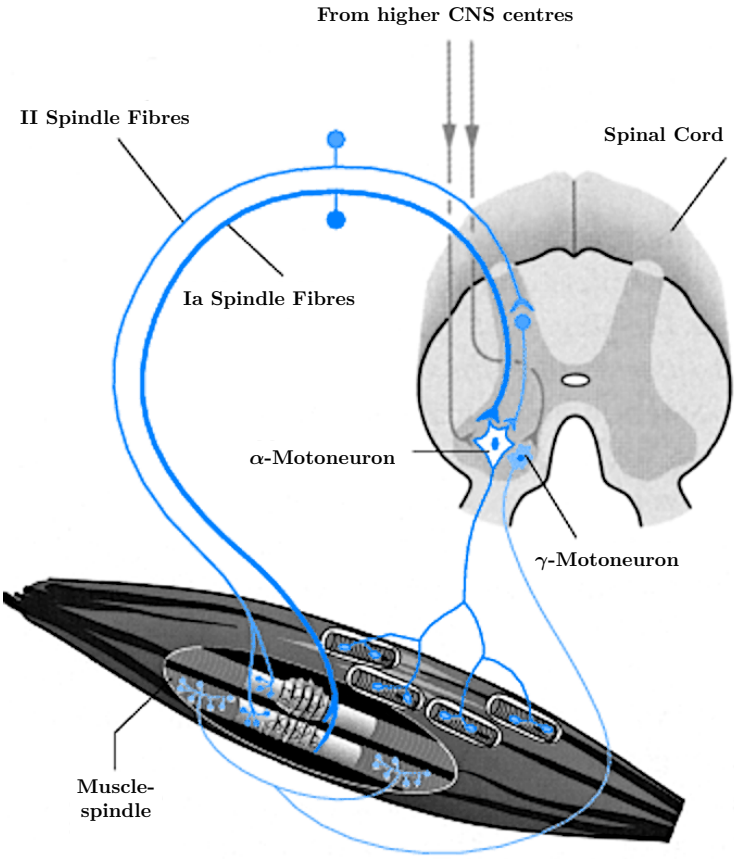


Figure 2.4: Schematic illustration of the mono-synaptic reflex arc (Haus, 2014). Embedded within the *intra-fusal* muscle fibres are the proprioceptive, stretch-sensitive, muscle spindle sensor organs. When the intra-fusal fibres are activated via a pool of γ-motoneurons, the strain on the spindles increases. The afferent respond signals from the spindle are fed back via Ia- and II-fibres to the pool of α-motoneurons. As a consequent the extra-fusal fibres are activated and the whole muscle contracts. This length-threshold feedback mechanism can be formulated in a control law of CE length of an MTU Matthews (1959); Feldman (1974); Kistemaker et al. (2006); McIntyre and Bizzi (1993).

2003). The symbol λ represents the length threshold of a muscle's stretch reflex (Feldman, 1974), which can be adjusted by the excitation (and inhibition) of the intra-fusal fibres via the γ-motoneurons (Matthews, 1959). This is reflected in Fig. 2.1 by the arrow labelled λ pointing towards the 'λ-Controller' block.

From a system-theoretical point of view, the λ-control law has the form of a PD controller on the CE length of a single MTU. (Kistemaker et al., 2006; McIntyre and Bizzi, 1993). The input of this controller is the desired CE length $\lambda \in \mathbb{R}$, which is compared to the actual CE length l^{CE} from (2.40) to form the control error $l^{\mathrm{err}}(t) = l^{\mathrm{CE}}(t) - \lambda(t)$. When the afferent, neural delay $\delta_\lambda^{\mathrm{a}}$ is considered, the control error is defined as

$$l^{\mathrm{err}}_{\delta_\lambda^{\mathrm{a}}}(t) := l^{\mathrm{CE}}(t - \delta_\lambda^{\mathrm{a}}) - \lambda(t). \tag{2.51}$$

The neural delay $\delta_\lambda^{\mathrm{a}}$ describes the time it takes for the sensor system to capture and transmit the signal. With this control error, the λ-controller feeds back the closed-loop controlled

stimulation signal

$$u^{\lambda}(t) = p_{\lambda} \cdot l^{\mathrm{err}}_{\delta^{\mathrm{a}}_{\lambda}}(t) + d_{\lambda} \cdot \dot{l}^{\mathrm{err}}_{\delta^{\mathrm{a}}_{\lambda}}(t), \tag{2.52}$$

with $u^{\lambda} \in \mathbb{R}$ that aims to activate the CE towards the desired length λ. This closed-loop feedback controller can be sufficient for the generation of asymptotically stable postures even in the presence of an external force (e.g. gravity) to which a remaining constant control deviation corresponds. This can be clearly seen, as $u^{\lambda} \to 0$ for $l^{\mathrm{CE}} \to \lambda$, which yields $f^{\mathrm{CE}} \to 0$. For still reaching a desired posture, i.e. compensating the control deviation, an additional 'open-loop' stimulation signal $u^{\mathrm{opn}}(t) \in \mathbb{R}$ can be deployed. Such compensating signals may be provided in a biological system via the pool of α-motoneurons adding contributions to the MTU stimulations that are based on memory or state knowledge in higher centres of the biological motor control system.

The combination of closed-loop λ-control stimulation u^{λ} according to (2.52) and open-loop stimulation u^{opn} has been shown to be capable of generating simple coordinated movements and is also termed 'hybrid controller' of muscle-based systems (Bayer et al., 2017; Kistemaker et al., 2006):

$$u(t) = u^{\lambda}(t) + u^{\mathrm{opn}}(t). \tag{2.53}$$

Note, that in (2.53) only the total resulting stimulation $u(t)$ is restricted to $\mathbb{R}_1$ (see (2.37)). The stimulation contributions $u^{\lambda}(t)$ and $u^{\mathrm{opn}}(t)$ can potentially be negative, for example to capture an inhibitory effect in an *agonistic-antagonistic setup* (AAS). When considering the efferent neural delay $\delta^{\mathrm{e}}_{\lambda}$, which describes the time it takes to transmit the stimulation signal to the MTU, this stimulation writes

$$u_{\delta^{\mathrm{e}}_{\lambda}}(t) = u(t - \delta^{\mathrm{e}}_{\lambda}) \tag{2.54}$$

when 'arriving' at the motor units within the modelled muscle. This signal describes the input to the activation dynamics model from Sec. 2.2.1. The notation of afferent or efferent latencies by the subscript $\delta^{\mathrm{a/e}}_{\lambda}$ is w.l.o.g. mostly omitted throughout this thesis for the sake of readability. In Fig. 2.1, a signals attribute (afferent/efferent) and the respective latency within the 'hybrid controller' is displayed in the grey boxes. According to this, each signal can be considered delayed when attributed with the subscript $\delta^{\mathrm{a}}_{\lambda}$ or $\delta^{\mathrm{e}}_{\lambda}$.

The 'hybrid controller', as it is described above, forms a basis of the hierarchical control architecture that is designed later in Pt. II. There it is implemented in the lowest layer in the cascaded hierarchy as a 'low-level' controller.

2.2.4 Moment arms

The forces of a muscle are transduced to the skeletal system by its attachments via tendons to the bones and by the deflection surfaces that wrap the muscle tissue around the joints. This corresponds to transforming the force of a muscle at a respective point to a torque at the joint in the direction of its DoF as described in Fig. 2.1 by the block 'Moment Arms'.

In the used simulation framework `demoa` the MTU routing is achieved 'via-ellipses' according to Hammer et al. (2019). With this method, the 1D, string like MTU is attached to the skeletal system by an 'origin' and an 'insertion' attachment point and the MTU force is applied to the respective bodies at those points. The general routing of the MTU is achieved by the definition of deflection ellipses. When inside an ellipse, the MTU is not affected by the deflection at all. When the MTU string is spanned to touch an edge of an ellipse, it is deflected at the point of contact and transduces a deflecting force to the body of the ellipse. This deflection method models the 3-dimensional circumference of a muscle and allows a realistic representation of the non-linear, joint angle dependent moment arms of a muscle. For details on the mathematical description of this deflection method it is referred to Hammer et al. (2019).

The resulting torque on a DoF (subscript i) that is generated by a MTU (subscript j) can be calculated by the sum of all forces of the MTU $\boldsymbol{f}_{jp}^{\mathrm{MTU}} \in \mathbb{R}^3$ at the attachment or deflection points (subscript p) with the respective projected perpendicular distance to the DoF $\boldsymbol{r}_{pi} \in \mathbb{R}^3$ by

$$\tau_{ji}^{\mathrm{MTU}} = \sum_p \boldsymbol{f}_{jp}^{\mathrm{MTU}} \times \boldsymbol{r}_{pi}.$$

Alternatively, the total, scalar (in MTU string direction) MTU force f_j^{MTU} is transformed to a joint torque τ_i^{MTU} directly by an equivalent scalar moment arm $r_{ji}^{\mathrm{MTU}} \in \mathbb{R}$:

$$\tau_{ji}^{\mathrm{MTU}} = f_j^{\mathrm{MTU}} \cdot r_{ji}^{\mathrm{MTU}}.$$

This equivalent moment arm r_{ji}^{MTU} is the perpendicular distance of the joint point to the MTU string, projected onto the plane that is perpendicular to the DoF axis $\boldsymbol{\omega}$ from (2.20). It is defined by

$$r_{ji}^{\mathrm{MTU}} := \frac{\partial \theta_i}{\partial l_j^{\mathrm{MTU}}},$$

where θ_i is the joint angle of the DoF that is embraced by the muscle with total length l_j^{MTU}. It is important to note, that a MTU can be spanned over multiple DoFs and that each DoF may be actuated by multiple MTUs. The definition of the MTUs' moment arms can therefore easily be generalized into a matrix equation. For this, consider the set of joint angles $\boldsymbol{\theta} \subseteq \boldsymbol{q}$ as a subset $\boldsymbol{\theta} \in \mathbb{R}^{\bar{n}_{\mathrm{DoF}}}$ of the $\bar{n}_{\mathrm{DoF}}$ DoFs of the mechanical system (2.33) that are actuated by MTUs. The vector $\boldsymbol{f}^{\mathrm{MTU}} \in \mathbb{R}^{n_{\mathrm{MTU}}}$ of all n_{MTU} MTU forces generates the torques $\boldsymbol{\tau}^{\mathrm{MTU}} \in \mathbb{R}^{\bar{n}_{\mathrm{DoF}}}$ that act on the $\bar{n}_{\mathrm{DoF}}$ DoFs by the matrix of moment arms

$$R := \tfrac{\partial \boldsymbol{\theta}}{\boldsymbol{l}^{\mathrm{MTU}}} \in \mathbb{R}^{\bar{n}_{\mathrm{DoF}} \times n_{\mathrm{MTU}}}. \tag{2.55}$$

The generalized forces $\boldsymbol{\tau}^{\mathrm{MTU}}$ that act on the DoFs can then be calculated by (Walter et al., 2021a; Stanev and Moustakas, 2019):

$$\boldsymbol{\tau}^{\mathrm{MTU}} = -R^T(\boldsymbol{\theta}) \boldsymbol{f}^{\mathrm{MTU}}. \tag{2.56}$$

Here, the negative sign is chosen in accordance to Stanev and Moustakas (2019). However, it must be emphasised that a set of conventions for the physical variables (l^{MTU}, θ, f^{MTU} and τ^{MTU}) determine what sign each component of the matrix of moment arms $R(\boldsymbol{\theta})$ eventually carries.

For the simulation studies in Pt. III of this thesis a novel approach for calculating the moment arms is used that makes use of a graph-theory based description of the musculoskeletal system. The graph based system description is introduced in Sec. 4.1 and the most relevant source-code for the moment arms calculation is given in the Lsts. E.1 and E.2 in Appendix E.5.

2.3 Joint-limitations and visco-elastic forces

The (generalized) joint-limitation forces (torques) $\boldsymbol{\tau}_j^{\mathrm{lmt}}$ of the j-th rotational DoF q_j are modelled as linear, one-sided, spring-damper elements that act directly on the respective DoF:

$$\tau_j^{\mathrm{lmt}} = \begin{cases} k_{\mathrm{l},j}^{\mathrm{lmt}} \cdot (q_j - q_{\mathrm{l},j}) + d_{\mathrm{l},j}^{\mathrm{lmt}} \cdot \dot{q}_j & , q_j < q_{\mathrm{l},j} \\ 0 & , q_{\mathrm{l},j} \leq q_j \leq q_{\mathrm{u},j} \\ k_{\mathrm{u},j}^{\mathrm{lmt}} \cdot (q_j - q_{\mathrm{u},j}) + d_{\mathrm{u},j}^{\mathrm{lmt}} \cdot \dot{q}_j & , q_j > q_{\mathrm{u},j} \end{cases} \tag{2.57}$$

with the lower and upper (index l/u) threshold angles $q_{\mathrm{l/u},j}$ and the respective linear spring and damping parameters $k_{\mathrm{l/u},j}^{\mathrm{lmt}}$ and $d_{\mathrm{l/u},j}^{\mathrm{lmt}}$.

The visco-elastic (bushing) forces τ_j^{bsh}, which are used as a description for passive visco-elastic tissue around the joint (e.g. ligaments), are modelled with the same force law in the special case of $q_{\mathrm{l},j} = q_{\mathrm{u},j} = 0$, $k_{\mathrm{l},j}^{\mathrm{bsh}} = k_{\mathrm{u},j}^{\mathrm{bsh}} = k_j^{\mathrm{bsh}}$, and $d_{\mathrm{l},j}^{\mathrm{bsh}} = d_{\mathrm{u},j}^{\mathrm{bsh}} = d_j^{\mathrm{bsh}}$. With this, the visco-elastic force model for the j-th DoF contracts to

$$\tau_j^{\mathrm{bsh}} = k_j^{\mathrm{bsh}} \cdot q_j + d_j^{\mathrm{bsh}} \cdot \dot{q}_j. \tag{2.58}$$

The parameter values for the joint limitations and the visco-elastic force elements are given in Appendix C.

2.4 Contact forces

Contacts of mechanically driven rigid bodies occur when their spatial dimensions overlap. Forces develop at the point of interaction, which are dependent on the bodies' mechanical states and their material properties. When contacting rigid bodies move relative to each other, their contact can be in the state of *slipping* or *sticking*. This state is thereby dependent on the magnitude of the force that is transduced at the contact point and of its direction. Altogether, to realistically model a contact of two rigid bodies in a forward dynamics simulation, complex contact models are required, especially when considering arbitrary surface geometries of the bodies involved. For this, physics-based laws for contacts are used as a basis for an algorithmic implementation. Although the definition of rigid bodies (section 2.1) demands rigid bodies not to be deformable, a 'soft-parametrised' contact definition can be used to mimic this effect.

In the used simulation environment `demoa`, a contact model that captures state-dependent *stick/slip* transitions, as well as parameters for the material properties is implemented (Henze, 2002; Walter et al., 2021a). Refer to Appendix B for a description of the contact algorithm. In this contact model, a point on a rigid body can engage to contact a spatially restricted plane on another body. By modelling multiple contact points and respective planes, for simple geometries a realistic interaction with the environment can be simulated. The contact forces are subsequently transformed to their equivalent generalised forces, i.e. joint torques. As an external force they are an input to the mechanical equations of motion (2.33) as described in Sec. 2.1.

Part II

The Hierarchical Control Architecture

Design of the hierarchical control architecture 3

As biological movement is driven by stimulated muscles, both the movement and the goal of the task are implicitly present in the coordinate space of muscle stimulations. This does not imply that the whole movement is planned in terms of muscle stimulation coordinates, and it is worth considering that planning may be done significantly easier in terms of a more suitable coordinate space. Even more, for different movements, different control spaces may be better suited. For example, a pointing or grasping task may be best described in the 3D, spatial coordinates of the hand and maintaining balance in upright standing is feasible in the coordinates of joint torques (Walter et al., 2021a). For complex and/or combined movements, multiple of such control tasks can be active in parallel, where each has it's individual conceptional planning space (Walter et al., 2021b). The number of stimulation signals generated is typically larger than the size of the planning space. This requires a transformation of the movement plan into muscle stimulations. In such a transformation the redundancy of many muscles acting on a smaller set of joints is resolved (Walter et al., 2021a). The control architecture that is designed in the following therefore assumes biological motor control as a layered, hierarchically-structured system. This was also already proposed by Wolpert (1997); Prescott et al. (1999); Todorov et al. (2005); DeWolf and Eliasmith (2011); Sober and Sabes (2005) and Merel et al. (2019). In the present work, three hierarchical layers of control are identified, namely, the 'structural layer', the 'transformational layer' and the 'conceptional layer' (Fig. 3.1).

The structural layer represents the direct wiring of both the efferent motor neurons from the spinal cord to the neuronal endplate and the afferent sensory neurons from the muscle receptor organs to the spinal cord, as well as the interconnections of these using one or very few synaptic connections, i. e. the mono-synaptic reflex loop.

The conceptional layer is an assumed, lumped layer representing the whole brain and complex spinal cord functions, including the crucial building blocks of motion planning. Among them are motor prediction (Wolpert and Flanagan, 2001), haptic perceptions (Blakemore et al., 1999), motor learning (Tseng et al., 2007), movement intention (Ganesh et al., 2018), task and environment context (Wolpert et al., 2003), body representation (Naito et al., 2016), and sensory predictions (Gentili et al., 2010). In the design of the control architecture presented here, the output of this layer is a postural plan in the form of a desired angle for each body joint. This is symbolised by the arrow labelled 'postural plan' in Fig. 3.1. Additionally, co-contractions for the muscles and the joints are provided by this layer. In this work, the conceptional layer is implemented in a most simple representation of *proportional-integral-derivative* (PID) feedback controllers and subsequent transformations from the conceptional coordinate spaces of joint torques and of limb positions to form the

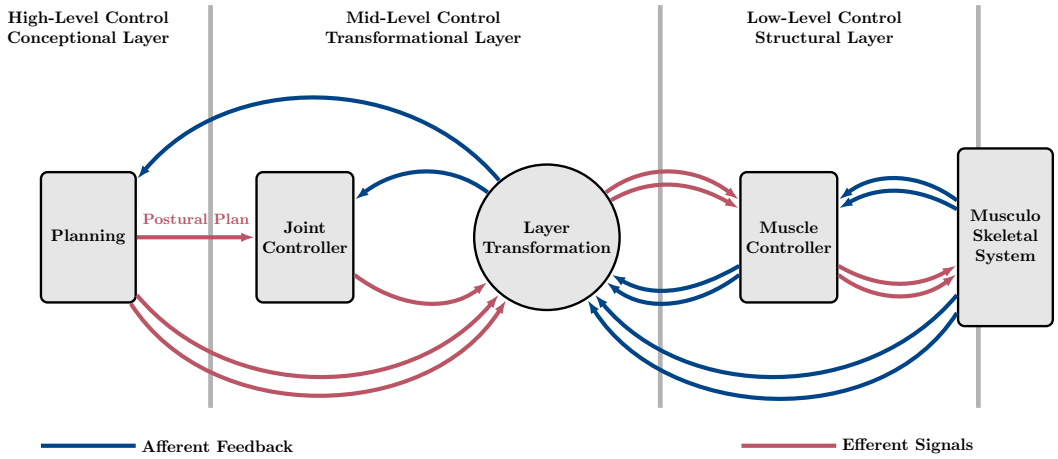


Figure 3.1: Overview of the hierarchical structure of the presented control architecture. The result of a conceptional planning process is transformed to the transformational layer in the form of a postural plan and of muscle and joint co-contraction. The transformational layer translates the postural plan to eventually produce muscle stimulations and other control related signals that are passed on to the structural layer. Thereby the muscle-joint redundancy is resolved. The signal dimensionality is displayed by single (low dimensional) and multiple (high dimensional) lines.

postural plan.

In between the conceptional and the structural layer, the transformational layer is defined. It serves the main function of reducing the cognitive load on higher CNS centres, such as the brain and the higher spinal cord. Cognitive load in this sense is, e.g. total neuronal information communicated between the periphery and the brain, control effort (Haeufle et al., 2014b) and storage of knowledge about local material characteristics and geometry, e.g. muscle moment arms and limb dimensions. The transformation, as presented here, is done purely by exploiting the geometry and actuator characteristics, namely, the morphology. That is, adding this layer reduces computational costs of higher CNS parts by increasing the morphological intelligence (Ghazi-Zahedi, 2019) on lower to mid levels of the control structure. The transformational layer synergises several distributed local control parts of the structural layer by using a more centralised knowledge about these parts. In turn, this centralisation enables time- and signal-efficient processing of sensory information of the structural layer. Due to the standardised input of a 'postural plan' and the communication with the structural layer, the transformational layer is the key part of the presented hierarchical control architecture. It enables a modular design that allows to synthesise complex movements with multiple combined movement tasks, each with an individual conceptional planning space.

To obtain a mathematical formulation of the hierarchical control architecture that enables an algorithmic implementation, in the following sections its design process is elaborated from the lowest to the highest layer. Following this bottom-up approach, the structural, transformational and conceptional layers are presented in such a way that closes each control loop in a hierarchical order. This also includes the formulation of the respective Jacobian based layer transformations to resolve the redundancy. The design of each layer identifies standardised inputs and outputs, leading to a modular design that is flexible in terms of conceptional control inputs and guarantees the generation of appropriate MTU stimulations. Furthermore, in this hierarchical setup, the motion planning occurs in the conceptional layer only, meaning that only the conceptional coordinates have to be considered

in the planning process and nothing has to be known about the biological structures. The incorporation of those is achieved subsequently in the autonomous transformation process of the architecture that generates MTU stimulations based on the conceptional plan. More precisely, the Jacobian matrices are calculated based on biophysical features of the musculoskeletal model, most prominently the geometric implementations and internal stiffness relations of the MTUs. The Jacobian matrices are moreover derived in closed-form, making the required biophysical information apparent that is needed for resolving the redundancy.

In very short words, the control architecture performs as follows: The structural layer produces MTU stimulations based on the low-level CE-length controller (hybrid controller, see Sec. 2.2.3) with the respective inputs of desired CE-length $\boldsymbol{\lambda}$ and task-fulfilling, 'open-loop' stimulations $\boldsymbol{u}_{\text{task}}^{\text{opn}}$. Exactly those inputs are provided by the transformational layer as a reinterpretation of it's own input, the postural plan $\boldsymbol{\theta}^{\text{des}}$, which itself is provided by the conceptional layer as a result of the planning process. On each layer, PID-controllers are implemented to ensure a stable and robust task fulfilment and Jacobian matrices are used for the transformation of the controller's outputs to the respective lower layer. Picking up this consideration and applying it in a straightforward way, in the following sections each layer is described in detail and an essentially plain concept of planning and then generating coordinated movements is laid out. That is, in Sec. 3.1, the structural layer and in the Secs. 3.2 and 3.3 the transformational and the conceptional layers are introduced. Beyond the generation of coordinated movement, the biological feature of redundancy opens up a 'uncontrolled manifold' to fulfil additional criteria, such as a joint wide co-contraction. This concept of joint wide co-contraction is part of the transformational layer and is described in Sec. 3.2.4. A synopsis of the algorithmic implementation of the hierarchical control architecture in the forward dynamics simulation software (`C/C++`) is given in Appendix E.

3.1 The Structural Layer

The structural layer is perceived here as the space that contains the representations of the muscles, associated biological structures (e.g., muscle spindles) and their mono-synaptic neural connection within the spinal cord. The mathematical models that constitute this layer are those of the MTU model from section 2.2, including it's activation dynamics from Section 2.2.1 and the low-level hybrid controller from Section 2.2.3. Within the control framework, these models are written in a generalised vector/matrix notation that allows a comprehensive formulation of the control system. The space of the structural layer in general covers the whole set of MTUs and has a dimensionality of n_{MTU}. This space can be reduced to a subset of only a part of the whole model with dimension $\tilde{n}_{\text{MTU}} \leq n_{\text{MTU}}$ to, e.g. only consider the control of a single limb. Throughout the design of the control architecture in this chapter and w.l.o.g. the whole space of MTUs is used.

In vector notation, the total neural input $\boldsymbol{u} = [u_1, ..., u_{n_{\text{MTU}}}]^T \in \mathbb{R}_1^{n_{\text{MTU}}}$ of the MTUs is defined analogously to (2.53) and in accordance with Kistemaker et al. (2006); Bayer et al. (2017) as the sum of a 'closed-loop' stimulation contribution $\boldsymbol{u}^{\lambda} \in \mathbb{R}^{n_{\text{MTU}}}$ and an 'open-loop' contribution $\boldsymbol{u}^{\text{opn}} \in \mathbb{R}^{n_{\text{MTU}}}$:

$$\boldsymbol{u}(t) = \boldsymbol{u}^{\lambda}(t) + \boldsymbol{u}^{\text{opn}}(t). \tag{3.1}$$

By defining a vector of efferent neural latency times $\boldsymbol{\delta}_{\text{SL}}^{e} := \left[\delta_{\text{SL},1}^{e}, ..., \delta_{\text{SL},n_{\text{MTU}}}^{e}\right]^T \in \mathbb{R}^{n_{\text{MTU}}}$ (see also (2.54) for the scalar case), the delayed vector of MTU stimulations is given by

$$\boldsymbol{u}_{\delta_{\text{SL}}^{e}}(t) = \begin{bmatrix} u_1(t - \delta_{\text{SL},1}^{e}) \\ \dots \\ u_{n_{\text{MTU}}}(t - \delta_{\text{SL},n_{\text{MTU}}}^{e}) \end{bmatrix} \in \mathbb{R}_1^{n_{\text{MTU}}}. \tag{3.2}$$

That is, each stimulation is delayed by it's respective efferent, neural latency time $\delta_{\text{SL},i}^{\text{e}}$ For the sake of notation and w.l.o.g. the subscript of δ_{SL}^{e} is mostly omitted in the following.

Simply by adding this subscript, an efferent signal from the structural layer can be considered to be delayed by $\boldsymbol{\delta}^{e}_{\mathrm{SL}}$. The same holds for afferent sensor signals with the respective latency time (and subscript) $\boldsymbol{\delta}^{a}_{\mathrm{SL}}$ that it takes to transmit the signal to the structural layer. The attribute of a signal to be an afferent or an efferent signal is displayed in the Figs. 2.1, 3.2 and 3.4 by an arrow crossing the border to the right of the structural layer.

In this structural layer, for each MTU, a mathematical representation of a low-level feedback control mechanism (2.52) on the spinal level is implemented. The structural embodiment of this feedback control mechanism is the mono-synaptic spinal cord reflex arc that transmits muscle spindle signals from the intrafusal muscle fibres via a pool of α-motoneurons to the extrafusal fibres, as described in Sec. 2.2.3. The biophysical control-related signals within this layer are completely in accordance with Section 2.2.3 and include the vector of muscle stimulation signals $\boldsymbol{u} \in \mathbb{R}_1^{n_{\mathrm{MTU}}}$ from (3.1), which is the layer's output that serves as neural input to the muscles' activation dynamics (2.38), and the vector of CE lengths $\boldsymbol{l}^{\mathrm{CE}} = \left[l_1^{\mathrm{CE}}, ..., l_{n_{\mathrm{MTU}}}^{\mathrm{CE}}\right]^T \in \mathbb{R}^{n_{\mathrm{MTU}}}$ being the state variables of the muscles' contraction dynamics. Together with the vector of desired CE lengths $\boldsymbol{\lambda} \in \mathbb{R}^{n_{\mathrm{MTU}}}$, the control error

$$\boldsymbol{l}^{\mathrm{err}}(t) := \boldsymbol{l}^{\mathrm{CE}}(t) - \boldsymbol{\lambda}(t) \in \mathbb{R}^{n_{\mathrm{MTU}}} \tag{3.3}$$

is defined in accordance to (2.51). The closed loop stimulation contribution in it's vector form then follows from (2.52):

$$\boldsymbol{u}^{\lambda}(t) = P_\lambda \cdot \boldsymbol{l}^{\mathrm{err}}_{\boldsymbol{\delta}_{\boldsymbol{\lambda}}}(t) + D_\lambda \cdot \dot{\boldsymbol{l}}^{\mathrm{err}}_{\boldsymbol{\delta}_{\boldsymbol{\lambda}}}(t), \tag{3.4}$$

where the diagonal control matrices $P_\lambda = \mathrm{diag}(p_{\lambda,1}, \dots p_{\lambda,n_{\mathrm{MTU}}}) \in \mathbb{R}^{n_{\mathrm{MTU}} \times n_{\mathrm{MTU}}}$ and $D_\lambda = \mathrm{diag}(d_{\lambda,1}, \dots d_{\lambda,n_{\mathrm{MTU}}}) \in \mathbb{R}^{n_{\mathrm{MTU}} \times n_{\mathrm{MTU}}}$ contain the control parameters $p_{\lambda,i} > 0$ and $d_{\lambda,i} > 0$, respectively. The vector of desired CE lengths $\boldsymbol{\lambda}$ is hereby an input of the structural layer of dimension n_{MTU}.

The additional 'open-loop' stimulation signal $\boldsymbol{u}^{\mathrm{opn}}(t) = \left[u_1^{\mathrm{opn}}, ..., u_{n_{\mathrm{MTU}}}^{\mathrm{opn}}\right]^T \in \mathbb{R}^{n_{\mathrm{MTU}}}$ from (3.1) is the second input to the structural layer. Both inputs, $\boldsymbol{\lambda} \in \mathbb{R}^{n_{\mathrm{MTU}}}$ and $\boldsymbol{u}^{\mathrm{opn}} \in \mathbb{R}^{n_{\mathrm{MTU}}}$ must be chosen in accordance to each other to fulfil a movement task. That is, a total bandwidth of $2 \cdot n_{\mathrm{MTU}}$ signals need to be provided. To reduce this signal bandwidth and thereby the cognitive load on higher CNS centres that is required for planning, the inputs of the structural layer are provided by the lower-dimensional transformational layer that is constructed in the following.

3.2 The transformational Layer

The transformational layer is implemented one stage higher in the control hierarchy than the structural layer from the previous section. It shifts the control space to the coordinates of joint angles of the model and produces synergistic input for all muscles that act on the same joint. In the transformation from the transformational to the structural layer, the muscle-joint redundancy is resolved. This is achieved by exploiting characteristic knowledge about the morphology of the musculo-skeletal system, namely, moment arms and tissue stiffness. Once these characteristics are implemented, a movement plan can be formulated in the control space of joint angles and subsequently transformed to actuate the system. This reduces the total cognitive load of the planning process, simply as the space of joint angles is of lower dimension than the space of the MTUs. Additionally, the same characteristics can be further exploited to fulfil the same movement task under different levels of a joint-wide co-contraction (Sec 3.2.4).

Throughout this section, the transformational layer is assumed to act on the whole set of MTU-actuated joints $\boldsymbol{\theta} \subseteq \boldsymbol{q}$ with $\boldsymbol{\theta} \in \mathbb{R}^{n_\theta}$ (compare to (2.33)). Similar as outlined in Section 3.1, also only a subset $\tilde{\boldsymbol{\theta}} \subset \boldsymbol{\theta}$ with $\tilde{\boldsymbol{\theta}} \in \mathbb{R}^{\tilde{n}_\theta}$ can be considered for control. The transformation of a subset of the joint angles to the structural layer projects exactly in the

space of those muscles that actuate the respective joint, i.e., a well defined structural layer emerges naturally. For the sake of notation and w.l.o.g., the whole set of MTU-actuated joints $\boldsymbol{\theta} \in \mathbb{R}^{n_\theta}$ is considered for control in the following. The neural latencies of afferent and efferent communication with the structural layer follow the same notation as for the structural layer from Sec 3.1 by defining the delay vectors $\boldsymbol{\delta}_{\mathrm{TL}}^{\mathrm{a/e}}$ and introducing the respective subscripts exactly as described in (3.2). A signal's property of being affected by an afferent or an efferent neural delay is displayed in Fig 3.2 by an arrow that crosses the border to the right of the transformational layer.

The mathematical formulation of the transformational layer that is presented in this section is designed such that it exactly provides the inputs of desired CE lengths $\boldsymbol{\lambda} \in \mathbb{R}^{n_{\mathrm{MTU}}}$ and 'open-loop' stimulation $\boldsymbol{u}^{\mathrm{opn}} \in \mathbb{R}^{n_{\mathrm{MTU}}}$ that are required by the structural layer. Each of these signals is generated in the transformational layer separately by a 'mid-level' PID-controller on joint angles $\boldsymbol{\theta} \in \mathbb{R}^{n_\theta}$ and a subsequent transformation that is based on a Jacobian-matrix. Both PID controllers thereby obtain the same input of a postural plan $\boldsymbol{\theta}^{\mathrm{des}} \in \mathbb{R}^{n_\theta}$, which also forms the main input to the transformational layer itself. In Fig. 3.2 both mid-level controllers and the hierarchical cascade of the transformational and the structural layer is displayed.

The first of the two mid-level controllers is termed θ_λ-controller (for details, see Sec. 3.2.1). It takes the desired state in terms of the postural plan $\boldsymbol{\theta}^{\mathrm{des}} \in \mathbb{R}^{n_\theta}$ and uses the angle-length Jacobian $J^{\lambda\theta} \in \mathbb{R}^{n_{\mathrm{MTU}} \times n_\theta}$, which contains structural knowledge about muscle moment arms and MTU-internal stiffness ratios, to transmit values of desired muscle lengths $\boldsymbol{\lambda}^\theta(t) = \left[\lambda_1^\theta, ..., \lambda_{n_{\mathrm{MTU}}}^\theta\right]^T \in \mathbb{R}^{n_{\mathrm{MTU}}}$ to the associated low-level λ-controller in the structural layer. This can be seen as a re-interpretation of the movement plan in terms of nominal muscle lengths. The nominal lengths $\boldsymbol{\lambda}^\theta$ are prepared to feed the model representation of the mono-synaptic spinal cord reflex arc (λ-controller (3.3,3.4)). The output of the cascaded θ_λ- and λ-controllers, which form a hierarchical control sub-structure of the overall architecture, is the stimulation signal $\boldsymbol{u}^{\theta\lambda}(t) = \left[u_1^{\theta\lambda}, ..., u_{n_{\mathrm{MTU}}}^{\theta\lambda}\right]^T \in \mathbb{R}^{n_{\mathrm{MTU}}}$, which is defined in accordance to (3.3) and (3.4) as $\boldsymbol{u}^{\theta\lambda}(t) := \boldsymbol{u}^\lambda(t, \boldsymbol{\lambda}(t) = \boldsymbol{\lambda}^\theta(t))$.

The second controller is termed the θ-controller (for details, see Sec. 3.2.2). It likewise takes the desired state $\boldsymbol{\theta}^{\mathrm{des}} \in \mathbb{R}^{n_\theta}$ and uses the angle-stimulation Jacobian $J^{u\theta} = J^{u\lambda} . J^{\lambda\theta} \in \mathbb{R}^{n_{\mathrm{MTU}} \times n_\theta}$, which additionally contains steady-state knowledge in $J^{u\lambda} \in \mathbb{R}^{n_{\mathrm{MTU}} \times n_{\mathrm{MTU}}}$ about the stimulation-length relation of the MTU's activation dynamics, to transmit a second task-fulfilling contribution to the MTU stimulations. This contribution is 'open-loop' from the perspective of the structural layer, as bypassing the muscle spindle reflex path, even though 'closed-loop' from the perspective of the transformational layer. It consist of the two stimulation vectors $\boldsymbol{u}^\theta(t) = \left[u_1^\theta, ..., u_{n_{\mathrm{MTU}}}^\theta\right]^T \in \mathbb{R}^{n_{\mathrm{MTU}}}$ and $\boldsymbol{u}_{\mathrm{ref}}^{\mathrm{coc}}(t) = \left[u_{\mathrm{ref},1}^{\mathrm{coc}}, ..., u_{\mathrm{ref},n_{\mathrm{MTU}}}^{\mathrm{coc}}\right]^T \in \mathbb{R}_1^{n_{\mathrm{MTU}}}$, of which the sum is termed

$$\boldsymbol{u}_{\mathrm{task}}^{\mathrm{opn}} = \boldsymbol{u}^\theta + \boldsymbol{u}_{\mathrm{ref}}^{\mathrm{coc}} \tag{3.5}$$

to be still in accordance with (3.1). The two addends to $\boldsymbol{u}_{\mathrm{task}}^{\mathrm{opn}}$ play complementary roles in the task fulfillery: $\boldsymbol{u}_{\mathrm{ref}}^{\mathrm{coc}}$ are arbitrary reference stimulation signals (base contraction levels, for details see Sec. 3.2.1), and $\boldsymbol{u}^\theta$ is set up to minimise the error between $\boldsymbol{u}_{\mathrm{ref}}^{\mathrm{coc}}$ and the desired stimulation values that corresponds to $\boldsymbol{\theta}^{\mathrm{des}}$, namely, $\boldsymbol{u}_{\mathrm{task}}^{\mathrm{opn}}$ itself.

To allow the system to satisfy additional criteria or side conditions along with the movement task, a second co-contraction contribution $\boldsymbol{u}_\theta^{\mathrm{coc}} = [u_{\theta,1}^{\mathrm{coc}}, \ldots, u_{\theta,n_{\mathrm{MTU}}}^{\mathrm{coc}}]^T \in \mathbb{R}^{n_{\mathrm{MTU}}}$ to 'open-loop' MTU stimulation can be provided by the transformational layer within the control architecture. This adjusts the stimulations of all MTUs that act on the same joint exactly without interfering with the fulfilment of the movement task (for details see Section 3.2.4).

Taken together, four contributions of MTU stimulations have been distinguished that feed the two addends in (3.1). The addend $\boldsymbol{u}^\lambda$ identifies with $\boldsymbol{u}^{\theta\lambda}$, and $\boldsymbol{u}^{\mathrm{opn}}$ with $\boldsymbol{u}^\theta + \boldsymbol{u}_{\mathrm{ref}}^{\mathrm{coc}} +$

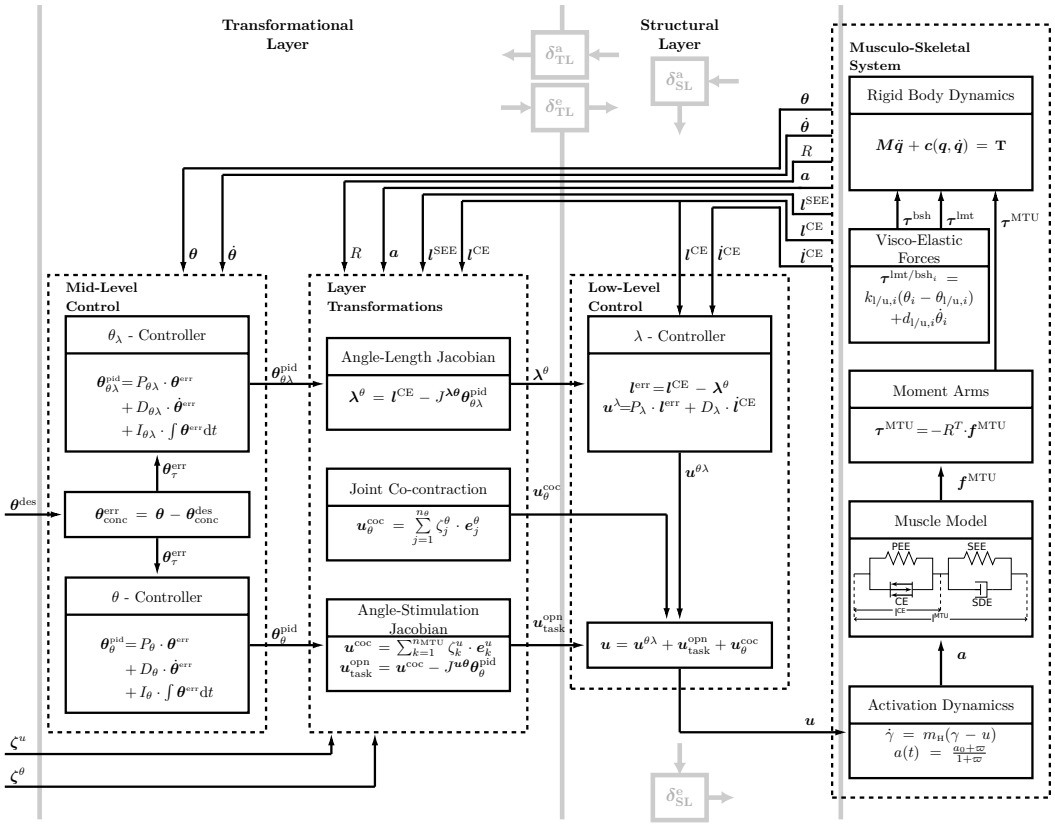


Figure 3.2: Block diagram of the hierarchical cascaded transformational and structural layers. The postural plan $\boldsymbol{\theta}^{\mathrm{des}}$ is the input of the transformational layer and it is processed to form the inputs $\boldsymbol{\lambda}^{\theta}$ and $u^{\mathrm{opn}}_{\mathrm{task}}$ of the structural layer.

$\boldsymbol{u}^{\mathrm{coc}}_{\theta}$:

$$\boldsymbol{u}(t) = \underbrace{\boldsymbol{u}^{\theta\lambda}(t)}_{\widehat{=}\boldsymbol{u}^{\lambda}(t)} + \underbrace{\boldsymbol{u}^{\theta}(t) + \boldsymbol{u}^{\mathrm{coc}}_{\mathrm{ref}}(t) + \boldsymbol{u}^{\mathrm{coc}}_{\theta}}_{\widehat{=}\boldsymbol{u}^{\mathrm{opn}}(t)}. \tag{3.6}$$

The summed stimulation signal in (3.6) constitutes the total output of the proposed hierarchical control architecture, which drives the actuators of the musculo-skeletal system. The details on the calculations of these stimulation signals and their underlying control concepts are given in the following, starting with the hierarchical θ_λ-controller in Sec. 3.2.1, followed by the direct θ-controller including co-contraction in Sec. 3.2.2. The joint co-contraction $\boldsymbol{u}^{\mathrm{coc}}_{\theta}$ and its basis for construction is outlined in Section 3.2.4.

3.2.1 The hierarchical θ_λ-controller

In this section, the mathematical particulars of the θ_λ-controller are outlined. It is described here, how the postural plan $\boldsymbol{\theta}^{\mathrm{des}}$ is interpreted in the transformational layer, and how it is transformed to the structural layer to provide the corresponding desired MTU lengths $\boldsymbol{\lambda}^{\theta}$ that lead to the hierarchically controlled stimulation signal $\boldsymbol{u}^{\theta\lambda}$ in (3.6) (see also Fig. 3.2).

Based on the set of controlled joint angles $\boldsymbol{\theta}(t)$ and the vector of desired joint angles $\boldsymbol{\theta}^{\mathrm{des}}(t)$, firstly, the control error is defined as

$$\boldsymbol{\theta}^{\mathrm{err}}(t) := \boldsymbol{\theta}(t) - \boldsymbol{\theta}^{\mathrm{des}}(t) \in \mathbb{R}^{n_\theta}. \tag{3.7}$$

This error signal describes the deviation of the actual body posture $\boldsymbol{\theta}$ from the desired state $\boldsymbol{\theta}^{\text{des}}$. Subsequently, the PID control law

$$\boldsymbol{\theta}_{\theta_\lambda}^{\text{PID}} := P_{\theta_\lambda} \cdot \boldsymbol{\theta}^{\text{err}} + D_{\theta_\lambda} \cdot \dot{\boldsymbol{\theta}}^{\text{err}} + I_{\theta_\lambda} \cdot \int \boldsymbol{\theta}^{\text{err}} \mathrm{d}t \in \mathbb{R}^{n_\theta} \tag{3.8}$$

is used to create a robust and stabilising output of controlled joint angles, where the matrices $P_{\theta_\lambda} = \text{diag}(p_{\theta_\lambda,1}, \ldots, p_{\theta_\lambda,n_\theta}) \in \mathbb{R}^{n_\theta \times n_\theta}$, $D_{\theta_\lambda} = \text{diag}(d_{\theta_\lambda,1}, \ldots, d_{\theta_\lambda,n_\theta}) \in \mathbb{R}^{n_\theta \times n_\theta}$, and $I_{\theta_\lambda} = \text{diag}(\text{I}_{\theta_\lambda,1}, \ldots, \text{I}_{\theta_\lambda,n_\theta}) \in \mathbb{R}^{n_\theta \times n_\theta}$ are diagonal control matrices, with parameters $p_{\theta_\lambda,i} > 0$, $d_{\theta_\lambda,i} > 0$, and $\text{I}_{\theta_\lambda,i} > 0$. The output $\boldsymbol{\theta}_{\theta_\lambda}^{\text{PID}}(t)$ allows to fix the values of desired CE lengths $\boldsymbol{\lambda}^\theta(t)$ (see (3.13) below). It mainly scales by P_{θ_λ} with the control error signal $\boldsymbol{\theta}^{\text{err}}(t)$, while a modifying angular-rate-dependent by D_{θ_λ} contribution and some memory by $I_{\theta_\lambda} \cdot \int \boldsymbol{\theta}^{\text{err}} \mathrm{d}t$ of the angle error are added, with the latter eliminating any residual control deviations.

To eventually fix the desired CE lengths $\boldsymbol{\lambda}^\theta(t)$ (3.13), the output of the θ_λ-PID control law (3.8) from the transformational layer must be transformed to the structural layer. Instead of transforming the joint angles directly to CE lengths, changes of the joint angles $\partial\boldsymbol{\theta}$ are transformed to changes of muscle lengths $\partial\boldsymbol{l}^{\text{CE}}$, which corresponds to answering the question how much the contractile parts of the muscles need to contract (or relax) to reach $\boldsymbol{\theta}^{\text{des}}$. Such a transformation has already been discussed in literature (Pellionisz and Llinás, 1985) and can be achieved by a Jacobian matrix of the form

$$J^{\lambda\theta} := \frac{\partial \boldsymbol{l}^{\text{CE}}}{\partial \boldsymbol{\theta}} \in \mathbb{R}^{n_{\text{MTU}} \times n_\theta} \quad \Leftrightarrow \quad \partial\boldsymbol{l}^{\text{CE}} = J^{\lambda\theta} \cdot \partial\boldsymbol{\theta}. \tag{3.9}$$

Note, that this angle-length Jacobian has a similar definition as the matrix of MTU moment arms (2.55). In fact, $J^{\lambda\theta}$ can approximately be derived from the matrix of moment arms $R(\boldsymbol{\theta})$ from (2.55). The detailed calculations of $J^{\lambda\theta}$ is given at the end of this section in Sec. 3.2.1.1 and for now it is assumed that $J^{\lambda\theta}$ is available as defined in (3.9).

The transformation of the discrete error signal $\boldsymbol{\theta}^{\text{err}}$ in (3.7) with the Jacobian $J^{\lambda\theta}$ from (3.9), which relates infinitesimal changes, is achieved by integrating $\partial\boldsymbol{l}^{\text{CE}} = J^{\lambda\theta}(\boldsymbol{\theta}) \cdot \partial\boldsymbol{\theta}$ with the limits of the respective error signals (i.e. $\boldsymbol{l}^{\text{CE}}(t)$ and $\boldsymbol{\lambda}^\theta(t)$, and $\boldsymbol{\theta}(t)$ and $\boldsymbol{\theta}^{\text{des}}(t)$):

$$\begin{aligned} \int_{\boldsymbol{l}^{\text{CE}}}^{\boldsymbol{\lambda}^\theta} \mathrm{d}\boldsymbol{l}^{\text{CE}} &= \int_{\boldsymbol{\theta}}^{\boldsymbol{\theta}^{\text{des}}} J^{\lambda\theta}(\boldsymbol{\theta}) \cdot \mathrm{d}\boldsymbol{\theta}, \qquad \text{with } \lambda^\theta := \boldsymbol{l}^{\text{CE}}\big|_{\boldsymbol{\theta}^{\text{des}}} \\ \boldsymbol{l}^{\text{CE}} - \boldsymbol{\lambda}^\theta &= \bar{J}^{\lambda\theta}(\boldsymbol{\theta}) - \bar{J}^{\lambda\theta}(\boldsymbol{\theta}^{\text{des}}). \end{aligned} \tag{3.10}$$

Here the antiderivate $\bar{J}^{\lambda\theta}$ is an unknown, joint-angle dependent function $\bar{J}^{\lambda\theta}(\boldsymbol{\theta})$ that can be approximated by a first-order Taylor approximation at the set-point of the actual measured joint angle configuration $\boldsymbol{\theta}^* = \boldsymbol{\theta}(t)$:

$$\begin{aligned} \bar{J}^{\lambda\theta}(\boldsymbol{\theta})\big|_{\boldsymbol{\theta}^*} &= \bar{J}^{\lambda\theta}(\boldsymbol{\theta}^*) + J^{\lambda\theta}\big|_{\boldsymbol{\theta}^*} \cdot (\boldsymbol{\theta} - \boldsymbol{\theta}^*) + \mathcal{O}(\boldsymbol{\theta}^2), \text{ i.e.} \\ \bar{J}^{\lambda\theta}(\boldsymbol{\theta}^{\text{des}})\big|_{\boldsymbol{\theta}^* = \boldsymbol{\theta}_{\delta_\theta}} &\approx \bar{J}^{\lambda\theta}(\boldsymbol{\theta}) + J^{\lambda\theta} \cdot (\boldsymbol{\theta}^{\text{des}} - \boldsymbol{\theta}). \end{aligned} \tag{3.11}$$

The use of (3.11) in (3.10) cancels out all occurrences of the unknown antiderivative $\bar{J}^{\lambda\theta}$ and the following remains:

$$\boldsymbol{l}^{\text{err}}(t) = J^{\lambda\theta} \cdot \boldsymbol{\theta}^{\text{err}}(t) + \mathcal{O}(\boldsymbol{\theta}^{\text{err}2}), \tag{3.12}$$

with the respective control errors (3.3) and (3.7). The Jacobian matrix $J^{\lambda\theta}$ can thus be used to approximately transform the discrete control error $\boldsymbol{\theta}^{\text{err}}$ from the transformational layer to a control error of CE lengths $\boldsymbol{l}^{\text{err}}$ in the structural layer. Due to the linear approximation (3.11), the transformation is less accurate for large control errors. During a successful movement execution ($\boldsymbol{\theta} \to \boldsymbol{\theta}^{\text{des}}$), this approximation becomes more accurate and, thus, the

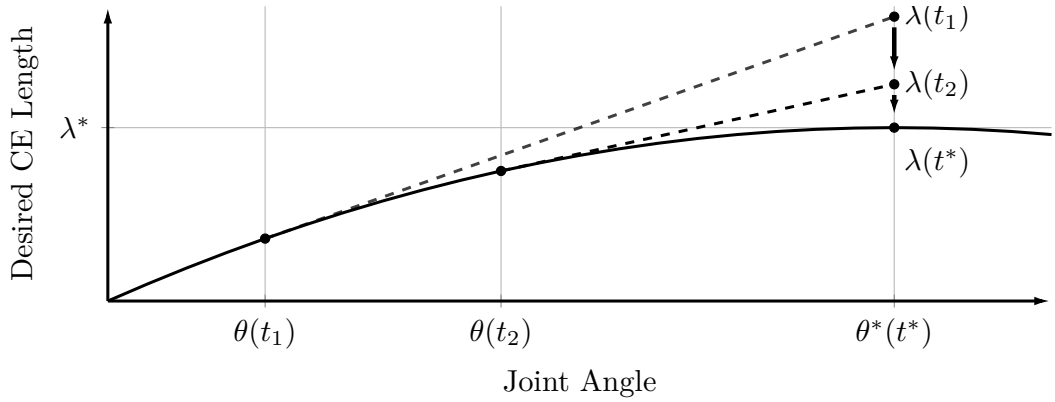


Figure 3.3: Simplified visualisation of the Taylor approximation of desired CE-lengths based on the current system state, e.g. $\boldsymbol{\theta}$. At time t_1, when still far away from the desired state $\boldsymbol{\theta}^*$, the linear approximation of the desired CE-lengths $\boldsymbol{\lambda}^*$ that correspond to $\boldsymbol{\theta}^*$ is potentially inaccurate. When approaching the desired state $\boldsymbol{\theta}^a st$ the Taylor estimation of $\boldsymbol{\lambda}$ becomes more precise, as displayed by the arrows consecutively pointing from $\boldsymbol{\lambda}(t_1)$ towards $\boldsymbol{\lambda}^*(t^*)$.

controller performances is more precise the closer the system approaches the desired state, as displayed in Fig. 3.3.

Finally, to complete the hierarchical θ_λ-controller, the vector of desired CE lengths $\boldsymbol{\lambda}^\theta(t)$ and their time rates $\dot{\boldsymbol{\lambda}}^\theta$ must be provided as input to the λ-controller (3.4). The vector $\boldsymbol{\lambda}^\theta(t)$ is obtained by equating $\boldsymbol{l}^{\mathrm{err}}(t)$ in (3.12) with (3.3) and solve the remaining equation for $\boldsymbol{\lambda}(t)$. As these desired CE lengths are assigned within the transformational layer, and based on angle information, they are written as $\boldsymbol{\lambda}^\theta(t)$. Additionally, to compensate approximation errors, and to ensure that $\boldsymbol{\theta}^{\mathrm{err}}(t)$ approaches zero along with $\boldsymbol{\lambda}^\theta(t) \to \boldsymbol{l}^{\mathrm{CE}}(t)$, the output $\boldsymbol{\theta}^{\mathrm{PID}}_{\theta_\lambda}(t)$ of the θ_λ-PID-controller (3.8), rather than $\boldsymbol{\theta}^{\mathrm{err}}(t)$, is used as input for the Jacobian transformation (3.12). With this, and by neglecting the higher order terms $\mathcal{O}(\boldsymbol{\theta}^{\mathrm{err}\,2})$, $\boldsymbol{l}^{\mathrm{err}}(t)$ in (3.12) now writes

$$\boldsymbol{l}^{\mathrm{err}} = \boldsymbol{l}^{\mathrm{CE}}(t) - \boldsymbol{\lambda}^\theta(t) \approx J^{\lambda\theta} \cdot \boldsymbol{\theta}^{\mathrm{PID}}_{\theta_\lambda}(t),$$

which eventually yields (compare to 3.3)

$$\boldsymbol{\lambda}^\theta(t) := \boldsymbol{l}^{\mathrm{CE}}(t) - J^{\lambda\theta} \cdot \boldsymbol{\theta}^{\mathrm{PID}}_{\theta_\lambda}(t), \tag{3.13}$$

that is, a reliable estimate of how the respective CE lengths have to be adapted to fulfil the task, i.e. reach $\boldsymbol{\theta}^{\mathrm{des}}$. In this, an estimation of the desired contraction velocity $\dot{\boldsymbol{\lambda}}^\theta$ is already included, since the definition of the angle-length Jacobian matrix (3.9) can also be read as (Sherman et al., 2013)

$$J^{\lambda\theta} = \frac{\partial \boldsymbol{l}^{\mathrm{CE}}}{\partial \boldsymbol{\theta}} = \frac{\mathrm{d}\boldsymbol{l}^{\mathrm{CE}}}{\mathrm{d}t} \Big/ \frac{\mathrm{d}\boldsymbol{\theta}}{\mathrm{d}t} = \frac{\dot{\boldsymbol{l}}^{\mathrm{CE}}}{\dot{\boldsymbol{\theta}}}, \tag{3.14}$$

which can be used for transforming desired joint angle velocities $\dot{\boldsymbol{\theta}}^{\mathrm{des}}$.

The (linear summation inherent to the) θ_λ-PID-controller (3.8) delivers an input for the layer transformation (3.13), that includes a velocity component to be transformed along in a natural way. Therefore, the transformational layer is not required to generate a separate signal $\dot{\boldsymbol{\lambda}}^\theta$ (i.e. $\dot{\boldsymbol{\lambda}}^\theta = 0$) for feeding the structural layer.

By making use of the λ-control law (3.4) with the hierarchical input signal $\boldsymbol{\lambda}^\theta$ according to (3.13), the stimulation signal contribution by the θ_λ-controller to (3.6) is

$$\boldsymbol{u}^{\theta\lambda}(t) := P_\lambda \cdot \left(\boldsymbol{l}^{\mathrm{CE}}(t) - \boldsymbol{\lambda}^\theta(t)\right) + D_\lambda \dot{\boldsymbol{l}}^{\mathrm{CE}}(t),$$

that is, the closed-loop, low-level feedback control law in the structural layer, to which the input signals $\boldsymbol{\lambda}^{\theta}(t)$ originate from the mid-level control in the transformational layer. Due to the hierarchical cascade $\boldsymbol{u}^{\theta\lambda}(\boldsymbol{\lambda}^{\theta}(\boldsymbol{\theta}^{\mathrm{des}}))$, with distributed PID- and PD-controllers, this stimulation contribution is task-fulfilling, even under gravity, notably because of the integrative part[1] of the θ_{λ}-controller (3.8).

3.2.1.1 Details on the calculations of the angle-length Jacobian $J^{\lambda\theta}$

The angle-length Jacobian matrix $J^{\lambda\theta} = \frac{\partial \boldsymbol{l}^{\mathrm{CE}}}{\partial \boldsymbol{\theta}} \in \mathbb{R}^{n_{\mathrm{MTU}} \times n_{\theta}}$ relates the change of the MTU's CE length $\boldsymbol{l}^{\mathrm{CE}} \in \mathbb{R}^{n_{\mathrm{MTU}}}$ to a change of the joint angles $\boldsymbol{\theta} \in \mathbb{R}^{n_{\theta}}$. It is used to transform a control signal from the transformational layer to the structural layer. The definition of $J^{\lambda\theta}$ in (3.9) is similar to the definition of the matrix of moment arms (2.55), which is also the basis of the calculation of $J^{\lambda\theta}$ when combined with the MTU length equation (2.40):

$$\begin{aligned} R(\boldsymbol{\theta}) &= \frac{\partial \boldsymbol{l}^{\mathrm{MTU}}}{\partial \boldsymbol{\theta}} = \frac{\partial \left(\boldsymbol{l}^{\mathrm{CE}} + \boldsymbol{l}^{\mathrm{SEE}}\right)}{\partial \boldsymbol{\theta}} = \frac{\partial \boldsymbol{l}^{\mathrm{CE}}}{\partial \boldsymbol{\theta}} + \frac{\partial \boldsymbol{l}^{\mathrm{SEE}}}{\partial \boldsymbol{\theta}} \\ &= \frac{\partial \boldsymbol{l}^{\mathrm{CE}}}{\partial \boldsymbol{\theta}} + \frac{\partial \boldsymbol{l}^{\mathrm{SEE}}}{\partial \boldsymbol{l}^{\mathrm{CE}}} \cdot \frac{\partial \boldsymbol{l}^{\mathrm{CE}}}{\partial \boldsymbol{\theta}} \\ &= \left(I_{n_{\mathrm{MTU}}} + \frac{\partial \boldsymbol{l}^{\mathrm{SEE}}}{\partial \boldsymbol{l}^{\mathrm{CE}}}\right) \cdot \frac{\partial \boldsymbol{l}^{\mathrm{CE}}}{\partial \boldsymbol{\theta}}, \end{aligned} \tag{3.15}$$

with $I_{n_{\mathrm{MTU}}}$ being the identity matrix of dimension n_{MTU}. By substituting $J^{\lambda\theta} := \frac{\partial \boldsymbol{l}^{\mathrm{CE}}}{\partial \boldsymbol{\theta}}$ and solving (3.15) for $J^{\lambda\theta}$, the angle-length Jacobian matrix is expressed as

$$J^{\lambda\theta} = \left(I_{n_{\mathrm{MTU}}} + \tfrac{\partial \boldsymbol{l}^{\mathrm{SEE}}}{\partial \boldsymbol{l}^{\mathrm{CE}}}\right)^{-1} \cdot R(\boldsymbol{\theta}). \tag{3.16}$$

The partial derivative $\frac{\partial \boldsymbol{l}^{\mathrm{SEE}}}{\partial \boldsymbol{l}^{\mathrm{CE}}} = \mathrm{diag}\left(\partial \mathrm{l}_{\mathrm{i}}^{\mathrm{SEE}} / \partial \mathrm{l}_{\mathrm{i}}^{\mathrm{CE}}\right)$ with $i = 1 ..., n_{\mathrm{MTU}}$, describes the change of the length of the SEE w.r.t. the change of the length of the CE. To calculate this term, the partial derivative of the MTU forces $\boldsymbol{f}^{\mathrm{MTU}}$ w.r.t. the CE lengths $\boldsymbol{l}^{\mathrm{CE}}$ is evaluated in a matrix equation, while considering the force equilibrium (2.41):

$$\frac{\partial \boldsymbol{f}^{\mathrm{MTU}}}{\partial \boldsymbol{l}^{\mathrm{CE}}} = \frac{\partial \boldsymbol{f}^{\mathrm{CE}}}{\partial \boldsymbol{l}^{\mathrm{CE}}} + \frac{\partial \boldsymbol{f}^{\mathrm{PEE}}}{\partial \boldsymbol{l}^{\mathrm{CE}}} = \frac{\partial \boldsymbol{f}^{\mathrm{SDE}}}{\partial \boldsymbol{l}^{\mathrm{CE}}} + \frac{\partial \boldsymbol{f}^{\mathrm{SEE}}}{\partial \boldsymbol{l}^{\mathrm{CE}}} \tag{3.17}$$

Here, the partial derivative of the SEE force vector $\boldsymbol{f}^{\mathrm{SEE}}$ w.r.t. the CE length vector $\boldsymbol{l}^{\mathrm{CE}}$ can be expressed as $\frac{\partial \boldsymbol{f}^{\mathrm{SEE}}}{\partial \boldsymbol{l}^{\mathrm{CE}}} = \frac{\partial \boldsymbol{f}^{\mathrm{SEE}}}{\partial \boldsymbol{l}^{\mathrm{SEE}}} \cdot \frac{\partial \boldsymbol{l}^{\mathrm{SEE}}}{\partial \boldsymbol{l}^{\mathrm{CE}}}$ and is assumed to be non-zero, i.e. the tendon is not slack. With this, equation (3.17) is rearranged to:

$$\frac{\partial \boldsymbol{l}^{\mathrm{SEE}}}{\partial \boldsymbol{l}^{\mathrm{CE}}} = \left(\frac{\partial \boldsymbol{f}^{\mathrm{CE}}}{\partial \boldsymbol{l}^{\mathrm{CE}}} + \frac{\partial \boldsymbol{f}^{\mathrm{PEE}}}{\partial \boldsymbol{l}^{\mathrm{CE}}} - \frac{\partial \boldsymbol{f}^{\mathrm{SDE}}}{\partial \boldsymbol{l}^{\mathrm{CE}}}\right) \cdot \left(\frac{\partial \boldsymbol{f}^{\mathrm{SEE}}}{\partial \boldsymbol{l}^{\mathrm{SEE}}}\right)^{-1}. \tag{3.18}$$

According to the force law equations of the MTU (2.47, 2.48, 2.49) from Sec. 2.2, the functions $\boldsymbol{f}^{\mathrm{CE}}$ and $\boldsymbol{f}^{\mathrm{PEE}}$ depend on $\boldsymbol{l}^{\mathrm{CE}}$, while $\boldsymbol{f}^{\mathrm{SEE}}$ depends on $\boldsymbol{l}^{\mathrm{SEE}}$. By assuming quasi-static conditions ($\dot{\boldsymbol{l}}^{\mathrm{MTU}} \approxeq \dot{l}^{\mathrm{CE}} \approxeq 0$), the SDE force vanishes ($\boldsymbol{f}^{\mathrm{SDE}} \approx 0$) and the CE force (2.47) of each individual MTU simplifies to $f^{\mathrm{CE}}(l^{\mathrm{CE}}) = f_{\max}^{\mathrm{CE}} \cdot a(\boldsymbol{l}^{\mathrm{CE}}) \cdot f_{\mathrm{isom}}(l^{\mathrm{CE}})$.
Subsequently, equation (3.18) can be written to the matrix equation

$$\begin{aligned} \frac{\partial \boldsymbol{l}^{\mathrm{SEE}}}{\partial \boldsymbol{l}^{\mathrm{CE}}} &= \left(\frac{\partial \boldsymbol{f}^{\mathrm{CE}}}{\partial \boldsymbol{l}^{\mathrm{CE}}} + \frac{\partial \boldsymbol{f}^{\mathrm{PEE}}}{\partial \boldsymbol{l}^{\mathrm{CE}}}\right) \cdot \left(\frac{\partial \boldsymbol{f}^{\mathrm{SEE}}}{\partial \boldsymbol{l}^{\mathrm{SEE}}}\right)^{-1} \\ &= \left(F_{\max} \cdot \left(\frac{\partial \boldsymbol{a}}{\partial \boldsymbol{l}^{\mathrm{CE}}} \cdot \boldsymbol{f}_{\mathrm{isom}}^{\mathrm{CE}} + \boldsymbol{a} \cdot \frac{\partial \boldsymbol{f}_{\mathrm{isom}}^{\mathrm{CE}}}{\partial \boldsymbol{l}^{\mathrm{CE}}}\right) + \frac{\partial \boldsymbol{f}^{\mathrm{PEE}}}{\partial \boldsymbol{l}^{\mathrm{CE}}}\right) \cdot \left(\frac{\partial \boldsymbol{f}^{\mathrm{SEE}}}{\partial \boldsymbol{l}^{\mathrm{SEE}}}\right)^{-1}. \end{aligned} \tag{3.19}$$

[1] The integrative part, on the other hand, may show so called 'wind-up' behavior, wherefore a simple anti wind-up technique may be included if required, e.g. see (Tarbouriech and Turner, 2009)

and thus, the partial derivatives of $\boldsymbol{a}(\boldsymbol{l}^{\mathrm{CE}})$, $\boldsymbol{f}^{\mathrm{isom}}(\boldsymbol{l}^{\mathrm{CE}})$, $\boldsymbol{f}^{\mathrm{PEE}}(\boldsymbol{l}^{\mathrm{CE}})$ and $\boldsymbol{f}^{\mathrm{SEE}}(\boldsymbol{l}^{\mathrm{SEE}})$ remain to be calculated.

Given the PEE force equation (2.48) for a single MTU with, its derivative w.r.t l^{CE} is given by

$$\frac{\partial f^{\mathrm{PEE}}}{\partial l^{\mathrm{CE}}} = \begin{cases} 0 & , l^{\mathrm{CE}} < l_0^{\mathrm{PEE}} \\ k^{\mathrm{PEE}} \cdot \nu^{\mathrm{PEE}} \cdot (l^{\mathrm{CE}} - l_0^{\mathrm{PEE}})^{(\nu^{\mathrm{PEE}}-1)} & , l^{\mathrm{CE}} \geq l_0^{\mathrm{PEE}} \end{cases}.$$

With the serial elastic force (2.49) for a single MTU, its derivative w.r.t. to l^{SEE} can be calculated and the result is

$$\frac{\partial f^{\mathrm{SEE}}}{\partial l^{\mathrm{SEE}}} = \begin{cases} 0 & , l^{\mathrm{SEE}} < l_0^{\mathrm{SEE}} \\ k_{\mathrm{nl}}^{\mathrm{SEE}} \cdot \nu^{\mathrm{SEE}} \cdot (l^{\mathrm{SEE}} - l_0^{\mathrm{SEE}})^{\nu^{\mathrm{SEE}}-1} & , l^{\mathrm{SEE}} < l_{\mathrm{nll}}^{\mathrm{SEE}} \\ k_1^{\mathrm{SEE}} & , l^{\mathrm{SEE}} \geq l_{\mathrm{nll}}^{\mathrm{SEE}} \end{cases},$$

The derivative of the isometric force (2.46) for a single MTU is given by

$$\frac{\partial f^{\mathrm{isom}}}{\partial l^{\mathrm{CE}}} = \frac{\nu_{\mathrm{limb}}^{\mathrm{CE}} \cdot f^{\mathrm{isom}} \cdot (l_{\mathrm{opt}}^{\mathrm{CE}} - l^{\mathrm{CE}}) \left| \frac{(l^{\mathrm{CE}}/l_{\mathrm{opt}}^{\mathrm{CE}})-1}{\Delta W_{\mathrm{limb}}^{\mathrm{CE}}} \right|^{\nu_{\mathrm{limb}}^{\mathrm{CE}}-2}}{{l^{\mathrm{CE}}}^2 \cdot {\Delta W_{\mathrm{limb}}^{\mathrm{CE}}}^2}.$$

The derivative of the non-linear activation $a(l^{\mathrm{CE}}, \gamma)$ w.r.t l^{CE} is

$$\begin{aligned} \frac{\partial a}{\partial l^{\mathrm{CE}}} &= \frac{(1-a_0) \cdot \frac{\partial \vartheta}{\partial l^{\mathrm{CE}}}}{(1+\vartheta)^2}, \quad \text{with} \\ \frac{\partial \vartheta}{\partial l^{\mathrm{CE}}} &= \nu \cdot \left(\frac{\gamma(t) \cdot \varpi_{\mathrm{opt}}}{l_{\mathrm{opt}}^{\mathrm{CE}}} \right)^{\nu} \cdot l^{\mathrm{CE}\,(\nu-1)} = \frac{\nu \cdot \vartheta}{l^{\mathrm{CE}}}. \end{aligned} \tag{3.20}$$

For the derivative of a w.r.t γ the same steps are taken, leading to[2]

$$\frac{\partial a}{\partial \gamma} = \frac{(1-a_0) \cdot \frac{\partial \vartheta}{\partial \gamma}}{(1+\vartheta)^2}, \quad \text{with} \quad \frac{\partial \vartheta}{\partial \gamma} = \frac{\nu \cdot \vartheta}{\gamma}. \tag{3.21}$$

With those equations all terms of equation (3.19) are sufficiently resolved and $\frac{\partial \boldsymbol{l}^{\mathrm{SEE}}}{\partial \boldsymbol{l}^{\mathrm{CE}}}$ can be calculated during runtime at each time-step of the simulation, using the respective parameters.

The above calculation (3.19) that assumes quasi-static conditions (i.e. $\dot{\boldsymbol{l}}^{\mathrm{MTU}} \approxeq \dot{\boldsymbol{l}}^{\mathrm{CE}} \approxeq 0$), is used in this dissertation for the calculation of the angle-length Jacobian (3.16). To evaluate the impact of this quasi-static assumption, a brief estimation without it follows: Calculating the partial derivative $\frac{\partial \boldsymbol{f}^{\mathrm{SDE}}}{\partial \boldsymbol{l}^{\mathrm{CE}}}$ in (3.18) from (2.50) with $R^{\mathrm{SDE}} \approx 0$ (Bayer et al., 2017), (3.18) can be rewritten in the scalar case for the k-th MTU as

$$\frac{\partial l^{\mathrm{SEE}}}{\partial l^{\mathrm{CE}}} = \left(1 - \left(\frac{D_{\mathrm{max}}^{\mathrm{SDE}} \cdot (\dot{l}^{\mathrm{MTU}} - \dot{l}^{\mathrm{CE}})}{f_{\mathrm{max}}^{\mathrm{CE}}} \right) \right) \cdot \left(\frac{\partial f^{\mathrm{CE}}}{\partial l^{\mathrm{CE}}} + \frac{\partial f^{\mathrm{PEE}}}{\partial l^{\mathrm{CE}}} \right) \cdot \left(\frac{\partial f^{\mathrm{SEE}}}{\partial l^{\mathrm{SEE}}} \right)^{-1}. \tag{3.22}$$

Additionally estimating the SEE contraction velocity to be as high as the maximum concentric contraction velocity of the CE for the MTU, i.e. $\dot{l}^{\mathrm{SEE}} = (\dot{l}^{\mathrm{MTU}} - \dot{l}^{\mathrm{CE}}) = v_{\mathrm{max}} = \frac{B_{\mathrm{rel}}}{A_{\mathrm{rel}}} \cdot l_{\mathrm{opt}}^{\mathrm{CE}}$, (3.22) contracts to (for substituting $D_{\mathrm{max}}^{\mathrm{SDE}}$, see Section 2.2.2.4)

$$\frac{\partial l^{\mathrm{SEE}}}{\partial l^{\mathrm{CE}}} = (1 - D^{\mathrm{SDE}}) \cdot \left(\frac{\partial f^{\mathrm{CE}}}{\partial l^{\mathrm{CE}}} + \frac{\partial f^{\mathrm{PEE}}}{\partial l^{\mathrm{CE}}} \right) \cdot \left(\frac{\partial f^{\mathrm{SEE}}}{\partial l^{\mathrm{SEE}}} \right)^{-1}. \tag{3.23}$$

[2] Note, that the derivative of a w.r.t. γ is not used for the calculation of the Jacobian $J^{\lambda\theta}$, but later for $J^{u\lambda}$ in Section 3.2.2.

As the relative damping coefficient is about $D^{\mathrm{SDE}} = 0.3$ for each MTU (Bayer et al., 2017), it is seen that (3.19) is a reasonable approximation, at least in all non-explosive movements, where the contraction velocity is below $v_{\max}$. Such movements, for example, include most everyday tasks, such as pointing and grasping, as well as standing and walking.

3.2.2 The direct θ-controller with co-contraction

By following the hybrid control approach (3.1, 3.6) (Bayer et al., 2017; Kistemaker et al., 2006), an 'open-loop' stimulation contribution, which is, in addition, physiological meaningful, can improve the control performance (Kistemaker et al., 2006). The 'open-loop' part of the present control architecture is presented in the following: it produces a stimulation signal that is based on the set of desired joint angles $\boldsymbol{\theta}^{\mathrm{des}}$ (postural plan), without making draft on the low-level control mechanisms in the structural layer.

The approach to the direct θ-controller with co-contraction is analogous to the hierarchical θ_λ controller from above. The desired joint angles $\boldsymbol{\theta}^{\mathrm{des}}(t)$ are compared to the actual joint angles $\boldsymbol{\theta}(t)$, and the resulting control errors $\boldsymbol{\theta}^{\mathrm{err}}_{\delta_\theta}$, with latencies $\boldsymbol{\delta_\theta}$ according to (3.7), serves as input to a PID control law in the transformational layer

$$\boldsymbol{\theta}^{\mathrm{PID}}_{\theta} := P_\theta \cdot \boldsymbol{\theta}^{\mathrm{err}}_{\delta_\theta} + D_\theta \cdot \dot{\boldsymbol{\theta}}^{\mathrm{err}}_{\delta_\theta} + I_\theta \cdot \int \boldsymbol{\theta}^{\mathrm{err}}_{\delta_\theta} \mathrm{d}t, \tag{3.24}$$

with the diagonal control matrices $P_\theta = \mathrm{diag}(p_{\theta,1}, \ldots, p_{\theta,n_\theta})$, $D_\theta = \mathrm{diag}(d_{\theta,1}, \ldots, d_{\theta,n_\theta})$, $I_\theta = \mathrm{diag}(\imath_{\theta,1}, \ldots, \imath_{\theta,n_\theta})$, containing the components $p_{\theta,i} > 0$, $d_{\theta,i} > 0$ and $\imath_{\theta,i} > 0$.

As an intermediate but only half-way step, the output of this θ-controller can be transformed through the angle-length Jacobian $J^{\lambda\theta}$ (3.9, 3.16), in a first instance, to the transformational layer. However, and in contrast to the θ_λ-controller, the low-level λ control law in the structural layer is eventually in fact bypassed by an additional Jacobian transformation $J^{u\lambda}$, obtaining the stimulation contribution $\boldsymbol{u}^\theta$ directly. By constructing this (length-stimulation) Jacobian $J^{u\lambda}$ (3.26), an angle-stimulation Jacobian $J^{u\theta}$ can be composed that contains $J^{\lambda\theta}$ and immediately transforms the changes of joint angles $\partial\boldsymbol{\theta}$ in the transformational layer to changes of MTU stimulations $\partial\boldsymbol{u}$ in the structural layer:

$$\begin{aligned} J^{u\theta} = J^{u\lambda} \cdot J^{\lambda\theta} &= \frac{\partial \boldsymbol{u}}{\partial \boldsymbol{l}^{\mathrm{CE}}} \cdot \frac{\partial \boldsymbol{l}^{\mathrm{CE}}}{\partial \boldsymbol{\theta}} = \frac{\partial \boldsymbol{u}}{\partial \boldsymbol{\theta}} \in \mathbb{R}^{n_{\mathrm{MTU}} \times n_\theta} \\ \partial \boldsymbol{u} &= J^{u\theta} \cdot \partial \boldsymbol{\theta}, \end{aligned} \tag{3.25}$$

with $J^{\lambda\theta}$ known from (3.16) and

$$J^{u\lambda} := \frac{\partial \boldsymbol{u}}{\partial \boldsymbol{l}^{\mathrm{CE}}} \in \mathbb{R}^{n_{\mathrm{MTU}} \times n_{\mathrm{MTU}}}. \tag{3.26}$$

For calculating this length-stimulation Jacobian $J^{u\lambda}$, structural knowledge of the muscle activation dynamics (2.38) and their individual (scalar) steady state ($\dot{\gamma} = 0$, $\gamma = u$), symbolised by a^{ss}, is utilised. This yields a steady-state activation $a^{\mathrm{ss}}(u(t), l^{\mathrm{CE}}(t))$, dependent of the current stimulation $u(t)$ and CE length $l^{\mathrm{CE}}(t)$ of the MTU. In steady-state, the activation a^{ss} does not change and its total differential vanishes.

$$0 = \mathrm{d}a^{\mathrm{ss}} = \frac{\partial a^{\mathrm{ss}}}{\partial u}\mathrm{d}u + \frac{\partial a^{\mathrm{ss}}}{\partial l^{\mathrm{CE}}}\mathrm{d}l^{\mathrm{CE}},$$

Using the partial derivatives (3.20-3.21) of a^{ss}, known from Section 3.2.1.1, leads to

$$\begin{aligned} 0 = \frac{\mathrm{d}a^{\mathrm{ss}}}{\mathrm{d}l^{\mathrm{CE}}} &= \frac{\partial a^{\mathrm{ss}}}{\partial u} \cdot \frac{\mathrm{d}u}{\mathrm{d}l^{\mathrm{CE}}} + \frac{\partial a^{\mathrm{ss}}}{\partial l^{\mathrm{CE}}} \\ \Rightarrow \frac{\mathrm{d}u}{\mathrm{d}l^{\mathrm{CE}}} &= -\frac{\partial a^{\mathrm{ss}}}{\partial l^{\mathrm{CE}}} \Big/ \frac{\partial a^{\mathrm{ss}}}{\partial u} = -\frac{u}{l^{\mathrm{CE}}} \left(= j^{u\lambda}\right). \end{aligned} \tag{3.27}$$

The length-stimulation Jacobian matrix $J^{u\lambda}$ is finally obtained as a diagonal matrix $J^{u\lambda} = \mathrm{diag}(j_i^{u\lambda})$ for $i = 1 \dots n_{\mathrm{MTU}}$.

For using the length-stimulation Jacobian (3.27) in the θ-controller, a first-order Taylor approximation is applied at the set-point of the current CE lengths $\boldsymbol{l}^{\mathrm{CE}}$ and reference stimulation levels $\boldsymbol{u}_{\mathrm{ref}}^{\mathrm{coc}}$ (see Section 3.2.3) with the integration limits of $\boldsymbol{\lambda}^{\theta} = \boldsymbol{l}^{\mathrm{CE}}|_{\boldsymbol{\theta}^{\mathrm{des}}}$ and $\boldsymbol{u}_{\mathrm{task}}^{\mathrm{opn}}$. This Taylor approximation is carried out with the same reasoning as before in the transformational layer in Section 3.2.1. It implies the definition of the stimulation error

$$\boldsymbol{u}^{\mathrm{err}}(t) := \boldsymbol{u}_{\mathrm{ref}}^{\mathrm{coc}}(t) - \boldsymbol{u}_{\mathrm{task}}^{\mathrm{opn}}(t) \in \mathbb{R}^{n_{\mathrm{MTU}}}, \tag{3.28}$$

where $\boldsymbol{u}_{\mathrm{ref}}^{\mathrm{coc}}$ is an arbitrary but non-zero reference stimulation and $\boldsymbol{u}_{\mathrm{task}}^{\mathrm{opn}}$ is a task-fulfilling stimulation contribution that eventually corresponds to the postural plan $\boldsymbol{\theta}^{\mathrm{des}}$. To obtain an estimate of this stimulation error, firstly the discrete integration with the limits of $\boldsymbol{u}_{\mathrm{ref}}^{\mathrm{coc}}$ and $\boldsymbol{u}_{\mathrm{task}}^{\mathrm{opn}}$ on the left hand side, and $\boldsymbol{l}_{\delta_\theta}^{\mathrm{CE}}$ and $\boldsymbol{\lambda}^{\theta}$ on the right hand side is performed on the definition of the length-stimulation Jacobian (3.26):

$$\begin{aligned} \partial \boldsymbol{u} &= J^{u\lambda} \cdot \partial \boldsymbol{l}^{\mathrm{CE}} \quad \Leftrightarrow \quad \int_{\boldsymbol{u}_{\mathrm{ref}}^{\mathrm{coc}}}^{\boldsymbol{u}_{\mathrm{task}}^{\mathrm{opn}}} \mathrm{d}\boldsymbol{u} = \int_{\boldsymbol{l}_{\delta_\theta}^{\mathrm{CE}}}^{\boldsymbol{\lambda}^{\theta}} J^{u\lambda}(\boldsymbol{l}^{\mathrm{CE}}) \mathrm{d}\boldsymbol{l}^{\mathrm{CE}} \\ \boldsymbol{u}_{\mathrm{ref}}^{\mathrm{coc}} - \boldsymbol{u}_{\mathrm{task}}^{\mathrm{opn}} &= \bar{J}^{u\lambda}(\boldsymbol{l}_{\delta_\theta}^{\mathrm{CE}}) - \bar{J}^{u\lambda}(\boldsymbol{\lambda}^{\theta}). \end{aligned} \tag{3.29}$$

With an analogue first-order Taylor approximation as in (3.11), the unknown antiderivatives $\bar{J}^{u\lambda}(\boldsymbol{l}_{\delta_\theta}^{\mathrm{CE}})$ and $\bar{J}^{u\lambda}(\boldsymbol{\lambda}^{\theta})$ in (3.29) can be linearly estimated to obtain:

$$\boldsymbol{u}^{\mathrm{err}} = J^{u\lambda} \cdot \boldsymbol{l}^{\mathrm{err}} + \mathcal{O}(\boldsymbol{l}^{\mathrm{CE}^2}) \tag{3.30}$$

With this, an error of CE lengths can be transformed to an error of the reference co-contraction (base stimulation level) $\boldsymbol{u}_{\mathrm{ref}}^{\mathrm{coc}}$ and the desired task fulfilling stimulation $\boldsymbol{u}_{\mathrm{task}}^{\mathrm{opn}}$. By rearranging, the task fulfilling stimulation $\boldsymbol{u}_{\mathrm{task}}^{\mathrm{opn}}$ can be obtained, based on the transformed CE length error $\boldsymbol{l}^{\mathrm{err}}$ and the chosen reference stimulation level $\boldsymbol{u}_{\mathrm{ref}}^{\mathrm{coc}}$. With this, and by neglecting all higher-order terms $\mathcal{O}(\boldsymbol{l}^{\mathrm{err}^2})$, (3.28, 3.30) can be solved for the stimulation contribution $\boldsymbol{u}_{\mathrm{task}}^{\mathrm{opn}}(t)$ (3.5) sought after.

To complete the construction of the θ-controller, similar as for the θ_λ-controller (see (3.13)), $\boldsymbol{l}^{\mathrm{err}}(t)$ is now replaced in (3.30) by making use of (3.12) and the angle-length Jacobian $J^{\lambda\theta}$ (3.9) as well as the θ-PID controller output $\boldsymbol{\theta}^{\mathrm{PID}}(t)$ (3.24) as an estimate of $\boldsymbol{\theta}^{\mathrm{err}}(t)$. This gives a robust, stabilising estimate

$$\boldsymbol{u}^{\theta}(t) := -\, J^{u\lambda}\big|_{\boldsymbol{u}=\boldsymbol{u}_{\mathrm{ref}}^{\mathrm{coc}}} \cdot J^{\lambda\theta} \cdot \boldsymbol{\theta}_{\theta}^{\mathrm{PID}}(t)$$

of $\boldsymbol{u}^{\mathrm{err}}(t)$ (3.28, 3.30) , which substantiates the proposed ansatz for the θ-controller by approximating $\boldsymbol{u}_{\mathrm{task}}^{\mathrm{opn}}$ according to (3.5) by

$$\begin{aligned} \boldsymbol{u}_{\mathrm{task}}^{\mathrm{opn}}(t) &:\approx \boldsymbol{u}_{\mathrm{ref}}^{\mathrm{coc}}(t) - J^{u\lambda}\big|_{\boldsymbol{u}=\boldsymbol{u}_{\mathrm{ref}}^{\mathrm{coc}}} \cdot J^{\lambda\theta} \cdot \boldsymbol{\theta}_{\theta}^{\mathrm{PID}}(t) \\ &= \boldsymbol{u}_{\mathrm{ref}}^{\mathrm{coc}}(t) - \underbrace{J^{u\theta}\big|_{\boldsymbol{u}=\boldsymbol{u}_{\mathrm{ref}}^{\mathrm{coc}}} \cdot \boldsymbol{\theta}_{\theta}^{\mathrm{PID}}(t)}_{:=-\boldsymbol{u}^{\theta}(t)}. \end{aligned} \tag{3.31}$$

Here, $\boldsymbol{u}^{\theta}$ is the output of the θ-PID-controller that is designed to adapt the reference stimulation $\boldsymbol{u}_{\mathrm{ref}}^{\mathrm{coc}}$ to the task-fulfilling 'open-loop' (from the perspective of the structural layer) stimulation contribution $\boldsymbol{u}_{\mathrm{task}}^{\mathrm{opn}}$. Thus, the base contraction level $\boldsymbol{u}_{\mathrm{ref}}^{\mathrm{coc}}$ is not required to yield the desired equilibrium and can be chosen more freely.

3.2.3 Choosing the base reference stimulation level Stimcocref

The assignment of the base reference stimulations $\boldsymbol{u}_{\text{ref}}^{\text{coc}}$ is technically simple. Firstly, a basis for the $k = 1 \dots n_{\text{MTU}}$ MTU stimulations is constructed by the linear independent vectors $\mathbf{e}_k^u = [e_{k,1}, \dots, e_{k,n_{\text{MTU}}}] \in \mathbb{R}_1^{n_{\text{MTU}}}$, with $e_{k,i}^u = 0$ for $i \neq k$ and $e_{k,i}^u = 1$ for $i = k$. The reference stimulation $\boldsymbol{u}_{\text{ref}}^{\text{coc}}$ is then obtained by choosing a base contraction parameter $\zeta_k^u \in \mathbb{R}_{01}$ for each MTU to scale the basis of the stimulation vectors:

$$\boldsymbol{u}_{\text{ref}}^{\text{coc}} = \sum_{k=1}^{n_{\text{MTU}}} \zeta_k^u \cdot \mathbf{e}_k^u \in \mathbb{R}_1^{n_{\text{MTU}}}.$$

With this, an arbitrary reference stimulation level can be assigned to each MTU individually. As already outlined in the previous section, the presented control architecture does not require $\boldsymbol{u}_{\text{ref}}^{\text{coc}}$ to be in accordance with the specific movement task that is specified by $\boldsymbol{\theta}^{\text{des}}$. Moreover, in most of the simulation tasks examined in Pt. III of this thesis, assigning one and the same co-contraction parameter value $\zeta_k^u = \zeta^{u*}$ to all MTUs was sufficient.

3.2.4 A co-contraction on joint layer

Above, the two Jacobian matrices $J^{\lambda\theta}$ (see (3.9, 3.16)) and $J^{u\lambda}$ (see (3.26, 3.27)) that resolve the muscle redundancy have been presented. In composition $J^{u\theta} = J^{u\lambda} J^{\lambda\theta}$ (see (3.25)), they even allow to transform joint angle changes $\partial\boldsymbol{\theta}$ immediately to MTU stimulation changes $\partial\boldsymbol{u}$. This finding can be further exploited for allowing the system to satisfy criteria beyond fulfilling the primary task. A criterion added to fulfilling a specific movement task may be the level of joint stiffness or speed of execution, while co-contraction has a significant impact on both joint stiffness (Bayer et al., 2017; De Serres and Milner, 1991; Gribble et al., 2003; Kistemaker et al., 2007; Milner, 2002; Milner et al., 1995) and movement speed (Bayer et al., 2017; Gribble et al., 1998; Kistemaker et al., 2006; McIntyre and Bizzi, 1993). It is derived in the following how this architectural integration potential, which is implicit to the muscle redundancy imprinted in $J^{\lambda\theta}$ (and thus $J^{u\theta}$), can be used to set a functionally desired joint-related co-contracting contribution to any of MTU stimulations.

In addition to setting, as described in Sec. 3.2.3 (c), a base contraction level $\zeta_k^u \in \mathbb{R}^{n_{\text{MTU}}}$ for each MTU individually, a second co-contraction parameter $\zeta_j^\theta \in \mathbb{R}$ for each of the $j = 1 \dots n_\theta$ muscle-actuated joint DoFs $\boldsymbol{\theta}$ of the skeletal system can be deployed. The redundant nature of the embodiment of the n_{MTU} muscles opens a manifold of dimension $(n_{\text{MTU}} - n_\theta)$ to assign MTU stimulation contributions $\boldsymbol{u}_\theta^{\text{coc}}$ derived from these co-contraction parameters ζ_j^θ. To exactly not interfere with the movement task specified by $\boldsymbol{\theta}^{\text{des}}$, the joint co-contraction $\boldsymbol{u}_\theta^{\text{coc}}$ is chosen based on the null-space of the (Moore-Penrose pseudo) inverse $J^{u\theta\dagger}$ of $J^{u\theta}$. The null-space—or kernel, respectively—of $J^{u\theta\dagger}$ is the set of infinitesimal MTU stimulation changes $\partial\boldsymbol{u}$ that are mapped via $J^{u\theta\dagger}$ to the null vector of angular changes, i.e. $\partial\boldsymbol{\theta} = 0$. This means that the discrete addition of any sufficiently small vector $\boldsymbol{u}_\theta^{\text{coc}}$ from the null space of $J^{u\theta\dagger}$ to the MTU stimulation $\boldsymbol{u}$ (i.e. (3.6)) results in the joint angles $\boldsymbol{\theta}(t)$ not significantly changing; therefore, adding $\boldsymbol{u}_\theta^{\text{coc}}$ does not interfere with the movement task $\boldsymbol{\theta}^{\text{des}}$.

Mathematically the null-space is obtained by solving

$$0 \stackrel{!}{=} \partial\boldsymbol{\theta} = J^{u\theta\dagger} \partial\boldsymbol{u}, \tag{3.32}$$

e.g. via Gaussian elimination, where $J^{u\theta\dagger}$ is the Moore-Penrose pseudo inverse of $J^{u\theta}$ defined by

$$J^{u\theta\dagger} := (J^{u\theta T} \cdot J^{u\theta})^{-1} \cdot J^{u\theta T} \in \mathbb{R}^{n_\theta \times n_{\text{MTU}}}. \tag{3.33}$$

Note here, that the pseudo inverse (3.33) can only be calculated for $J^{u\theta}$ having full column rank ($\text{rk}(J^{u\theta}) = n_\theta$), which is not the case, e.g. if the joint DoFs $\tilde{\boldsymbol{\theta}}$ that are not actuated by

muscles are considered in the control architecture. In all feasible cases, the solution of (3.32) yields a basis of the null-space (kernel) of dimension $\dim(\ker(J^{u\theta\dagger})) = (n_{\mathrm{MTU}} - n_\theta)$, which equals to $\dim(\ker(J^{u\theta\dagger})) = n_\theta$ in the model that is used in this study. Thus, the basis of the null-space has exactly the same size as the set of MTU actuated joint angles $\boldsymbol{\theta}$, allowing n_θ additional DoFs to satisfy further movement criteria along with fulfilling the primary movement task. By this, the transformational layer of the control architecture proposed here facilitates to apply joint-based co-contraction: The linearly independent basis vectors $\mathbf{e}_j^\theta$ ($\|\mathbf{e}_j^\theta\| = 1$, $j = 1 \dots n_\theta$) that build the basis of the null-space obtained from (3.32) are simply scaled (similar to the base reference stimulation from Sec. 3.2.3 (c)) by the arbitrary co-contraction parameters $\zeta_j^\theta \in \mathbb{R}$ to obtain a joint co-contraction contribution

$$\boldsymbol{u}_\theta^{\mathrm{coc}} = \sum_{j=1}^{n_\theta} \zeta_j^\theta \cdot \mathbf{e}_j^\theta \in \mathbb{R}_{01}^{n_{\mathrm{MTU}}}$$

that fulfils (3.32). In this stimulation contribution, the implicit resolution of the muscle redundancy in $J^{u\theta}$ by anatomical knowledge (moment arms, contraction and activation dynamics) is exploited to generate synergistic MTU stimulations that do not interfere with fulfilling the movement task $\boldsymbol{\theta}^{\mathrm{des}}$, while yet allowing to vary simultaneously, but in a coordinated way, the activity level of all muscles that act on the same joint.

3.3 The conceptional layer

With the conceptional layer, the control space is shifted further above in the hierarchy. In this abstract space, the movement task can be efficiently formulated and cognitive load and planning resources of higher CNS parts are reduced. This reduction is primarily based on the potential to neglect morphological properties of the body in the planning process, such as muscle dynamics and parts of the body's geometry. Sufficient knowledge of the morphology that is required for control is already implemented in the lower and mid levels of the control hierarchy, namely in the structural and transformational layer. Exactly this representation of morphological intelligence can now be exploited in the planning process by the conceptional layer. That is, a desired movement can be described in a coordinate space in which it can be easily formulated and a plan exactly in these coordinates can be drafted without having to consider, e.g. the muscles. This concept of a decoupled planning and actuation space is also hypothesised by Wolpert (1997) and implicitly assumed for studies on joint-torque-based planning of human upright standing by, e.g. Rozendaal and van Soest (2005); Alexandrov et al. (2001); Edwards (2007): Solving the mechanical system by calculating torques that are required to maintain upright standing must be followed in the biological system by muscle stimulations that eventually lead to said torques. For other movements a different choice of the conceptional planning space may be better suited. A grasping movement, for example, may be described the most intuitive in the coordinate space of the position of the hand in 3D space.

To have a broad coverage of different planning spaces, in this section, the conceptional layer is mathematically designed in terms of joint angles $\boldsymbol{\theta}$, of joint torques $\boldsymbol{\tau}^{\mathrm{MTU}}$ and of limb positions $\boldsymbol{\chi}$. For each, a high-level PID feedback controller is implemented, to ensure that the movement is executed under desired conditions, and a Jacobian-based layer transformation is used, to provide the transformational layer's standardised input of a postural plan $\boldsymbol{\theta}^{\mathrm{des}}$. For this transformation further information of the morphology is exploited, e.g. geometry of the skeletal system for position control and muscle stiffness for joint control. The generation of the respective MTU stimulations then straightforwardly follows from the remainder of the hierarchically cascaded control architecture as described in the previous sections. That is, the postural plan is controlled in the transformational layer and the inputs for the structural layer of desired CE lengths $\boldsymbol{\lambda}^\theta$ and direct stimulations $\boldsymbol{u}_{\mathrm{task}}^{\mathrm{opn}}$ are generated

with the help of Jacobian matrices. In the structural layer, low level CE length controllers (hybrid controller) provide the MTU stimulations $\boldsymbol{u}^{\lambda}$ (3.1) that drive the skeletal system to follow the postural plan. In Fig. 3.4 the complete cascaded hierarchy of the control architecture is displayed. The mathematical design of a conceptional layer is technically equivalent to the θ_{λ} controller (Sec. 3.2.1) in the transformational layer. The introduction of the conceptional layers with the control space of joint angles in Sec. 3.3.1, of joint torques in Sec. 3.3.2 and of limb positions in Sec. 3.3.3 forms a first step towards a generalisation of arbitrary conceptional control spaces. This is shortly discussed in Sec. 3.3.4, where also an approach to the control of forces is presented.

3.3.1 Control of joint angles

For the control of joint angles, the conceptional layer takes the trivial form of simply providing the postural plan $\boldsymbol{\theta}_{\theta}^{\mathrm{des}}(t)$ directly to the PID controllers (3.8) and (3.24) in the transformational layer. Examples that include a conceptional movement planning in the space of joint angles are contained in most application examples presented in Pt. III.

3.3.2 Control of joint torques

Movement planning in terms of joint torques has the advantage of providing a way to manipulate the driving inputs to the body's mechanical dynamics (2.33). That is, the torques that are generated by the MTUs $\boldsymbol{\tau}^{\mathrm{MTU}}$ have a direct impact on the temporal development of the mechanical degrees of freedom $\boldsymbol{q} \in \mathbb{R}^{n_{\mathrm{DoF}}}$. By implementing a high-level planning process of the torques $\boldsymbol{\tau}^{\mathrm{MTU}}$, the dynamics of the mechanical equations of motion (2.33) can be manipulated. Thereby, the whole movement of the body can be controlled in the states of its mechanical dynamics.

The input to the torque controller in the conceptional layer is the vector of desired joint torques $\boldsymbol{\tau}^{\mathrm{des}} \in \mathbb{R}^{n_{\theta}}$. These desired torques are compared to the current torques of the MTUs $\boldsymbol{\tau}^{\mathrm{MTU}} \in \mathbb{R}^{n_{\theta}}$ and a PID-controller creates a stabilising and robust output. The output is transformed by the torque-angle Jacobian $J^{\theta\tau} \in \mathbb{R}^{n_{\theta} \times n_{\theta}}$ to form a postural plan $\boldsymbol{\theta}_{\tau}^{\mathrm{des}}$ and thereby to provide the input that is demanded by the transformational layer. The remainder of the control architecture eventually generates MTU stimulations $\boldsymbol{u}(t, \boldsymbol{\tau}^{\mathrm{des}}(t)) \in \mathbb{R}^{n_{\mathrm{MTU}}}$ that lead to the MTU force production $\boldsymbol{f}^{\mathrm{MTU}}(t, \boldsymbol{\tau}^{\mathrm{des}}(t))$. The joint torques that are generated by the MTUs are fed back to the conceptional layer and the high-level feedback loop is closed. Due to the cascaded control architecture, the generated MTU torques are set up to match the desired torques, concisely symbolised by $\boldsymbol{\tau}^{\mathrm{MTU}} \rightarrow \boldsymbol{\tau}^{\mathrm{des}}$.

Throughout this section and w.l.o.g. the control of the whole set of MTU torques $\boldsymbol{\tau}^{\mathrm{MTU}} \in \mathbb{R}^{n_{\theta}}$ is assumed. In the same way as described in Sec. 3.2, only a part of the body, i.e. a subset $\tilde{\boldsymbol{\tau}}^{\mathrm{MTU}} \in \mathbb{R}^{\tilde{n}_{\theta}}$ can be considered here. The afferent and efferent neural latencies $\boldsymbol{\delta}_{\mathrm{CL}}^{\mathrm{a/e}}$ of signals that communicate with the conceptional layer are taken care of in the same way as described in Sec. 3.1 and Sec. 3.2. A signals property of being affected by an afferent or efferent delay is illustrated in Fig 3.4 by an arrow that crosses the border to the right of the conceptional layer.

To deploy such a torque-based concept in the conceptional layer of the control architecture, at first the error signal of joint torques is defined:

$$\boldsymbol{\tau}^{\mathrm{err}}(t) = \boldsymbol{\tau}^{\mathrm{MTU}}(t) - \boldsymbol{\tau}^{\mathrm{des}}(t), \tag{3.34}$$

where $\boldsymbol{\tau}^{\mathrm{des}} = [\tau_1^{\mathrm{des}}, \ldots \tau_{n_{\theta}}^{\mathrm{des}}] \in \mathbb{R}^{n_{\theta}}$ is an input to the system. This error is controlled via the following PID control law on joint torques:

$$\boldsymbol{\tau}^{\mathrm{PID}} := P_{\tau} \cdot \boldsymbol{\tau}_{\boldsymbol{\delta}_{\boldsymbol{\tau}}}^{\mathrm{err}}(t) + D_{\tau} \cdot \dot{\boldsymbol{\tau}}_{\boldsymbol{\delta}_{\boldsymbol{\tau}}}^{\mathrm{err}}(t) + I_{\tau} \cdot \int \boldsymbol{\tau}_{\boldsymbol{\delta}_{\boldsymbol{\tau}}}^{\mathrm{err}}(t)\mathrm{d}t, \tag{3.35}$$

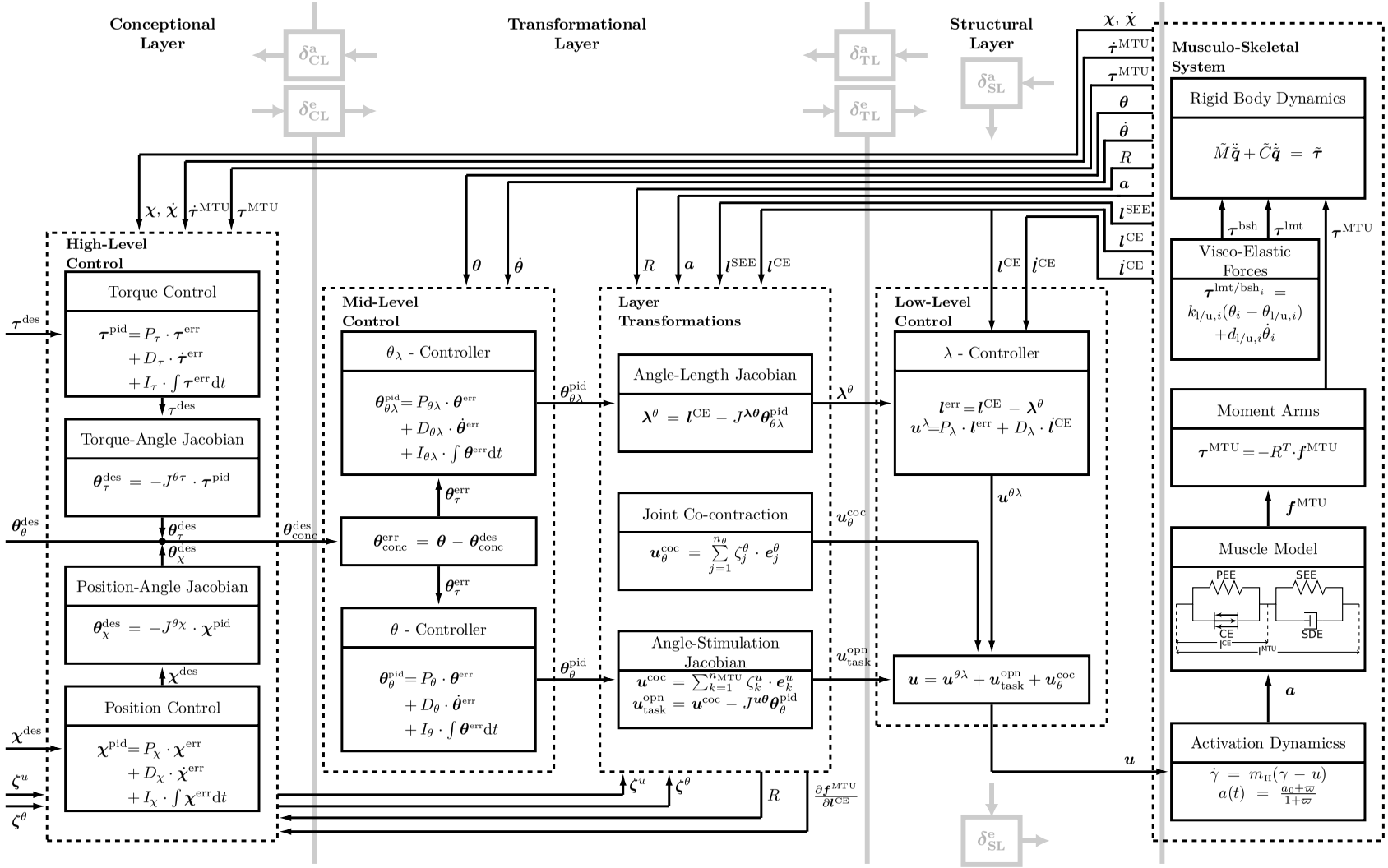


FIGURE 3.4: Block diagram of the complete hierarchical control architecture. Desired values for joint angles $\boldsymbol{\theta}^{\text{des}}$, joint torques $\boldsymbol{\tau}^{\text{des}}$ and/or limb positions $\boldsymbol{\chi}^{\text{des}}$, together with the parameters of muscle and joint co-contractions $\boldsymbol{\zeta}^{u}$ and $\boldsymbol{\zeta}^{\theta}$ are the total input of the hierarchical control architecture. Following the cascaded hierarchy of control, in each layer, PID controllers and Jacobian-based layer transformations eventually generate MTU stimulations $\boldsymbol{u}$. The resulting MTU forces/torques are therewith set-up to minimise the respective control errors, i.e. to fulfil the movement task.

where the control matrices $P_\tau = \mathrm{diag}(p_{\tau,1}, \ldots, p_{\tau,n_\theta})$, $D_\tau = \mathrm{diag}(d_{\tau,1}, \ldots, d_{\tau,n_\theta})$, and $I_\tau = \mathrm{diag}(\mathrm{I}_{\tau,1}, \ldots, \mathrm{I}_{\tau,n_\theta})$ are diagonal, with $p_{\tau,i} > 0$, $d_{\tau,i} > 0$ and $\mathrm{I}_{\tau,i} > 0$. The controlled signal $\boldsymbol{\tau}^{\mathrm{PID}}$ in the conceptional layer is subsequently transformed to the (transformational) layer of joint angles via the torque-angle Jacobian matrix

$$J^{\theta\tau} := \frac{\partial \boldsymbol{\theta}}{\partial \boldsymbol{\tau}} \in \mathbb{R}^{n_\theta \times n_\theta}, \tag{3.36}$$

which is the inverse of the joint stiffness matrix $K_\theta = \partial\boldsymbol{\tau}/\partial\boldsymbol{\theta}$ (Stanev and Moustakas, 2019). It can be calculated by inserting the moment arm equation of the joint torques (2.55) from Sec. 2.2.4:

$$K_\theta = \frac{\partial(-R^T \cdot \boldsymbol{f}^{\mathrm{MTU}})}{\partial \boldsymbol{\theta}}.$$

Applying the product rule and inserting the definition (3.9) of the angle-length Jacobian yields

$$K_\theta = -\frac{\partial R^T}{\partial \boldsymbol{\theta}} \bullet_2 \boldsymbol{f}^{\mathrm{MTU}} - R^T \frac{\partial \boldsymbol{f}^{\mathrm{MTU}}}{\partial \boldsymbol{l}^{\mathrm{CE}}} J^{\lambda\theta}. \tag{3.37}$$

where $\frac{\partial R^T}{\partial \boldsymbol{\theta}} \in \mathbb{R}^{n_\theta \times n_\theta \times n_{\mathrm{MTU}}}$ is a third-order tensor and $\bullet_2$ denotes a tensor product, leading to $(\frac{\partial R^T}{\partial \boldsymbol{\theta}} \bullet_2 \boldsymbol{f}^{\mathrm{MTU}}) \in \mathbb{R}^{n_\theta \times n_\theta}$ being a second-order tensor (Kolda and Bader, 2009; Stanev and Moustakas, 2019).
The calculation of $\partial \boldsymbol{f}^{\mathrm{MTU}}/\partial \boldsymbol{l}^{\mathrm{CE}}$ in (3.37) follows similar calculations as already given in Sec. 3.2.1.1. For the simulation study of this paper the first part of the joint stiffness matrix K_θ is neglected and the torque-angle Jacobian is used as

$$J^{\theta\tau} = \left(-R^T \frac{\partial \boldsymbol{f}^{\mathrm{MTU}}}{\partial \boldsymbol{l}^{\mathrm{CE}}} J^{\lambda\theta}\right)^{-1}. \tag{3.38}$$

In the closed form (3.38) of this Jacobian, the required anatomical knowledge is again apparent and consists of moment arms, MTU stiffnesses, and MTU-internal stiffness relations (see also (3.16)).

Similar as in (3.12) and (3.29), the torque error can be transformed to an error in joint angles using a first-order Taylor approximation, leading to the definition of a signal of desired joint angles, based on the controller output of joint torques:

$$\boldsymbol{\theta}_\tau^{\mathrm{des}}(t) := \boldsymbol{\theta}\,(t, \delta_\tau) - J^{\theta\tau} \cdot \boldsymbol{\tau}^{\mathrm{PID}}(t). \tag{3.39}$$

Here, with the same reasoning as in (3.14), the use of the PID-controller output—i.e. its D-part—in the transformation, brings along a transformation of the desired torque rate $\dot{\boldsymbol{\tau}}^{\mathrm{des}}$ to $\dot{\boldsymbol{\theta}}^{\mathrm{des}}$. Hence, it is not needed for the conceptional layer to feed a separate signal $\dot{\boldsymbol{\theta}}^{\mathrm{des}}$ into the PID-controllers (3.8) and (3.24) in the transformational layer ($\dot{\boldsymbol{\theta}}^{\mathrm{err}} = \dot{\boldsymbol{\theta}}$ is used). The postural plan (3.39) that results from the conceptional torque controller can be straightforwardly used to feed the joint-angle PID controllers (3.8) and (3.24) in the transformational layer. This closes the control loop of the three cascaded layers of high-level torque control, mid-level angle control and low level length control. See also Fig. 3.4 for a block diagram of the complete hierarchical control architecture.

3.3.3 Control of limb positions

Movements that involve the control of the position of a limb are everyday tasks such as reaching, pointing or grasping. During such a movement, the current position of the hand is estimated, via visual and proprioceptive sensory signals (Sober and Sabes, 2005), and muscle stimulations are synergised for its execution.

In this section, this general thought is mathematically integrated in the conceptional layer of the control architecture. The design process of this position controller follows the

same steps as for the θ_λ-controller (Sec. 3.2.1), the θ-controller (Sec. 3.2.2) and the τ-controller (Sec. 3.3.2). That is, a high-level, PID-controller of the position of a limb $\boldsymbol{\chi}^{\text{limb}}$ is implemented and its output is transformed by a Jacobian $J^{\theta\chi}$ to provide the postural plan $\boldsymbol{\theta}_\chi^{\text{des}}$ to the transformational layer. Following the remaining cascaded hierarchy, MTU stimulations $\boldsymbol{u}$ are eventually generated that are set up to drive the limb to the desired position $\boldsymbol{\chi}^{\text{limb}} \to \boldsymbol{\chi}^{\text{des}}$.

For the mathematical description of the limb position $\boldsymbol{\chi}^{\text{limb}}$ and of other position control related variables, many of the preliminaries of the mathematical system description of rigid body movements from Sec. 2.1 are used. The basic idea is to use the notion of 4×4 homogeneous rigid body transformations (2.11) to describe the current and desired positions in 3D space of a point at the end of a limb. As a reference coordinate system, the 'base-joint' of the limb is used, e.g. the shoulder of the respective hand. To control the movement of the limb, all joints are actuated accordingly that connect the endpoint of the limb to its base-joint. This can then be easily translated into the postural plan $\boldsymbol{\theta}_\chi^{\text{des}}$. For the notation on the afferent and efferent neural sensory latencies $\boldsymbol{\delta}_{\text{CL}}^{\text{a/e}}$ please refer to (3.2) in Sec. 3.1. The property of a signal to be affected by an afferent or efferent neural latency $\boldsymbol{\delta}_{\text{CL}}^{\text{a/e}}$ is illustrated in Fig. 3.4 and in the following the subscript is omitted.

By attaching a coordinate frame $\mathcal{L}$ at the end-point of a limb and a frame $\mathcal{D}$ at the desired place, their position and rotation in 3D space can be described in the inertial world frame $\mathcal{W}$ by the rigid body transformations

$$\boldsymbol{G}^{\mathcal{WL}} = \left[\begin{array}{ccc|c} & \boldsymbol{R}^{\mathcal{WL}} & & \boldsymbol{t}^{\mathcal{WL}} \\ \hline 0 & 0 & 0 & 0 \end{array}\right] \in SE(3),$$

which is already known from (2.11) in Sec. 2.1.3. According to (2.16), the respective coordinate vector is given by

$$\boldsymbol{\chi}^{\mathcal{WL}} = \left[\begin{array}{c} \boldsymbol{t}^{\mathcal{WL}} \\ \hline \boldsymbol{\phi}^{\mathcal{WL}} \end{array}\right] \in \mathbb{R}^6, \text{ with } \boldsymbol{\phi} := \frac{1}{2}\begin{bmatrix} r_{32} - r_{23} \\ r_{13} - r_{31} \\ r_{21} - r_{12} \end{bmatrix}.$$

Likewise a frame $\mathcal{D}$ is attached at the position that is desired to be reached by the limb. The relative configuration of the desired position with respect to the world is given by $\boldsymbol{G}^{\mathcal{WD}} \in SE(3)$ or $\boldsymbol{\chi}^{\mathcal{WD}} \in \mathbb{R}^6$. By recapitulating the rules for $\boldsymbol{G}$ in (2.15), the relative configuration of $\mathcal{L}$ in $\mathcal{D}$ is

$$\boldsymbol{G}^{\mathcal{DL}} = \boldsymbol{G}^{\mathcal{DW}}\boldsymbol{G}^{\mathcal{WL}} = \boldsymbol{G}^{\mathcal{WD}^{-1}}\boldsymbol{G}^{\mathcal{WL}}.$$

When the limb moves towards its desired position, the frame $\mathcal{L}$ of the limb approaches the frame $\mathcal{D}$, i.e. $\boldsymbol{G}^{\mathcal{WL}} \to \boldsymbol{G}^{\mathcal{WD}}$ and their relative distance decreases, i.e $\boldsymbol{G}^{\mathcal{DL}} \to \boldsymbol{I}_4$. Thus, the homogeneous representation $\boldsymbol{G}^{\mathcal{DL}}$ is a valid choice for the control error of the limbs position. The coordinate representation of this control error is given by the vector in $\mathbb{R}^6$ by (Ibuki et al., 2014)

$$\boldsymbol{\chi}^{\mathcal{DL}} = \left[\begin{array}{c} \boldsymbol{t}^{\mathcal{DL}} \\ \hline \boldsymbol{\phi}^{\mathcal{DL}} \end{array}\right] \in \mathbb{R}^6, \quad \text{with } \boldsymbol{t}^{\mathcal{DL}} = \begin{bmatrix} x^{\mathcal{DL}} \\ y^{\mathcal{DL}} \\ z^{\mathcal{DL}} \end{bmatrix} \in \mathbb{R}^3 \text{ and } \boldsymbol{\phi}^{\mathcal{DL}} = \begin{bmatrix} \phi_x^{\mathcal{DL}} \\ \phi_y^{\mathcal{DL}} \\ \phi_z^{\mathcal{DL}} \end{bmatrix} \in \mathbb{R}^3.$$

Before this signal is used for control, a more precise look into the system is taken: The mechanical description of the body (skeletal system) is usually a ramified kinematic chain,

i.e. it has a central part (e.g. the trunk) where the limbs are attached to. The limb itself is thereby the most important for manipulating its end point. To control the position of the hand, for example, it is the movement of the arm, and for the foot it is the leg. The joint that connects the respective limb to the central part of the ramified skeletal structure therefore can serve as a reference coordinate system for the control of the limb. Using this approach, the control error is transformed into the reference coordinate system (frame $\mathcal{R}$) of the joint that connects the limb to the central part. For this transformation it is sufficient to rotate the constituents $\boldsymbol{t}^{\mathcal{DL}}$ and $\boldsymbol{\phi}^{\mathcal{DL}}$ of $\boldsymbol{\chi}^{\mathcal{DL}}$ by $\boldsymbol{R}^{\mathcal{RD}} = \boldsymbol{R}^{\mathcal{RW}} \cdot \boldsymbol{R}^{\mathcal{WD}}$. Exactly this defines the positional control error $\boldsymbol{\chi}^{\text{err}}$ that is used in the following for the control of limb positions:

$$\boldsymbol{\chi}^{\text{err}} := \left[\begin{array}{c} \boldsymbol{t}^{\text{err}} \\ \hline \boldsymbol{\phi}^{\text{err}} \end{array}\right] \in \mathbb{R}^6 \quad \text{with } \boldsymbol{t}^{\text{err}} := \boldsymbol{R}^{\mathcal{RD}}\boldsymbol{t}^{\mathcal{DL}} \in \mathbb{R}^3 \text{ and } \boldsymbol{\phi}^{\text{err}} := \boldsymbol{R}^{\mathcal{RD}}\boldsymbol{\phi}^{\mathcal{DL}} \in \mathbb{R}^3. \quad (3.40)$$

Only a rotation into the reference coordinate frame $\mathcal{R}$ is used here, as, e.g. the control error $\boldsymbol{t}^{\text{err}}$ describes the connecting vector of the points at the origin of the frames $\mathcal{D}$ and $\mathcal{L}$. Such a connecting vector can also be derived by subtracting the respective points and the appended (1) in the fourth dimension cancels out to a (0). A subsequent transformation, e.g. by $\boldsymbol{G}^{\mathcal{RD}}$, corresponds to a pure rotation by $\boldsymbol{R}^{\mathcal{RD}}$ (see also (2.14) in Sec. 2.1.3).

Following the general approach of having a high-level PID feed-back controller in the conceptional layer, the control error (3.40) is the proportional (P) part that is used in combination with an integral (I) part and a derivative (D) part. The I part is simply obtained by integrating $\boldsymbol{\chi}^{\text{err}}$ over time:

$$\boldsymbol{\chi}^{\text{int}} := \int \boldsymbol{\chi}^{\text{err}} \mathrm{d}t.$$

The D part is obtained by recapitulating the preliminaries of rigid body velocities from section 2.1.5, i.e. (2.25) and (2.27). Given the homogeneous representations of the spatial velocities $\hat{\boldsymbol{V}}^{\mathcal{W}}_{\mathcal{WL}}$ and $\hat{\boldsymbol{V}}^{\mathcal{W}}_{\mathcal{WD}}$ of the limb and of the desired position with respect to the inertial world frame $\mathcal{W}$, their representation with respect to the frame $\mathcal{R}$ is given by

$$\dot{\boldsymbol{\chi}}^{\text{err}} = \boldsymbol{G}^{\mathcal{WR}^{-1}}(\hat{\boldsymbol{V}}^{\mathcal{W}}_{\mathcal{WL}} - \hat{\boldsymbol{V}}^{\mathcal{W}}_{\mathcal{WD}})\boldsymbol{G}^{\mathcal{WR}}. \quad (3.41)$$

With the application of a general PID-control law, the high-level control signal is given by

$$\boldsymbol{\chi}^{\text{pid}} := \boldsymbol{P}_\chi \cdot \boldsymbol{\chi}^{\text{err}} + \boldsymbol{I}_\chi \cdot \boldsymbol{\chi}^{\text{int}} + \boldsymbol{D}_\chi \cdot \dot{\boldsymbol{\chi}}^{\text{err}},$$

where $P_\chi = \text{diag}(p_{\chi,1}, \ldots, p_{\chi,6})$, $D_\chi = \text{diag}(d_{\chi,1}, \ldots, d_{\chi,6})$, and $I_\chi = \text{diag}(\text{I}_{\chi,1}, \ldots, \text{I}_{\chi,6})$ are diagonal control matrices. To feed this positional control signal to the transformational layer, a postural plan θ^{des} must be derived. This is achieved by following the general approach of the control architecture with the help of a Jacobian matrix. The position-joint Jacobian $\boldsymbol{J}^{\theta\chi}$ relates the change of a limbs position to a change of the limbs joint angles. It is defined by

$$\boldsymbol{J}^{\theta\chi} := \frac{\partial \boldsymbol{\theta}^{\text{lmb}}}{\partial \boldsymbol{\chi}} \in \mathbb{R}^{n_\theta^{\text{lmb}} \times 6}, \quad (3.42)$$

where θ^{lmb} is the set of DoFs that lie in between the the kinematic chain from the frame $\mathcal{R}$ to $\mathcal{L}$. The *inverse* of this Jacobian, i.e. $\boldsymbol{J}^{\chi\theta}$, can be simply calculated by geometrical analysis. Each entry $j^{\chi\theta}_{i,j}$ of $\boldsymbol{J}^{\chi\theta}$ describes the momentary change of a DoF θ_i with respect to a change of a positional (or rotational) direction (index j). That is, the columns $\boldsymbol{j}^{\chi\theta}_i$ that correlate to the i-th DoF are given by

$$\boldsymbol{j}^{\chi\theta}_i = \begin{bmatrix} \boldsymbol{r}^{\theta_i\mathcal{L}} \times \boldsymbol{\omega}_i \\ \boldsymbol{\omega}_i \end{bmatrix},$$

where $\boldsymbol{\omega}_i$ is the momentary screw axis of the i-th DoF according to (2.20) and $\boldsymbol{r}^{\theta_i \mathcal{L}}$ denotes the distance of the joint to the frame $\mathcal{L}$. To transform the positional error (3.41) to the transformational layer, a matrix inverse of $\boldsymbol{J}^{\chi\theta}$ is required. As $\boldsymbol{J}^{\chi\theta}$ is typically non-square, the following pseudo-inverse is used:

$$\boldsymbol{J}^{\theta\chi} = \boldsymbol{J}^{\chi\theta^\dagger} := (\boldsymbol{J}^{\chi\theta^T} \cdot \boldsymbol{J}^{\chi\theta})^{-1} \cdot \boldsymbol{J}^{\chi\theta^T} \in \mathbb{R}^{n_\theta \times 6}. \tag{3.43}$$

The postural plan $\boldsymbol{\theta}^{\text{des}}$ is then obtained by a first-order Taylor approximation, exactly as already described in (3.12) and (3.29), by

$$\boldsymbol{\theta}^{\text{des}}(t) := \boldsymbol{\theta}(t) - J^{\theta\chi} \cdot \boldsymbol{\chi}^{\text{pid}}(t).$$

When this signal is fed to the transformational layer, eventually MTU stimulations $\boldsymbol{u}^{\text{lmb}}(t)$ are generated that actuate the system, such that the DoFs are driven to let $\boldsymbol{\chi}^{\text{lm}}$ reach $\boldsymbol{\chi}^{\text{des}}$.

This method of control can potentially also be applied to manipulate the position of objects externally of the musculo-skeletal body, such as a stick in the hand, when the estimation of the Jacobian is sufficiently correct.

3.3.4 Control of forces and of other coordinates

In a similar manner as above, various different coordinate spaces can be thought of to be controlled. The only requirements are some measure of the coordinates to be controlled and a Jacobian matrix that eventually maps a PID-controlled variant of the coordinate space onto a postural plan $\boldsymbol{\theta}^{\text{des}}$. The latter can be a challenging task as a Jacobian may not be easy to find or can even turn out to be unfeasible. In some cases it may be sufficient, though, to map a coordinate space directly onto another conceptional layer.

The control of forces is such a case. By mapping a desired force of the end point of a limb onto equivalent joint angles, a vector of desired joint torques is obtained, that can be used as input for a conceptional torque controller as described in Sec. 3.3.2. Such a transformation can be achieved by the transpose of the angle-position Jacobian matrix $J^{\chi\theta} = \frac{\partial \boldsymbol{\chi}}{\partial \theta}$ (see also (3.42) and (3.43)). This becomes clear when considering the work done by the force $\boldsymbol{f}$ and by the torques $\boldsymbol{\tau}$, respectively, that can be rewritten as follows

$$\begin{aligned} \boldsymbol{\tau}^T \partial\boldsymbol{\theta} &= \boldsymbol{f}^T \partial\boldsymbol{\chi} \\ &= \boldsymbol{f}^T J^{\theta\chi} \partial\boldsymbol{\theta} \\ &= (J^{\theta\chi^T} \boldsymbol{f})^T \partial\boldsymbol{\theta} \\ \Rightarrow \boldsymbol{\tau} &= J^{\theta\chi^T} \boldsymbol{f} \end{aligned}$$

With this, a desired force $\boldsymbol{f}^{\text{des}}$ can directly be transformed into a vector of desired torques $\boldsymbol{\tau}^{\text{des}}$ that can be fed into the conceptional torque controller as described in Sec. 3.3.2.

Part III

In-silico Applications

The *digital human model* (DHM) '*allmin*' that is used for the application examples is based on anthropometric data from NASA (1978) and represents a 50 percentile human male with a mass of 81 kg and a total height of 1.78 m. Its muscular actuation is implemented on the principle of *elementary biological drives* (see Fig. 2.1 and (Schmitt et al., 2019)) using the MTUs from Sec. 2.2 (see also (Haeufle et al., 2014a)). For each muscle-actuated DoFs, there is exactly one *agonistic* and one *antagonistic* MTU modelled. The Body of the DHM consists of segments, which are modelled by means of rigid bodies, as described in Sec. 2.1. The body segments are connected by rotational joints with either one DoF (*hinge-joint*), or two DoFs (*universal-joint*). The pelvis body is the centre of the ramified rigid-body chain of a trunk with two legs, two arms and a head. The legs each consist of a thigh body, a shank and a foot. The *hip* (Hp) joint is allowed to rotate around the two DoFs of *flexion-extension* (FE) and *abduction-adduction* (AA) and the *knee* (Kn) and *ankle* (An) joints each around FE. The arms similarly consist each of an uparm body with the *shoulder* (Sh) movements of FE and AA, a forearm body with FE allowed in the *elbow* (Eb) joint and a hand with FE in the *wrist* (Wr). The trunk is allowed to rotate in *lateral flexion-extension* (lFE) and *ventral flexion-extension* (vFE)) in the *lumbar joint* (Lu) and in the *cervical joint* (Ce). The DHM is shown in Fig. 4.1 and its model parameters are given in Appendix C.

Taken together, the musculoskeletal model *allmin* consists of $n_{\mathrm{RGB}} = 15$ rigid bodies that are connected via 14 joints resulting in a total of $n_{\mathrm{DoF}} = 20$ DoFs. Each DoF, except for those of the Wrs, is controlled by two MTUs in an *agonistic-antagonistic setup* (AAS), to act as an elementary biological drive on the model (Haeufle et al., 2014a; Schmitt et al., 2019). Therewith, the musculoskeletal model is actuated by a total of $n_{\mathrm{MTU}} = 36$ MTUs and has $n_\theta = 18$ MTU-actuated DoFs. The complete *joint angle vector* $\boldsymbol{\theta}(t) \in \mathbb{R}^{n_\theta}$ consists of the elements $\theta_j(t) \in \mathbb{R}$ with the index j being one of the following joints:

$$
\begin{aligned}
j \in \{ \quad & \lceil \text{Lu, lFE} \rfloor, \lceil \text{Lu, vFE} \rfloor, \lceil \text{Ce, lFE} \rfloor, \lceil \text{Ce, vFE} \rfloor, \\
& \lceil \text{Sh,r, FE} \rfloor, \lceil \text{Sh,r, AA} \rfloor, \lceil \text{Eb,r, FE} \rfloor, \\
& \lceil \text{Sh,l, FE} \rfloor, \lceil \text{Sh,l, AA} \rfloor, \lceil \text{Eb,l, FE} \rfloor, \\
& \lceil \text{Hp,r, FE} \rfloor, \lceil \text{Hp,r, AA} \rfloor, \lceil \text{Kn,r, FE} \rfloor, \lceil \text{An,r, FE} \rfloor, \\
& \lceil \text{Hp,l, FE} \rfloor, \lceil \text{Hp,l, AA} \rfloor, \lceil \text{Kn,l, FE} \rfloor, \lceil \text{An,l, FE} \rfloor, \}^T
\end{aligned}
$$

A list of all bodies, their joints and actuating MTUs can be found in Tab. 4.1.

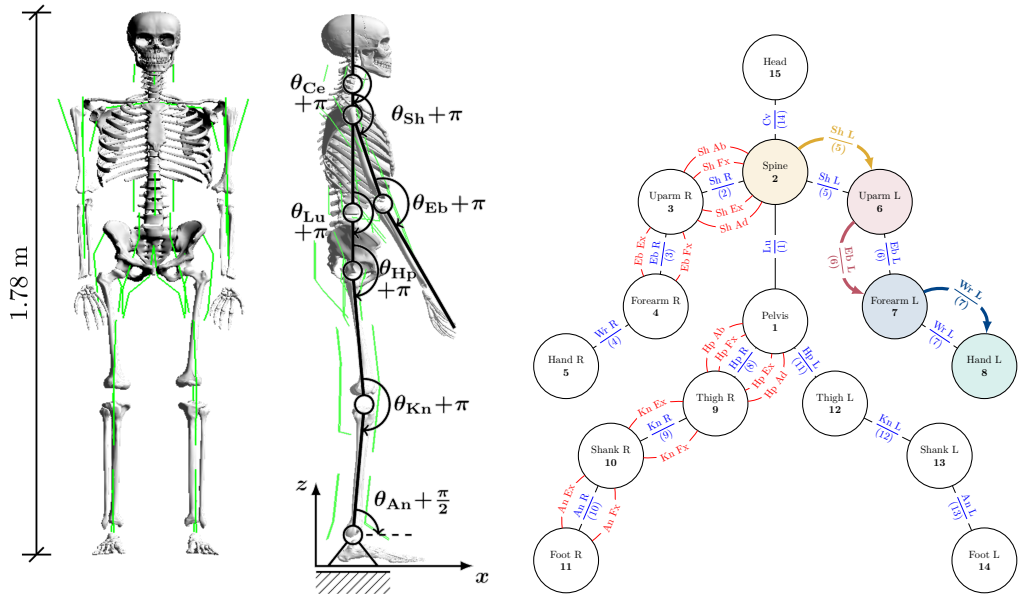


Figure 4.1: Left: The *digital human model* (DHM) used in this study is a simplified representation of the human body. The DHM consists of a chain of 15 rigid bodies (segments), which are connected by joints that allow movements in $n_{\mathrm{DoF}} = 20$ angular DoFs. The joints are actuated by $n_{\mathrm{MTU}} = 36$ string-like, massless Hill-type MTUs Right: Graph representation of the DHM with the vertexes being the bodies and the edges being the joints of the model. For the right leg and arm, additionally the muscles are displayed.

4.1 Graph-based model description

In this section the kinematic chain of the DHM, i.e. its base anthropometry, is represented in the form of a graph. In this graph, each rigid body of the model is related with each other body by the joint they are connected with. This can simply be written in the form of a matrix $\boldsymbol{A}$, in which each column and each row represents a rigid body of the model:

$$\boldsymbol{A} = \begin{bmatrix}
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 8 & 0 & 0 & 11 & 0 & 0 & 0 \\
1 & 0 & 2 & 0 & 0 & 5 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 14 \\
0 & 2 & 0 & 3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 3 & 0 & 4 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 4 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 5 & 0 & 0 & 0 & 0 & 6 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 6 & 0 & 7 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 7 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
8 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 9 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 9 & 0 & 10 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 10 & 0 & 0 & 0 & 0 & 0 \\
11 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 12 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 12 & 0 & 13 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 13 & 0 & 0 \\
0 & 14 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{bmatrix} \in \mathbb{R}^{n_{\mathrm{RGB}} \times n_{\mathrm{RGB}}} \tag{4.1}$$

The respective matrix entry a_{ij} for the i-th and j-th bodies exactly represents the k-th joint that connects i and j. This matrix is a slight modification of the so called *adjacency matrix*, which is well-known in the mathematical field of *graph-theory*, e.g. see Mesbahi

and Egerstedt (2010). The respective graph of the adjacency matrix (4.1) for the DHM *allmin* is displayed in Fig. 4.1 (right). Even though the adjacency matrix (4.1) seems to be very sparse and to have redundant information, a substantial amount of conclusions can be extracted from its structure.

Graph-searching, for example, can be efficiently executed on such a matrix to detect all joints that are crossed within a kinematic sub-chain from one body to another. To make this more comprehensive, imagine the kinematic sub-chain from the spine (ID = 2) to the hand (ID = 8). Starting at the shoulder body and following the row to the right, the joint with ID = 5 is found, that connects the spine to the body with column ID. The ID = 6 of the left uparm body is found when travelling the column upwards. Doing the same for the matrix row with the just obtained uparm ID, as a next body the forearm is found. Eventually the hand wit body ID = 8 is reached. In the following equation (4.2) and in Fig. 4.1 this is displayed by the arrows in the adjacency matrix and in the graph of the model.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Pelvis: 1	0	1	0	0	0	0	0	0	8	0	0	11	0	0	0
Spine: 2	1	0	2	0	0	5	0	0	0	0	0	0	0	0	14
Uparm R: 3	0	2	0	3	0	0	0	0	0	0	0	0	0	0	0
Forearm R: 4	0	0	3	0	4	0	0	0	0	0	0	0	0	0	0
Hand R: 5	0	0	0	4	0	0	0	0	0	0	0	0	0	0	0
Uparm L: 6	0	5	0	0	0	0	6	0	0	0	0	0	0	0	0
Forearm L: 7	0	0	0	0	0	6	0	7	0	0	0	0	0	0	0
Hand L: 8	0	0	0	0	0	0	7	0	0	0	0	0	0	0	0
Thigh R: 9	8	0	0	0	0	0	0	0	0	9	0	0	0	0	0
Shank R: 10	0	0	0	0	0	0	0	0	9	0	10	0	0	0	0
Foot R: 11	0	0	0	0	0	0	0	0	0	10	0	0	0	0	0
Thigh L: 12	11	0	0	0	0	0	0	0	0	0	0	0	12	0	0
Shank L: 13	0	0	0	0	0	0	0	0	0	0	0	12	0	13	0
Foot L: 14	0	0	0	0	0	0	0	0	0	0	0	0	13	0	0
Head: 15	0	14	0	0	0	0	0	0	0	0	0	0	0	0	0

(4.2)

Within the algorithmic implementation of the hierarchical control architecture such a graph-based model description and tree-searching has mainly two uses. Regarding the first, it forms the basis of the calculation of the muscles' moment arms (see Lsts. E.1 and E.2 in Appendix E.5). By knowing the ID's of the bodies at which the MTUs attach, graph-searching can be used for obtaining the joints that connect the bodies. Thereby the topological routing of the MTUs is available and the moment arms of the MTUs can be efficiently derived. Regarding the second use of graph-searching, it provides a general method for identifying a kinematic sub-chain of a limb in the regard of position control (see Sec. 3.3.3). By specifying the controlled body and its reference body, e.g. the control of the hand relative to the shoulder, graph searching identifies all joints that are responsible for control. For an algorithmic implementation for the identification of kinematic sub-chains, or to search a graph in general, efficient algorithms have been developed in the field of graph theory, such as presented by Dijkstra et al. (1959).

Table 4.1: List of bodies, joints and MTUs of the DHM. Each joint has either one or two DoFs and each DoF, except for the Wrs, is actuated by two MTUs. The ID numbers refer to the internal numbering that is used by the simulation software `demoa`.

Joint ID (DoF ID) ▼	Name RoM	Body ID	Name	Muscle ID	Name
				9	Lu lateral flexor, flexion
1	*Lumbar joint* (Lu)			10	Lu lateral extensor, extension
(1)	[−20 20]	1	Pelvis	11	Lu ventral flexor, flexion
(2)	[−20 20]	2	Spine	12	Lu ventral extensor, extension
				21	Sh extensor, extension
2	*Shoulder* (Sh) R			22	Sh flexor, flexion
(3)	[−20 20]	2	Spine	23	Sh abductor, abduction
(4)	[−20 20]	3	Uparm R	24	Sh adductor, adduction
3	*Elbow* (Eb) R	3	Uparm R	27	Eb flexor, flexion
(5)	[−20 20]	4	Forearm R	28	Eb extensor, extension
4	*Wrist* (Wr) R	4	Forearm R		
(6)	[−20 20]	5	Hand R		
				17	Sh extensor, extension
5	*Shoulder* (Sh) L			18	Sh flexor, flexion
(7)	[−20 20]	2	Spine	19	Sh abductor, abduction
(8)	[−20 20]	6	Uparm L	20	Sh adductor, adduction
6	*Elbow* (Eb) L	6	Uparm L	25	Eb flexor, flexion
(9)	[−20 20]	7	Forearm L	26	Eb extensor, extension
7	*Wrist* (Wr) L	7	Forearm L		
(10)	[−20 20]	8	Hand L		
				1	Hp extensor, extension
8	*Hip* (Hp) R			2	Hp flexor, flexion
(11)	[−20 20]	1	Pelvis	3	Hp abductor, abduction
(12)	[−20 20]	9	Thigh R	4	Hp adductor, adduction
9	*Knee* (Kn) R	9	Thigh R	31	Kn flexor, flexion
(13)	[−20 20]	10	Shank R	32	Kn extensor, extension
10	*Ankle* (An) R	10	Shank R	35	An extensor, extension
(14)	[−20 20]	11	Foot R	36	An flexor, flexion
				5	Hp extensor, extension
11	*Hip* (Hp) L			6	Hp flexor, flexion
(15)	[−20 20]	1	Pelvis	7	Hp abductor, abduction
(16)	[−20 20]	12	Thigh R	8	Hp adductor, adduction
12	*Knee* (Kn) R	12	Thigh R	29	Kn flexor, flexion
(17)	[−20 20]	13	Shank R	30	Kn extensor, extension
13	*Ankle* (An) R	13	Shank R	33	An extensor, extension
(18)	[−20 20]	14	Foot R	34	An flexor, flexion
				13	Ce lateral flexor, flexion
14	*Cervical joint* (Ce)			14	Ce lateral extensor, extension
(19)	[−20 20]	2	Spine	15	Ce ventral flexor, flexion
(20)	[−20 20]	15	Head	16	Ce ventral extensor, extension

In this chapter it is demonstrated how the control architecture can be used to perform coordinated movements in the control space of joint angles. The DHM from Ch. 4 is used as simulation subject. In the three subsequent Secs. 5.1, 5.2 and 5.3, the joint angles of an arm, a leg and of the trunk of the DHM are controlled. For performing the control task in isolated conditions, the DHM is fixed to the *world* at its pelvis joint. This means that even in the presence of gravity ($g = -9.8065\frac{\mathrm{m}}{\mathrm{s}^2}$) the position of the pelvis stays the same. Attached to the pelvis, each leg is able to freely swing with the two DoFs in the Hp and the trunk balances in upright position at the Lu. Details on the simulation software and the used solver are presented in Appendix D.1.

The model is actuated in the control space of joint angles by providing a postural plan $\boldsymbol{\theta}^{\mathrm{des}}(t)$ (see Sec. 3.3.1)[1]. By following the structured hierarchy of the control architecture (Fig. 3.4), MTU stimulations (3.6) are eventually generated. These stimulation signals actuate the DHM in order to achieve the postural plan, i.e. $\boldsymbol{\theta}(t) \to \boldsymbol{\theta}^{\mathrm{des}}(t)$. In the here presented examples of joint angle control, the postural plan is provided by discrete equilibrium poses. That is, for each joint angle a discrete desired value is assigned for a specific time interval. This allows to investigate the transient and asymptotic dynamics of the closed loop control system.

5.1 Control of the lower limb joint angles

For the control of the lower limb joint angles, in this section, an angle controller for the right Hp FE, Hp AA, Eb FE and An FE is defined. The desired movement is specified in terms of discrete set-points of desired joint angles as follows:

$$\boldsymbol{\theta}^{\mathrm{des}}_{\mathrm{Upex}}(t) = \begin{cases} \begin{bmatrix} \theta_{\mathrm{HpR,FE}} & \theta_{\mathrm{HpR,AA}} & \theta_{\mathrm{KnR}} & \theta_{\mathrm{AnR}} \end{bmatrix}^T \\ \begin{bmatrix} 0.0 & 0.0 & 0.0 & 0.0 \end{bmatrix}^T, \quad t \in (\ 0.0\mathrm{s} \ \dots \ 1.5\mathrm{s}\ ] \\ \begin{bmatrix} -40.0 & 30.0 & 110.0 & -20 \end{bmatrix}^T, \quad t \in (\ 1.5\mathrm{s} \ \dots \ 3.0\mathrm{s}\ ] \\ \begin{bmatrix} -110.0 & 10.0 & 30.0 & 30.0 \end{bmatrix}^T, \quad t \in (\ 3.0\mathrm{s} \ \dots \ 4.5\mathrm{s}\ ] \\ \begin{bmatrix} -10.0 & 40.0 & 70.0 & 0.0 \end{bmatrix}^T, \quad t \in (\ 4.5\mathrm{s} \ \dots \ 6.0\mathrm{s}\ ] \end{cases}.$$

In the first time interval $t \in \big[0.0\mathrm{s} \quad 1.5\mathrm{s}\big]$, the model is set to stay at the initial, neutral position. In the second time interval, the Hp is slightly flexed and abducted, the Kn is flexed and the An is flexed. In the third time interval, the Hp is fully flexed, Hp AA and Kn FE are reduced and the An is extended. The pose defined for the fourth time interval

[1] For methods of setting desired joint angles in the simulation framework `demoa` refer to Appendix E.4

corresponds to an abducted Hp and a flexed Kn. The control parameters are chosen to be the same for all poses and are listed in Tab. 5.1. All neural latency delay times $\boldsymbol{\delta}$ are set to zero.

The different body postures in the achieved steady-states are shown in the upper row in Fig. 5.1. In the second row, the right-leg angle-stimulation Jacobian $J^{\lambda\theta}$ from (3.9) is shown as heat-map matrix. This state-dependent matrix maps joint angle changes to CE length changes and thereby resolves the redundancy of the musculo-skeletal system. Based on the sign of the respective matrix entry, the kinesiologic effect of a certain muscle on a specific joint (flexor/extensor) can be identified (see colour coding on the right and joint angle notation in Fig. 4.1). In the line plots below, trajectories of joint angles, knee-MTU CE-lengths and stimulations are shown. It is clearly seen that the system is stable and approaches the desired values of joint angles, i.e. $\boldsymbol{\theta}(t) \to \boldsymbol{\theta}^{\mathrm{des}}(t)$. In its transient dynamic, the system is accelerated and smoothly approaches the desired states. Some overshooting and oscillations are visible for the Hp joints, which can mainly be traced back to inertia effects of the 'heavy' leg. In the desired MTU lengths $\boldsymbol{\lambda}^{\theta}$ that are demanded by the transformational layer, oscillations occur. These oscillations are passed through to the MTU stimulations and are eventually damped out by the intrinsic velocity characteristics of the MTUs, as well as by inertia of the rigid bodies.

5.2 Control of the upper limb joint angles

For the control of the upper limb joint angles, in this section, an angle controller for the left Sh FE, Sh AA and Eb FE is defined. The desired movement is specified in terms of discrete set-points of desired joint angles as follows:

$$\boldsymbol{\theta}^{\mathrm{des}}_{\mathrm{Upex}} = \begin{cases} [\ \theta_{\mathrm{ShL,FE}} \quad \theta_{\mathrm{ShL,AA}} \quad \theta_{\mathrm{EbL,FE}}\ ]^T & \\ [\ 0.0 \quad 0.0 \quad 0.0\ ]^T, & t \in (\ 0.0\mathrm{s} \ \dots \ 1.5\mathrm{s}\ ] \\ [\ -10.0 \quad 40.0 \quad -70.0\ ]^T, & t \in (\ 1.5\mathrm{s} \ \dots \ 3.0\mathrm{s}\ ] \\ [\ -90.0 \quad 10.0 \quad -30.0\ ]^T, & t \in (\ 3.0\mathrm{s} \ \dots \ 4.5\mathrm{s}\ ] \\ [\ -40.0 \quad 60.0 \quad -110.0\ ]^T, & t \in (\ 4.5\mathrm{s} \ \dots \ 6.0\mathrm{s}\ ] \end{cases}.$$

In the first time interval, the arm is held in neutral position. In the second time interval the arm is abducted to 40° and the elbow is flexed to −70°. After that, the arm is adducted back close to the body and flexed to 90° and the elbow flexion is released to 30°. In the final time interval the arm is flexed (90°) and abducted (60°) and the elbow is flexed (110°). The control parameters are given in Tab. 5.2.

In the upper row in Fig. 5.2, the body postures are shown after they have confidently reached equilibrium. Also in Fig. 5.2, excerpts of the angle-length Jacobian $J^{\lambda\theta}$ are shown, as well as plots of joint angles $\boldsymbol{\theta}_{\mathrm{Upex}}$ and CE-lengths $\boldsymbol{l}^{\mathrm{CE}}_{\mathrm{Upex}}$ together with their desired values $\boldsymbol{\theta}^{\mathrm{des}}_{\mathrm{Upex}}$, λ_{Upex} and the stimulations $\boldsymbol{u}(t)$ that are produced by the control architecture.

Table 5.1: Control parameters of the angle controller of the lower limb.

Joint angle Name	$P_{\theta\lambda}$	$I_{\theta\lambda}$	$D_{\theta\lambda}$	P_{θ}	I_{θ}	D_{θ}	MTU Name	κ	$u^{\mathrm{coc}}_{\mathrm{ref}}$
hip FE	0.01	0.01	0.001	0.1	0.1	0.01	Hp *flexor, flexion* (Fx)	2.25	0.1
	0.01	0.01	0.001	0.1	0.1	0.01	Hp *extensor, extension* (Ex)	2.25	0.1
hip AA	0.01	0.01	0.001	0.1	0.1	0.01	Hp *abductor, abduction* (Ab)	2.25	0.1
	0.01	0.01	0.001	0.1	0.1	0.01	Hp *adductor, adduction* (Ad)	2.25	0.1
knee FE	0.01	0.01	0.001	0.1	0.1	0.01	Kn Fx	2.25	0.1
	0.01	0.01	0.001	0.1	0.1	0.01	Kn Ex	2.25	0.1
ankle FE	0.01	0.01	0.001	0.1	0.1	0.01	An Fx	2.25	0.1
	0.01	0.01	0.001	0.1	0.1	0.01	An Ex	2.25	0.1

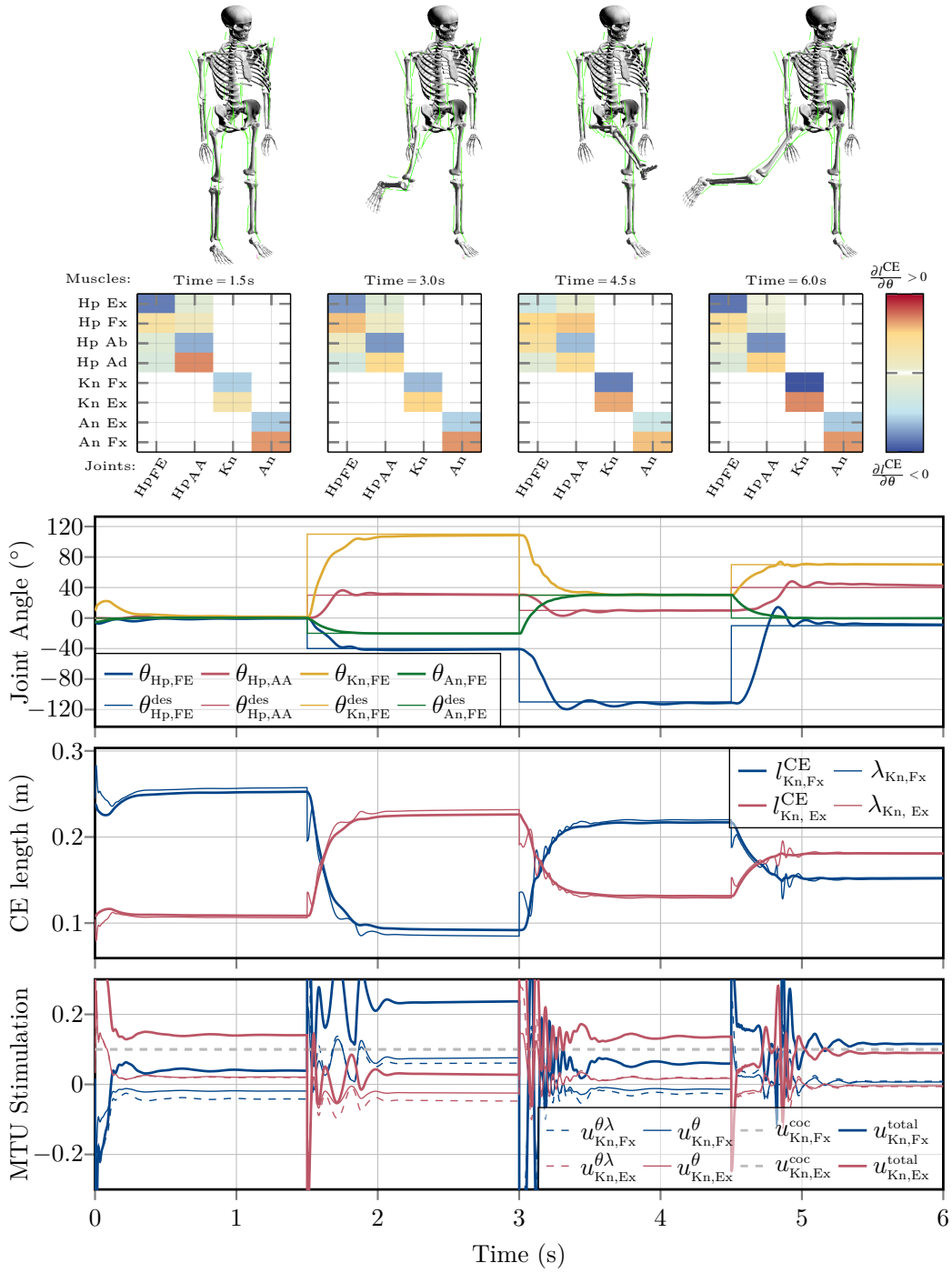


FIGURE 5.1: Simulation results of lower limb joint angle control. In the first row, the body postures after reaching the desired joint angles are shown. In the second row, the Jacobian entries of $J^{\lambda\theta}$ are displayed that are used by the control architecture for layer transformation and to resolve the muscle-joint redundancy. In the bottom three line-plots, the current and desired joint angles of the lower limb joints, the current and desired CE-lengths and the MTU stimulations of the Kn MTUs are shown. All desired values are reached confidently and the MTU stimulations are adjusted around their reference co-contraction of 0.1. Even though some oscillations in the MTU stimulations are present after a new pose is set, they are quickly damped out.

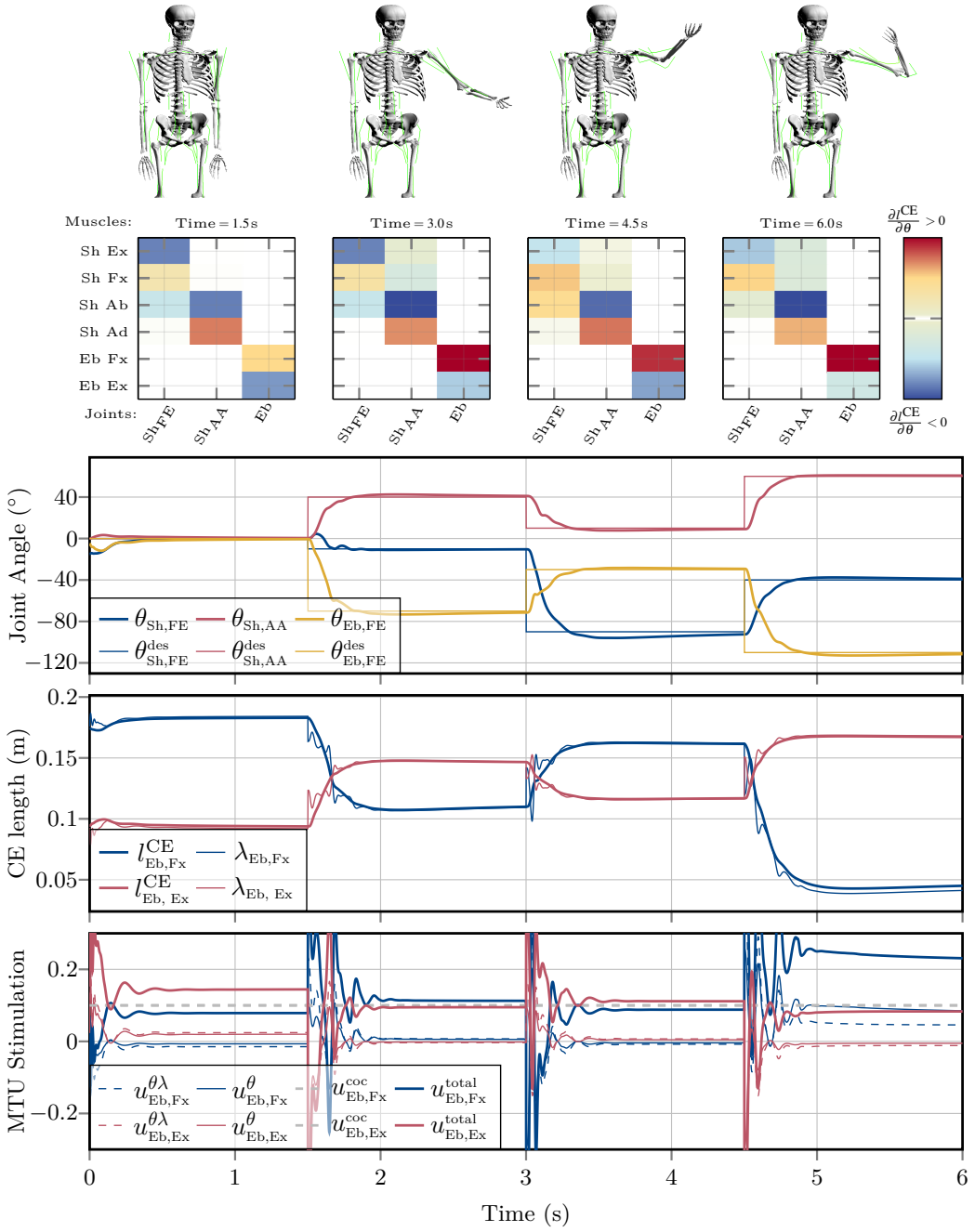


Figure 5.2: Simulation results of joint angle control of an upper extremity. In the upper row the body posture is shown after reaching the equilibrium of the desired pose. In the second row, the Jacobian entries of $J^{\lambda\theta}$ are displayed to visualize how the postural plan is translated into desired CE-lengths. The state dependency of the Jacobian matrix can be identified by different colour shadings for different body postures. The Sh FE MTUs, for example, have an increased effect in AA direction when the Hp joint deviates from the neutral position. In the line-plots in the bottom three rows, trajectories of joint angles, CE lengths and MTU stimulations are shown together with the desired values and the reference co-contraction stimulation. It is seen that the desired joint angles are reached quickly and confident for the used controller parametrisation. The desired CE-lengths that are generated by the transformational layer are mostly reached except for some spikes that are damped out by the intrinsic properties of the MTU dynamics.

5.3 Control of the trunk joint angles

The control of the trunk angles is completely analogue to the control of the upper and lower limb joint angles as described in the previous Secs. 5.1 and 5.2. The desired values for the angle controller of the trunk are chosen to follow the discrete trajectories of

$$\boldsymbol{\theta}^{\text{des}}_{\text{Trunk}} = \begin{cases} [\ \theta_{\text{Lu,lFE}} & \theta_{\text{Lu,vFE}} & \theta_{\text{Ce,lFE}} & \theta_{\text{Ce,vFE}}\ ]^T \\ [\ 0.0 & 0.0 & 0.0 & 0.0\ ]^T, & t \in (\ 0.0\text{s} & \dots & 1.5\text{s}\ ] \\ [\ -30.0 & 30.0 & 0.0 & 0\ ]^T, & t \in (\ 1.5\text{s} & \dots & 3.0\text{s}\ ] \\ [\ 0.0 & 0.0 & -30.0 & 30.0\ ]^T, & t \in (\ 3.0\text{s} & \dots & 4.5\text{s}\ ] \\ [\ 0.0 & 0.0 & 0.0 & 0.0\ ]^T, & t \in (\ 4.5\text{s} & \dots & 6.0\text{s}\ ] \end{cases}.$$

The final pose corresponds to the first pose and in-between, isolated front-left bendings of the Lu and Ce are demanded. This is also shown in Fig. 5.3 by screenshots of the desired equilibrium poses and by time-plots of the trunk joint angles, CE-lengths and MTU-stimulations of the ventral flexion of the Ce.

It can be observed that the Ce vFE does not reach its desired angle and also that the respective muscles do not reach their desired lengths. This can be traced back to a miss-parametrised MTU; either the agonistic MTU cannot be further contracted or the antagonist cannot be further stretched. After the pose that was not reached, the Ce vFE angle only slowly reacts to the desired set-point, due to an over-saturation from the previous pose. The control parameters for the trunk joint angle controller are listed in Tab. 5.3.

Table 5.2: Control parameters of the angle controller of the right upper limb.

Joint angle Name	$P_{\theta\lambda}$	$I_{\theta\lambda}$	$D_{\theta\lambda}$	P_θ	I_θ	D_θ	MTU Name	κ	$u^{\text{coc}}_{\text{ref}}$
shoulder FE	0.01	0.01	0.001	0.1	0.1	0.01	Sh Fx	2.25	0.1
							Sh Ex	2.25	0.1
shoulder AA	0.01	0.01	0.001	0.1	0.1	0.01	Sh Ab	2.25	0.1
							Sh Ad	2.25	0.1
elbow FE	0.01	0.01	0.001	0.1	0.1	0.01	Eb Fx	2.25	0.1
							Eb Ex	2.25	0.1

Table 5.3: Control parameters of the angle controller of the trunk.

Joint angle Name	$P_{\theta\lambda}$	$I_{\theta\lambda}$	$D_{\theta\lambda}$	P_θ	I_θ	D_θ	MTU Name	κ	$u^{\text{coc}}_{\text{ref}}$
Lu lFE	0.01	0.01	0.001	0.1	0.1	0.01	Lu *lateral flexor, flexion* (lFx)	2.25	0.1
							Lu *lateral extensor, extension* (lEx)	2.25	0.1
Lu vFE	0.01	0.01	0.001	0.1	0.1	0.01	Lu *ventral flexor, flexion* (vFx)	2.25	0.1
							Lu *ventral extensor, extension* (vEx)	2.25	0.1
Ce lFE	0.01	0.01	0.001	0.1	0.1	0.01	Ce lFx	2.25	0.1
							Ce lEx	2.25	0.1
Ce vFE	0.01	0.01	0.001	0.1	0.1	0.01	Ce vFx	2.25	0.1
							Ce vEx	2.25	0.1

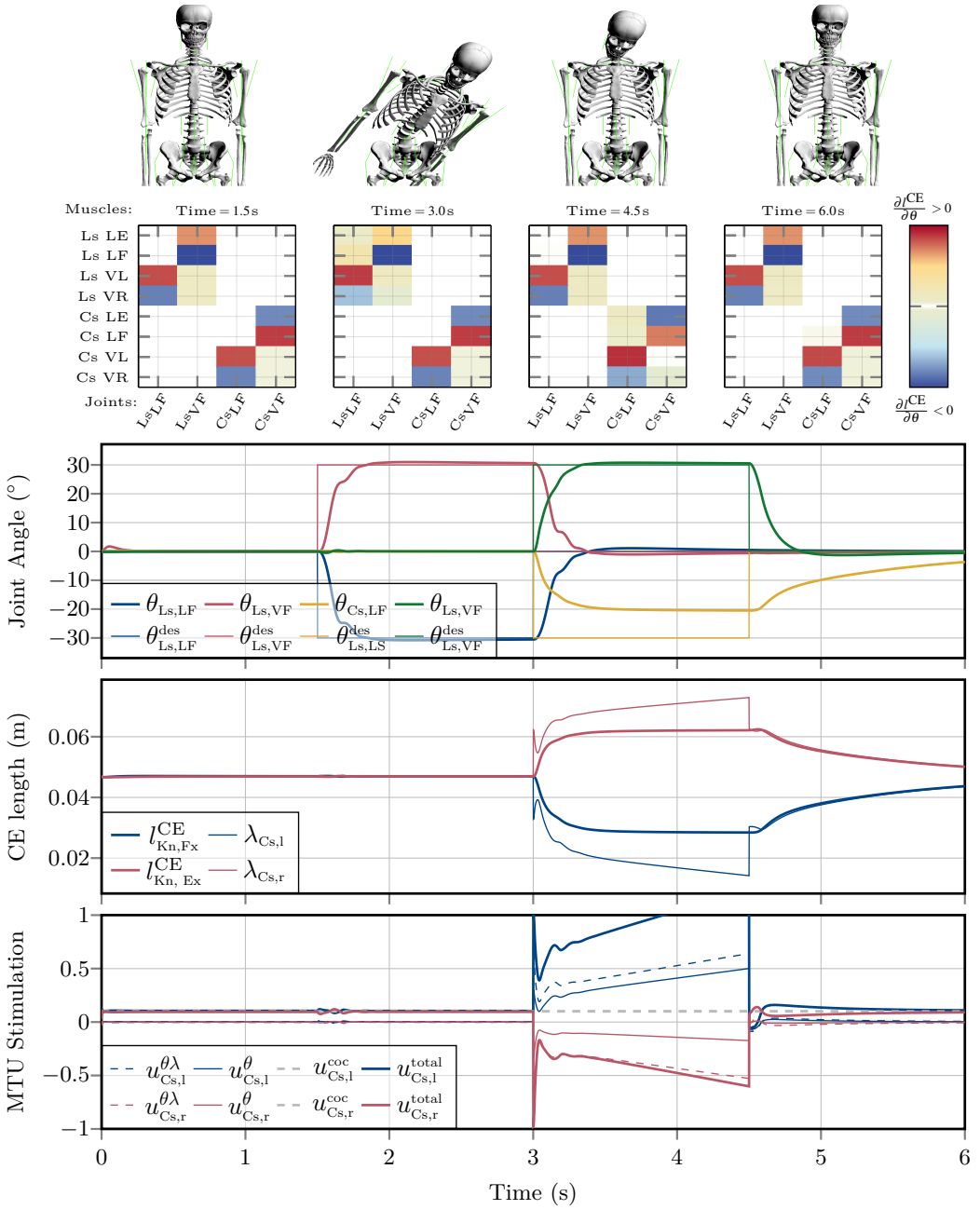


Figure 5.3: Simulation results of trunk joint angle control. In the first row the body postures of the desired movement are displayed as screenshots of the simulation's animation. In the second row the Jacobian entries of $J^{\lambda\theta}$ of the trunk angles are illustrated as they are used by the control architecture for transforming joint angle changes to CE length changes. The three bottom line-plots show the overall control behaviour in terms of joint angle trajectories and a detailed view for the Ce vFE MTUs. In the third time-interval, the Ce vFE joint does not reach its desired value. The respective agonistic MTU is unable to reach the desired length $\lambda^{\theta}_{Ce,vFx}$

To demonstrate the function of the conceptional torque controller (Sec. 3.3.2), it is here used to perform human-like upright stance by implementing an external joint-torque-based control concept on the musculo-skeletal model. This simulation study is also part of the publication by Walter et al. (2021a). There, the desired torques that are needed as input for the conceptional torque controller are obtained from the model of joint angle-torque characteristics from Günther and Wagner (2016). This concept determines joint torques that stabilise the mechanical system (2.33) around an upright position. By using such a torque concept in the conceptional layer and feeding its output to the hierarchical control architecture, MTU stimulations are generated that eventually lead to the desired joint torques demanded by the stabilising torque-based control concept.

This approach is used in a total of four simulations to demonstrate the control behaviour for the task of upright stance; of an unperturbed system (Sec. 6.1.1); with the additional criteria of joint-based co-contraction (Sec. 6.1.2); and with the synthetisation of a squat movement (Sec. 6.1.3), which is basically an additional control objective added to the balancing task. Details on the simulation software and the used solver are presented in Appendix D.1.

6.1 Conceptional task formulation in terms of joint torques

In a nutshell, Günther and Wagner (2016) proposed to approximate human upright stance as a three-segment inverted pendulum model. With this assumption they were able to predict that upright stance could be stabilised by combining joint-level stiffnesses in the An, Kn, and Hp joint with an active positional contribution based on the weighted deviation of the body's *center of mass* (COM) in the sagittal plane. This leads to the following equations for the desired joint torques to be implemented in (3.34):

$$\begin{aligned}\boldsymbol{\tau}^{\mathrm{des}}_{\mathrm{L/R}}(t) = &\begin{bmatrix} k_{\mathrm{Hp,FE,L/R}} & 0 & 0 \\ 0 & k_{\mathrm{Kn,L/R}} & 0 \\ 0 & 0 & k_{\mathrm{An,L/R}} \end{bmatrix} \cdot \begin{bmatrix} (\theta_{\mathrm{Hp,FE,L/R}}(t) - \theta_{\mathrm{Hp,FE,L/R,0}}) \\ (\theta_{\mathrm{Kn,L/R}}(t) - \theta_{\mathrm{Kn,L/R,0}}) \\ (\theta_{\mathrm{An,L/R}}(t) - \theta_{\mathrm{An,L/R,0}}) \end{bmatrix} \\ &+ \begin{bmatrix} g_{\mathrm{Hp,L/R}} \cdot (x_{\mathrm{COM}}(t) - x^{\mathrm{des}}_{\mathrm{COM}}(t)) \\ 0 \\ 0 \end{bmatrix},\end{aligned} \tag{6.1}$$

with x_{COM} and $x^{\mathrm{des}}_{\mathrm{COM}} := \frac{x_{\mathrm{An,l}}(t)+x_{\mathrm{An,r}}(t)}{2} + x_{\mathrm{COM},0}$ being the actual and desired positions of the body's COM in the direction of the x-axis, respectively, $\theta_{i,0}$ being the nominal angles for

the left and right (index L/R) Hp, Kn and An FE joints and k_i being the respective single-joint stiffness gains. These desired torques $\boldsymbol{\tau}^{\text{des}}$ (6.1) are the input to the hierarchical control architecture: $\boldsymbol{\tau}^{\text{des}}$ is compared to the actual joint torques $\boldsymbol{\tau}^{\text{MTU}}$ generated by the muscles to calculate the torque error $\boldsymbol{\tau}^{\text{err}}$ (3.34), which is the input to the torque control law (3.35). The resulting controller output is then transformed via the torque-angle Jacobian matrix $J^{\theta\tau}$ (3.36, 3.38) to obtain a vector of desired of joint angles $\boldsymbol{\theta}_{\tau}^{\text{des}}$ (postural plan) with (3.39). As this conceptional controller is only used to actuate the lower limb *flexion-extension* (FE) DoFs, the remaining DoFs of the full-body model—left and right (L/R) Ces, Shs, Ebs, Lus and Hp AA DoFs—are controlled by setting the postural plan directly to $\boldsymbol{\theta}_{j}^{\text{des}} = 0$ ($j \in \{\lceil\text{HpL/R, FE}\rfloor, \lceil\text{HpL/R, AA}\rfloor, \lceil\text{HpL/R, FE}\rfloor, \lceil\text{AnL/R, FE}\rfloor\}$). The completely filled vector of desired joint angles is then fed into the hierarchical θ_λ-controller (3.8) and the direct θ-controller (3.24), respectively, to generate the stimulation signal (3.6).

6.1.1 Simulation task: quiet upright stance

In the first simulation study, the task of upright stance is performed by conceptional planning and structural execution using the presented hierarchical control architecture to drive the DHM by means of MTU stimulations. The torque-angle characteristic (6.1) is straightforwardly used in the conceptional layer (3.34). The parameter values are chosen such that the control concept (6.1) is symmetric on the left (index L) and the right (index R) body side. The joint-stiffness gains k_i for the lower-limb FE DoFs in (6.1) are chosen in accordance to Günther and Wagner (2016) to be the critical single-joint stiffnesses, calculated by $k_i = m_i \cdot g \cdot h_i$, where m_i is the total mass above the respective joint, g is the gravitational acceleration and h_i is the distance of the system's COM above the respective joint. The numerical values of these stiffness gains k_i for the model used in this study, as well as the chosen value for the positional weight $g_{\text{Hp,L/R}}$, are listed in Tab. 6.1. The resulting joint torques are calculated w.r.t. the nominal joint-angle configuration of $\theta_{\text{Hp,L/R,0}} = 0°$, $\theta_{\text{Kn,L/R,0}} = 5°$ and $\theta_{\text{An,L/R,0}} = -5°$. In accordance to this nominal joint-angle configuration, the offset of the desired COM position is set to $x_{\text{COM,0}} = 2.86$cm. The co-contraction parameters and the PID-control parameters for the conceptional τ-controller, θ_λ-controller

Table 6.1: Initial conditions and control parameters used for the computational synthetisation of quiet upright stance with the hierarchical distributed PID-control laws. The conceptional joint-torque controllers are only active for the lower limb DoFs. For the conceptional plan of desired joint torques the additional position based parameter $g_{\text{Hp,L/R}} = -400$ N is used (see (6.1)) with the set-point for the COM of $x_{\text{COM,0}} = 2.86$ cm anterior to the mean ankle joint position. For the structural control law, for each of the k MTUs, the control parameters $p_{\lambda,k} = 1.0/l_{\text{opt},k}^{\text{CE}}$ and $d_{\lambda,k} = 0$ are chosen. As an additional initial condition, the pelvis body is positioned at $X_P(t=0) = [0\ 0\ 0.9954]^T$, such that the feet are placed at the ground contacts. The initial activations of the MTUs are in steady-state with $a_k = 0.1$ for each of the $k = 1...36$ MTUs, corresponding to the co-contraction that is parametrised by $\zeta_k^u = \zeta^u = 0.1$ and $\zeta_j^\theta = \zeta^\theta = 0.0$.

	Conceptional Control	τ-Controller			θ_λ-Controller			θ-Controller			Initial Conditions
Joint	k $\left[\frac{\text{Nm}}{\text{rad}}\right]$	p_τ [-]	I_τ $\left[\frac{1}{\text{s}}\right]$	d_τ [s]	p_{θ_λ} [-]	$\text{I}_{\theta_\lambda}$ $\left[\frac{1}{\text{s}}\right]$	d_{θ_λ} [s]	p_θ [-]	I_θ $\left[\frac{1}{\text{s}}\right]$	d_θ [s]	$\theta(t=0)$ [°]
Lu AA	–	–	–	–	0.01	0.01	0.001	0.1	0.1	0.01	0
Lu FE	–	–	–	–	0.01	0.01	0.001	0.1	0.1	0.01	0
Ce AA	–	–	–	–	0.01	0.01	0.001	0.1	0.1	0.01	0
Ce FE	–	–	–	–	0.01	0.01	0.001	0.1	0.1	0.01	0
Sh AA (L/R)	–	–	–	–	0.01	0.01	0.001	0.1	0.1	0.01	0
Sh FE (L/R)	–	–	–	–	0.01	0.01	0.001	0.1	0.1	0.01	0
Eb FE (L/R)	–	–	–	–	0.01	0.01	0.001	0.1	0.1	0.01	0
Hp AA (L/R)	–	–	–	–	0.01	0.0	0.002	0.05	0.0	0.01	0
Hp FE (L/R)	172	2.0	30	0.0	1.5	1.0	0.1	1.5	1.0	0.1	0
Kn FE (L/R)	467	0.75	15	0.0	0.25	1.0	0.0035	0.25	1.0	0.0035	5
An FE (L/R)	784	3.0	50	0.0	0.5	1.0	0.0035	0.5	1.0	0.0035	−5

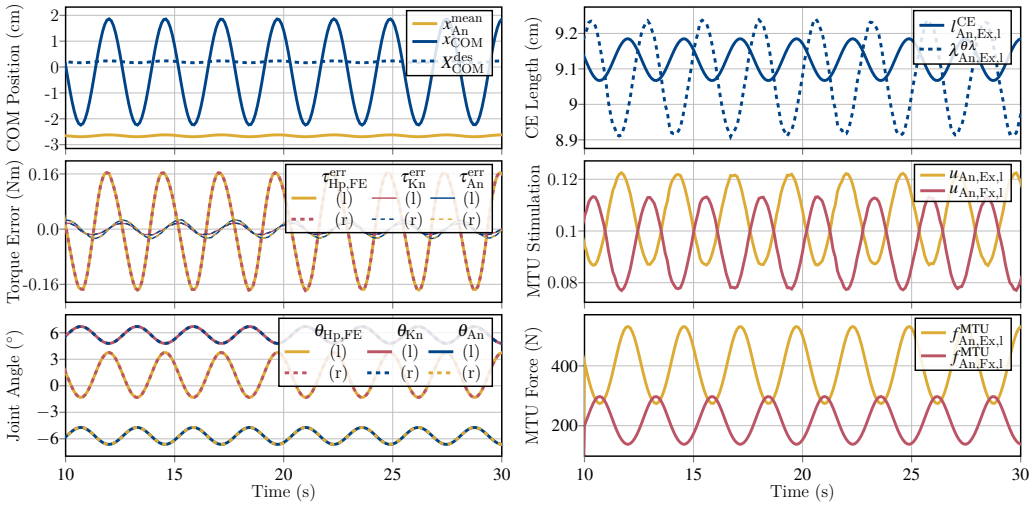


FIGURE 6.1: Simulation results of quiet upright stance. From top to bottom the trajectories of the body's COM, the joint angle sway of left and right (l/r) Hp, Kn and An FE joints, the respective control errors on joint torques, the control error of CE lengths for the An Ex, the resulting MTU stimulations and forces for the An Ex and Fx MTUs are displayed. The body's COM position is anterior to the ankle joints at all times (Smith, 1957).

and θ-controller are listed in Tab. 6.1. The control parameters for the low-level λ-controller are chosen to $p_{\lambda,k} = 1.0/l^{\mathrm{CE}}_{\mathrm{opt},k}$ and $d_{\lambda,k} = 0$ for all k MTUs. The sensor delays δ_λ, δ_θ and δ_τ are all set to 0 to be in accordance with the original concept paper (Günther and Wagner, 2016).

With these parameters a stable attractor was found that allowed the model to fulfil the requested task of upright stance. For the time from 10 s to 30 s, the COMs positional error, the lower-limb joint angles, the (high-level) control error of joint torques, the actual and desired (as demanded by the transformational layer) CE lengths of the An Ex, as well as signals of the generated MTU stimulations and MTU forces of the An Fx and Ex, are shown in Fig. 6.1. During this initial time-interval, the maximum positional error in the sagittal plane of the body's COM w.r.t. its desired set-point $x^{\mathrm{des}}_{\mathrm{COM}}(t)$ is about 2.5 cm. The maximum joint angle sway is about 5° in Hp FE and 1.9° in An and Kn FE. Based on the used convention of joint rotations (see Fig. 4.1), the Kn and the An joints show in-phase behaviour, where the Hp FE joint angle is in anti-phase with the other joints, with a main oscillation frequency of 0.38 Hz. The MTU stimulations vary around the reference co-contraction value of $u^{\mathrm{coc}}_{\mathrm{ref}} = 0.1$. Compared to this constant co-contraction value, the An Ex stimulation is inhibited with a maximum reduction of about 0.13% and the An Ex is stimulated up to an additional 22.5%. These stimulation signals stabilise the mechanical system of the triple inverted pendulum. The absolute control error on joint torques has a maximal value of ±0.17 Nm in the Hp joint, and ±0.017 Nm, and ±0.027 Nm in the Kn and An joints, respectively. Compared to the maximal absolute values of MTU generated torques of $|\boldsymbol{\tau}^{\mathrm{MTU}}_{\mathrm{max,\,Hp}}| = 5.84$ Nm, $|\boldsymbol{\tau}^{\mathrm{MTU}}_{\mathrm{max,\,Kn}}| = 13.9$ Nm and $|\boldsymbol{\tau}^{\mathrm{MTU}}_{\mathrm{max,\,An}}| = 22.3$ Nm, the relative control errors correspond to an estimated maximal deviation of about 2.9% for the Hp, 0.2% for the Kn and < 0.1% for the An joint. The hierarchical control architecture transforms this error signal firstly to the transformational layer and subsequently to the structural layer and, thus, produces desired muscle lengths $\boldsymbol{\lambda}^{\theta}$. The maximal error of the actual CE length $l^{\mathrm{CE}}_{\mathrm{Hp,Fx}}$ of the Hp Fx muscle and the desired value $\lambda^{\theta}_{\mathrm{Hp,Fx}}$ occurs shortly before the maximal positive body sway is reached. Further details are presented in Walter et al. (2021a).

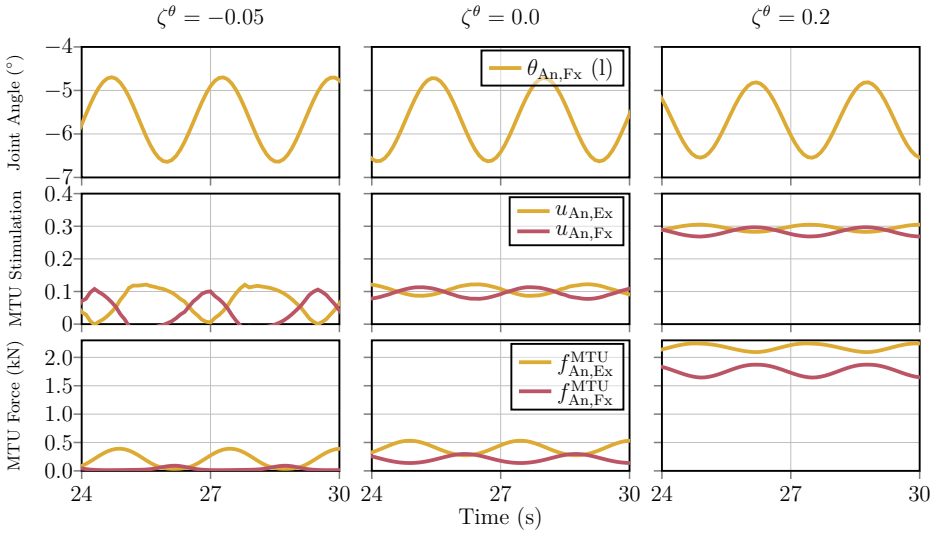


Figure 6.2: Simulation results of quiet upright stance with joint-based co-contraction variations. The middle graphs show the default ($\zeta^\theta_j = \zeta^\theta = 0.0$), where in the left column a joint-based co-contraction of $\zeta^\theta_j = \zeta^\theta = -0.05$ in all leg joints ($j \in [\text{Hp}_{\text{FE,L/R}}, \text{Hp}_{\text{AA,L/R}}, \text{Kn}_{\text{L/R}}, \text{An}_{\text{L/R}}]$) is used, and $\zeta^\theta_j = \zeta^\theta = 0.2$ in the right column. In both variations, the MTU stimulations are adjusted according to ζ^θ and so are the respective MTU forces. The joint-based co-contraction is hereby based on the null-space of the Jacobian transformations. Due to the approximations of the associated angle-stimulation Jacobian $J^{u\theta}$ (3.25), the resulting joint trajectories are slightly different.

6.1.2 Simulation task: joint-based co-contraction

Due to the redundant nature of the many MTUs acting on the fewer joints, an uncontrolled manifold arises that can be used to fulfil joint-based co-contraction constraints as described in Sec. 3.2.4. The simulations of upright stance with joint-based co-contraction is achieved by adding contributions $\boldsymbol{u}^{\text{coc}}_\theta$ to the MTUs' stimulation signals, of all MTUs acting on the same joint, exactly without interfering with the movement task of upright stance. As outlined in Sec. 3.2.4, such a stimulation contribution is obtained from the null-space of the pseudo inverse of the angle-stimulation Jacobian $J^{u\theta}$.

Two simulations are performed to examine the effects of setting the joint co-contraction parameters ζ^θ_j for the left and right (index L/R) Hp FE, Hp AA, Kn and An joints ($j \in [\text{Hp}_{\text{FEL/R}}, \text{Hp}_{\text{AAL/R}}, \text{Kn}_{\text{L/R}}, \text{An}_{\text{L/R}}]$). One simulation with an increased joint-based co-contraction value of $\zeta^\theta_j = \zeta^\theta = 0.2$ and one with a reduced value of $\zeta^\theta_j = \zeta^\theta = -0.05$, for all of the j lower limb joints, respectively. All other model and controller parameters remain the same as for the first simulation study from Sec. 6.1.1 and are listed in Tab. 6.1.

The results of variations in leg-joint co-contraction are shown in Fig. 6.2 for the time interval from 10 s to 30 s. In the left picture a simulation using the parameter $\zeta^\theta_j = -0.05$ is shown and $\zeta^\theta_j = 0.2$ on the right. These are compared to a simulation without joint-based co-contraction ($\zeta^\theta_j = 0$) in the middle. It can be seen, that the MTUs' stimulations of the Hp Fx and Ex are adapted accordingly by the joint co-contraction parameter, while upright stance is maintained. The An FE joint angle trajectories are similar for the different simulations, but slightly vary in amplitude and phase. This suggests, that the null-space projection $0 \stackrel{!}{=} \partial\boldsymbol{\theta} = J^{u\theta\dagger}\partial\boldsymbol{u}$ (3.32) behaves as intended within the presented hierarchical control architecture.

6.1.3 Simulation task: squat movement

To demonstrate the flexibility of the presented hierarchical control architecture a third and final simulation study was performed, which adds a squat movement on top of the free-balance controller. To achieve this squat movement, only changes in the conceptional layer are needed. More precisely, instead of having constant values of $\theta_{\mathrm{Hp,L/R,0}}$, $\theta_{\mathrm{Kn,L/R,0}}$ and $\theta_{\mathrm{An,R/R,0}}$ for the nominal Hp, Kn and An FE angles (6.1), a linear change over time of these nominal angles is employed. To keep the position of the pelvis body in x-direction approximately constant during the squat movement, the nominal Kn and Hp FE angles were chosen (trigonometrically) dependent on the nominal An angle:

$$\theta_{\mathrm{An,L/R,0}}(t) = \begin{cases} -5^\circ & t \leq t^* \\ -5^\circ - \frac{\theta^*_{\mathrm{An}}}{\Delta t} \cdot (t - t^*) & t > t^* \\ -5^\circ - \theta^*_{\mathrm{An}} & \text{for} \quad t > t^* + \Delta t \\ -5^\circ - \theta^*_{\mathrm{An}} \cdot \left(1 + \frac{t-(t^*+2\Delta t)}{\Delta t}\right) & t > t^* + 2\Delta t \\ -5^\circ & t > t^* + 3\Delta t, \end{cases} \tag{6.2}$$

$$\theta_{\mathrm{Kn,L/R,0}}(t) = \sin^{-1}\left(-\frac{L_{\mathrm{S}}}{L_{\mathrm{T}}} \cdot \sin(\theta_{\mathrm{An,L/R,0}}(t))\right) - \theta_{\mathrm{An,L/R,0}}(t) - \theta_{\mathrm{An,L/R,0}}\big|_{t=0} \tag{6.3}$$

$$\theta_{\mathrm{Hp,L/R,0}}(t) = -\theta_{\mathrm{Kn,L/R,0}}(t) - \theta_{\mathrm{An,L/R,0}}(t), \tag{6.4}$$

where L_{T} and L_{S} are the segment lengths of the thigh (T) and shank (S) bodies, respectively (see Appendix C), θ^*_{An} is the final An angle offset of the squat, t^* is the time instance initiating the squat and Δt is the interval for each of the three squat phases, i.e. the downwards movement, the squat stance and the upwards movement. With this simple linear approach two squat movements have been synthesised *in-silico*; a slow squat ($t^* = 10$ s, $\Delta t = 5$ s, $\theta^*_{\mathrm{An}} = -15^\circ$) and a fast squat ($t^* = 16.75$ s, $\Delta t = 0.5$ s, $\theta^*_{\mathrm{An}} = -10^\circ$). Beside these changes to the nominal angles, the single joint stiffnesses k_i had to be doubled, i.e., $k_i^{\mathrm{sqt}} = 2 \cdot k_i$, for the left and right Hp, Kn and An FE joints. The remaining parameters are exactly the same as for the first simulation study from Sec. 6.1.1 and are listed in Tab. 6.1. The results for these parameter manipulations, including joint-angle trajectories and an impression on the state-dependent redundancy solution by the angle-length Jacobian $J^{\lambda\theta}$ is given in Fig. 6.3. The joint angle trajectories mostly follow their linear set values from (6.4, 6.3, 6.2) and stable stance is maintained throughout the downwards movement, the squat stand and the stand-up phase for both, the slow and the fast squat. Additionally, in Fig. 6.3, excerpts of the angle-length Jacobian $J^{\lambda\theta}$ (3.9,3.16) are shown. As expected, the angle-length Jacobian is similar in all three states as it captures the kinesiological effect of the MTUs, which is a morphological feature and does not change. However, in the squat-position ($t = 17.5$ s), Kn Ex and Fx entries for the Kn joint have higher magnitudes, which indicates a higher sensitivity of changes in the Kn MTUs' CE lengths w.r.t. Kn joint angle changes.

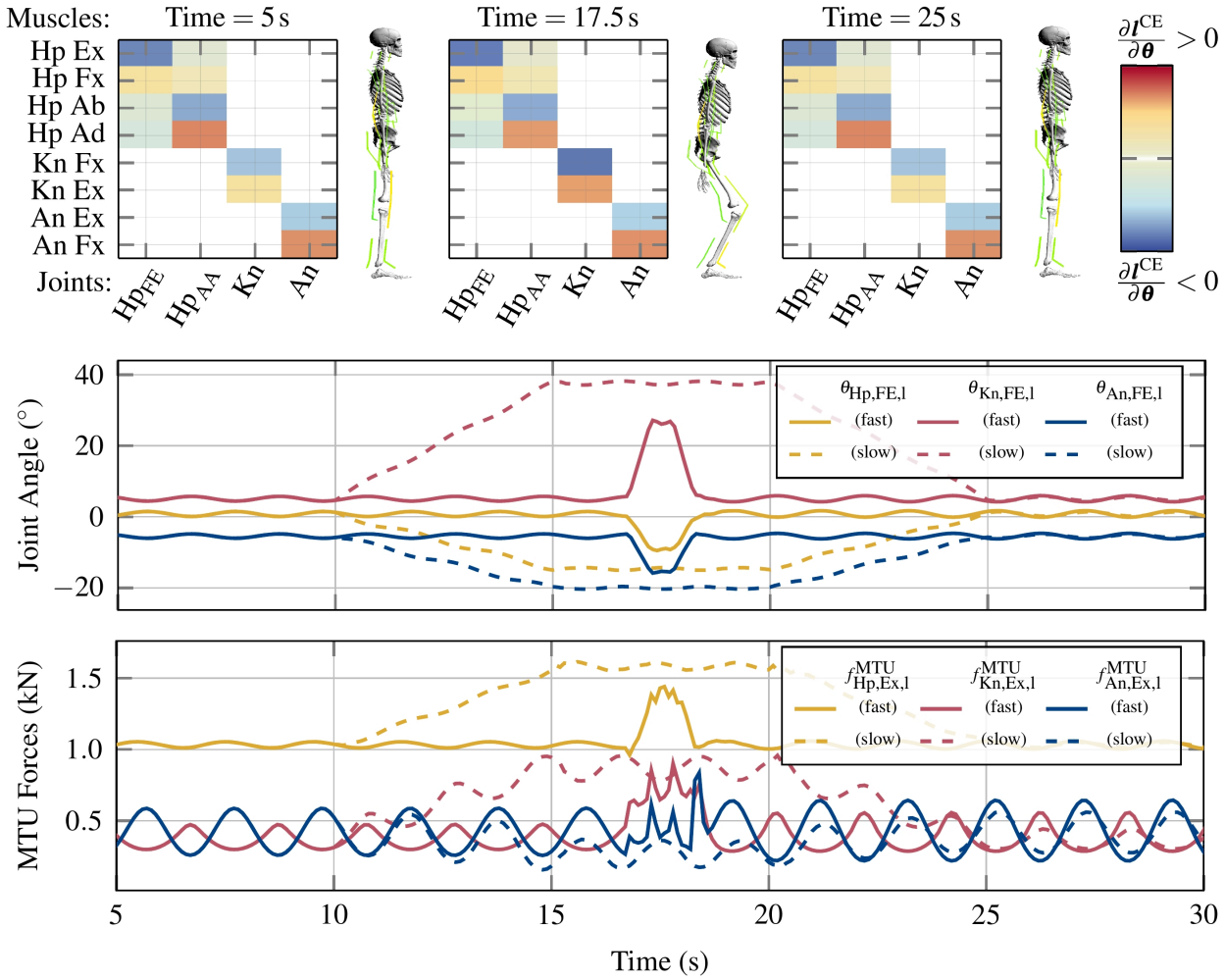


FIGURE 6.3: Simulation results of the fast ($\Delta t = 0.5$ s, solid lines) and slow ($\Delta t = 5$ s, dashed lines) squat movements according to the simulation task formulated in (6.2-6.4). The joint angles (upper line-plot) mostly follow their nominal trajectories (6.2-6.4) to execute the squat. The MTU forces of the Hp Fx and Kn Fx MTUs increase during the squat while the An Fx MTU force slightly decreases. This may be explained by the facts that the Hp and Kn joints swing further away from their unstable upright equilibria and the overall distance in z-direction of the body's COM to the An joint decreases. In the box-plots, excerpts of the angle-length Jacobian matrix (3.16) for the left leg for the time instances $t = 5$ s, $t = 17.5$ s and $t = 25$ s are displayed as heat-maps of MTU contribution to Fx and Ex movements.

7.1 Controlling the positions of lower and upper extremities

In this section, the capability of the control architecture to control the position in 3D space of a limb of the model is demonstrated. Therefore two scenarios are prepared in the simulation framework `demoa`. In the first scenario of *static* position control, the limbs of the right hand and the left foot are controlled to match a fixed position in space. The second scenario considers the *moving-case* to demonstrate the control behaviour for following a trajectory in space. The pelvis body of the DHM is fixed to the world at a height of 1 m and will not drop down due to the gravitational acceleration of $g = -9.8065 \frac{\mathrm{m}}{\mathrm{s}^2}$. The control behaviours for the right hand and the left foot of the static-case are presented in Sec. 7.1.1 and of the moving-case in Sec. 7.1.2. Details on the simulation software and the used solver are presented in Appendix D.1.

For position control, as described in Sec. 3.3.3, the frame that is controlled and a reference frame must be specified on the model. These two frames specify a kinematic sub-chain that is responsible for the movement. The frames are described by homogeneous 4×4 matrices, as known from (2.15). For controlling the hand, the frame $\mathcal{H}_C$ that is controlled is specified approximately at the palmar interphalangeal fold of the index finger. It is specified in (7.1) relative to the COM frame $\mathcal{H}$ of the hand. The reference frame $\mathcal{H}_R$ is attached to the Sh joint $\mathcal{H}_R = \mathcal{J}_{\mathrm{Sh,Spine}}$. The position controller of the hand includes movements of Sh FE, Sh AA and Eb FE. For controlling the foot, the control frame $\mathcal{F}_c$ is approximately placed at the proximal interphalangeal joint of the second toe:

$$\boldsymbol{G}^{\mathcal{H}\mathcal{H}_c} = \left[\begin{array}{ccc|c} 1 & 0 & 0 & 0\,\mathrm{m} \\ 0 & 1 & 0 & 0\,\mathrm{m} \\ 0 & 0 & 1 & -0.075\,\mathrm{m} \\ \hline 0 & 0 & 0 & 1 \end{array}\right] \qquad \boldsymbol{G}^{\mathcal{F}\mathcal{F}_c} = \left[\begin{array}{ccc|c} 1 & 0 & 0 & 0.1\,\mathrm{m} \\ 0 & 1 & 0 & 0\,\mathrm{m} \\ 0 & 0 & 1 & 0\,\mathrm{m} \\ \hline 0 & 0 & 0 & 1 \end{array}\right]. \tag{7.1}$$

The reference frame of the foot $\mathcal{F}_R$ is attached to the Hp, $\mathcal{F}_c = \mathcal{J}_{\mathrm{Hp,Pelvis}}$. The control of the foot involves Hp FE, Hp AA, Kn FE and An FE. For each controller a desired position is set in the form of another frame in the world or attached to a body. Based on the error (3.40) between the current and desired frames in the conceptional layer a postural plan $\boldsymbol{\theta}_{\chi}^{\mathrm{des}}(t)$ is derived by the position-joint Jacobian matrix (3.43), as described is Sec. 3.3.3. Following the transformational layer (Sec. 3.2) and the structural layer (Sec. 3.1), MTU stimulations (3.6) are eventually generated that strive to minimise the control error $\boldsymbol{\chi}^{\mathrm{limb}} \rightarrow \boldsymbol{\chi}^{\mathrm{des}}$. The additional inputs of co-contraction ζ^u and ζ^θ, and the control parameters are chosen to be the same for the static and the moving case scenarios and are listed in Tab. 7.1.

7.1.1 Static-case position control

For the simulation scenario of static-case position control of the hand and the foot, desired positions in the form of coordinate frames must be provided to the conceptional layer. The position of a frame relative to the inertial world frame $\mathcal{W}$ is described by homogeneous 4×4 matrices, as known from (2.13). During the first 0.1 s of the simulation, the desired positions $\boldsymbol{G}^{\mathcal{W}\mathcal{H}_d}$ and $\boldsymbol{G}^{\mathcal{W}\mathcal{F}_d}$ are set exactly to the controller frames coordinates (7.1), i.e. $\boldsymbol{G}^{\mathcal{W}\mathcal{H}_d} = G^{\mathcal{W}\mathcal{H}}\boldsymbol{G}^{\mathcal{H}\mathcal{H}_c}$ and $\boldsymbol{G}^{\mathcal{W}\mathcal{F}_d} = G^{\mathcal{W}\mathcal{F}}\boldsymbol{G}^{\mathcal{F}\mathcal{F}_c}$. By this, the control architecture does not produce any task fulfilling stimulation contributions (3.6), as $\boldsymbol{G}^{\mathcal{D}\mathcal{L}} = \boldsymbol{I_4}$ and thereby $\boldsymbol{\chi}^{\text{err}} = \mathbf{0}_6$. Additionally, during this time interval, the reference co-contraction for all muscles involved are set to $\zeta^u = 0$ and the joint co-contraction as well $\zeta^\theta = 0$. This effectively disables any control. After $t = 0.1\,\text{s}$, the desired frames are set to

$$\boldsymbol{G}^{\mathcal{W}\mathcal{H}_d} = \left[\begin{array}{ccc|c} 1 & 0 & 0 & 0.4\,\text{m} \\ 0 & 1 & 0 & 0.3\,\text{m} \\ 0 & 0 & 1 & 1.2\,\text{m} \\ \hline 0 & 0 & 0 & 1 \end{array}\right] \qquad \boldsymbol{G}^{\mathcal{W}\mathcal{F}_d} = \left[\begin{array}{ccc|c} 1 & 0 & 0 & 0.4\,\text{m} \\ 0 & 1 & 0 & -0.2\,\text{m} \\ 0 & 0 & 1 & 0.25\,\text{m} \\ \hline 0 & 0 & 0 & 1 \end{array}\right]. \tag{7.2}$$

The simulation for the position controllers of the hand and of the foot are performed in separate simulations. By only setting the desired position (7.2) at the time $t = 0.1$s for one of both, the control architecture is sufficiently set up. The generation of MTU stimulations exactly follows the description in Part II of this dissertation.

The simulation results for the hand controller are shown in Fig. 7.1 and those of the foot controller in Fig. 7.2. For each controller, a plot of the trajectory in 3D space is shown (Figs. 7.1, 7.2 a)), together with plots of the position control error in the conceptional layer, joint angle control error in the transformational layer, as well as plots of the current and desired CE lengths and MTU stimulations from the structural layer (Figs. 7.1, 7.2 d)). In the upper box plots (Figs. 7.1, 7.2 b)), snapshots of the position-joint Jacobian $J^{\theta\chi}$ are shown as heat-maps. This Jacobian (3.42) relates required changes in the positional space to changes in the joint space. In the lower box plots (Figs. 7.1, 7.2 c)), snapshots of the angle-length Jacobian $J^{\lambda\theta}$ are shown. The angle-length Jacobian (3.9) resolves the muscle-joint redundancy and translates an error in joint angles to an error in CE lengths.

After some acceptable overshooting transition, both, the hand and the foot, asymptotically approach their desired set points as seen in Figs. 7.1 and 7.2 a) and d). The rotational degrees of freedom in 3D space are hereby not controlled, as the PID matrix entries for these DoFs are set to zero (see also Tab. 7.1). Due to the different morphology, e.g. muscle parameters, segment lengths and weights, and the different controller parametrisation (see Tab. 7.1) the control behaviours for the hand and the foot are different. While the foot shows a more smooth movement that takes longer to reach the asymptotic state, the hand approaches its goal faster. This is also reflected in the respective control signals in the Figs. 7.1 d) and 7.2 d).

The size of the stable positional region of attraction is restricted due to the limited DoFs of the DHM. Most prominently by the missing capability of circumduction in the Hp and Sh joints. To analyse the control performance within its stable range, in the following section a movement of the desired frames is included.

7.1.2 Moving-case position control

For the moving-case scenario of the positional control of the hand and the foot, the desired triads are smoothly moved in a circular shape. This is achieved, by firstly defining an auxiliary body with the three open transversal DoFs of the Cartesian coordinate axes of the inertial *world* frame. After the time $t_0 = 0.1$s the velocity of these DoFs are then directly manipulated to follow a circular shape. By defining the desired frames to be attached to the auxiliary body, the respective position of the hand or the foot is asked to follow the circular

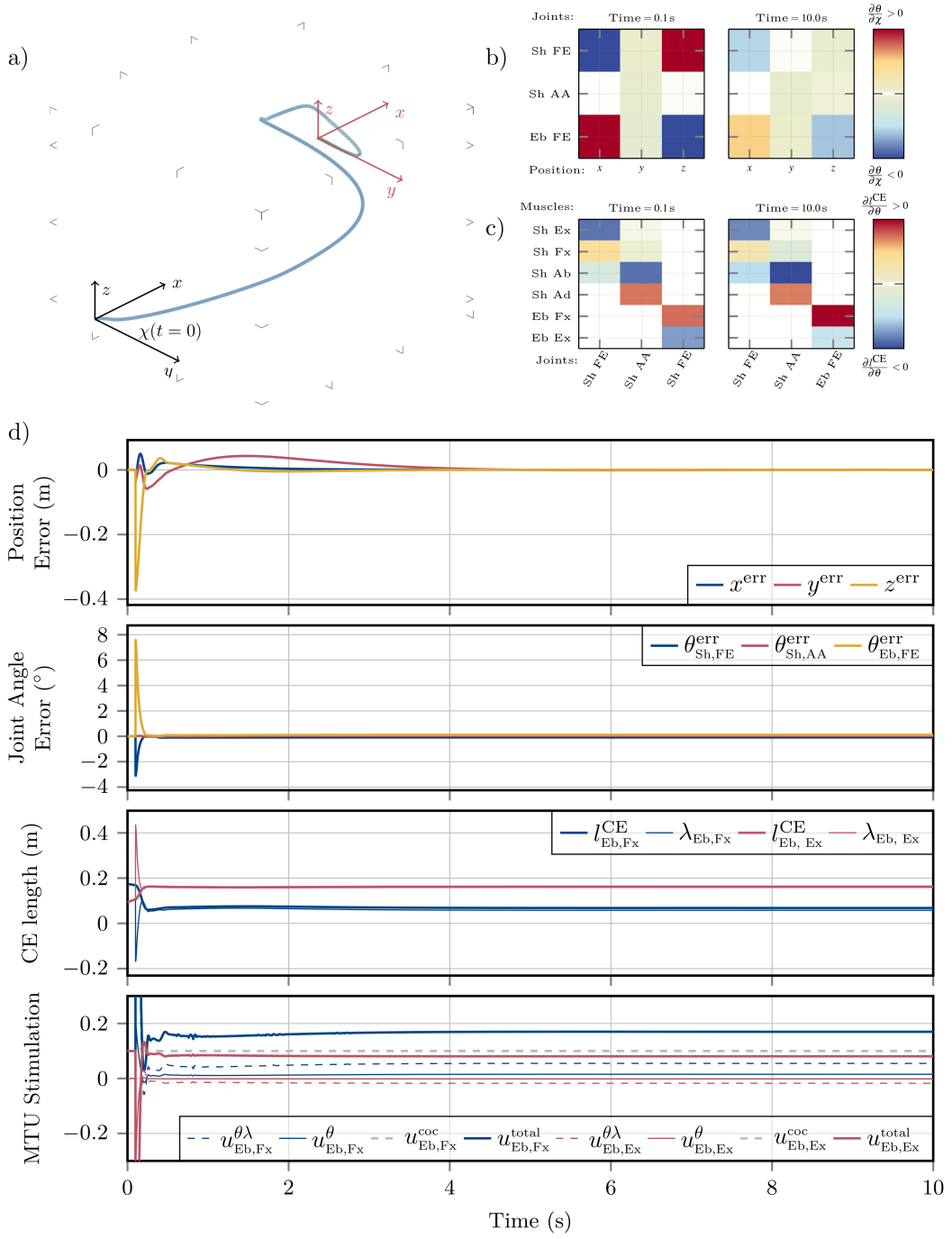


FIGURE 7.1: Simulation results of static control of the hand. In a) the 3D trajectory of the hand over the whole movement is shown. In b) and c) box-plots of the Jacobian matrices $J^{\theta\chi}$ and $J^{\lambda\theta}$ are shown. In the line-plots in d), the time trajectories of the hand's position, its joint angles and Kn MTUs CE-lengths are shown together with their desired values. In the line plot at the bottom, additionally the stimulations of the Kn MTUs are displayed. It is clearly visible that the control errors are very quickly reduced and then slowly but asymptotically approach the desired position. The deviation in y-direction thereby takes the longest to reach.

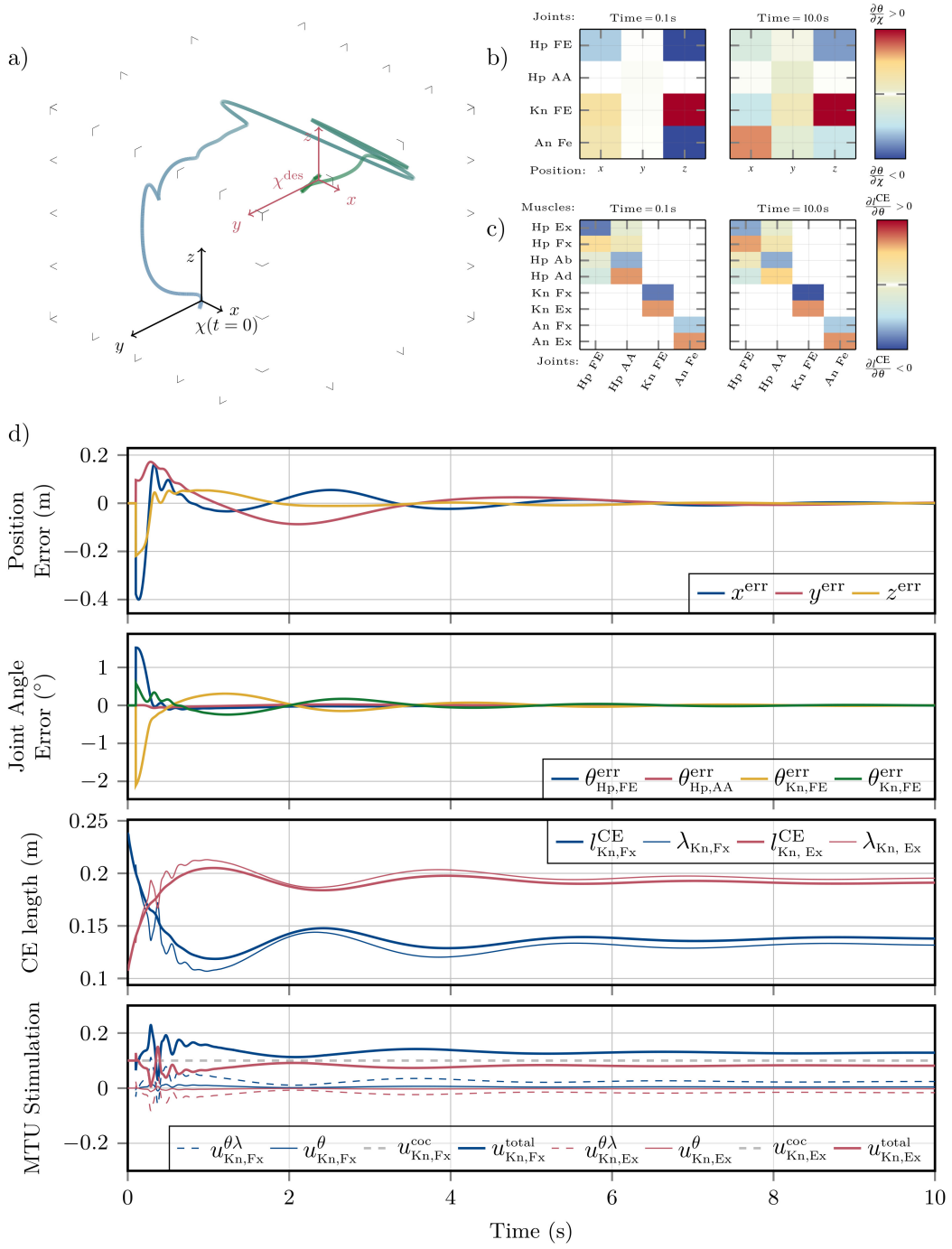


Figure 7.2: Simulation results of static control of the foot. After some acceptable overshooting, the desired positions and joint angles are already closely reached and further asymptotically approached. Even though this strongly hints towards asymptotic stability of the conceptional and transformational controllers, some control deviation in the structural layer remains. From this remaining control deviation of the Kn Ex MTU a stimulation contribution $\boldsymbol{u}^{\lambda}$ corresponds that acts against gravity.

shape. For the auxiliary bodies B_H and B_F that define the desired positions $\mathcal{H}_d$ and $\mathcal{F}_d$ of the left hand and of the right foot, respectively, the following translational velocities according to (2.23) are set with respect to the world frame $\mathcal{W}$:

$$\boldsymbol{\nu}^{\mathcal{W}}_{\mathcal{W}\mathcal{B}_H}(t) = \begin{bmatrix} a \cdot \sin(f \cdot (t - t_0)) \\ 0 \\ a \cdot \cos(f \cdot (t - t_0)) \end{bmatrix} \qquad \boldsymbol{\nu}^{\mathcal{W}}_{\mathcal{W}\mathcal{B}_F}(t) = \begin{bmatrix} a \cdot \sin(f \cdot (t - t_0)) \\ a \cdot \cos(f \cdot (t - t_0)) \\ 0 \end{bmatrix}, \tag{7.3}$$

with the parameters of an amplitude of $a = 0.03\,\mathrm{m}$ and a frequency of $f = \frac{1}{3}\frac{\mathrm{rad}}{\mathrm{s}}$.[1] The control parameters are set exactly the same as for the static-case scenario and are listed in the Tab. 7.1.

The simulation results of the moving-case scenario are shown in Fig. 7.3 for the hand controller and those of the foot controller in Fig. 7.4: For each controller, in a), a plot of the trajectory in 3D space is shown together with the moving desired position (red dashed line in a)). In the upper box plots in b), snapshots of the position-joint Jacobian $J^{\theta\chi}$ are shown as heat-maps. This Jacobian (3.42) relates required changes in the positional space to changes in the joint space. In the box plots in c), snapshots of the angle-length Jacobian $J^{\lambda\theta}$ are shown. The angle-length Jacobian (3.9) resolves the muscle-joint redundancy and translates an error in joint angles to an error in CE lengths. In the line plots in d), the positions relative to the inertial world frame are shown together with plots of the joint angles, current and desired CE lengths and MTU stimulations. The overall superhuman behaviour of static-case and moving-case position control hints towards a model description that lacks some important biophysical properties, e.g. neural latency times or signal noise.

[1] Directly setting the velocity of a body violates the laws of physics and can be considered as a minor hack in the simulation environment `demoa`. As a consequence, in the velocity component in z-direction remains some acceleration due to gravity and the auxiliary body slowly falls down creating, e.g the spiral shaped movement in Fig. 7.4.

Table 7.1: Control parameters of the position controller of the left hand and the right foot. The PID parameters of the rotational DoFs are set to zero. This disables the respective stimulation contributions from the controller for the rotational DoFs. Both position controllers are acting as PI-controllers as their D parameters are set to zero. For the hand controller, the θ_λ and the θ controllers are parametrised as PD-controllers. For the foot controller only the θ_λ controller is used in the transformational layer. All delays $\boldsymbol{\delta}$ are set to zero.

Conceptional Layer				Transformational Layer							Structural Layer		
Position Controller `Hand_l`													
Position				Joint angle							MTU		
	P_χ	I_χ	D_χ	Name	$P_{\theta\lambda}$	$I_{\theta\lambda}$	$D_{\theta\lambda}$	P_θ	I_θ	D_θ	Name	κ	$u^{\mathrm{coc}}_{\mathrm{ref}}$
x	0.2	0.2	0	shoulder flexion-extension	1.0	0	0.001	1.0	0	0.001	Sh Fx	1.0	0.1
y	−0.2	−0.2	0								Sh Ex	1.0	0.1
z	0.2	0.2	0	shoulder abduction-adduction	1.0	0	0.001	1.0	0	0.001	Sh Ab	1.0	0.1
rx	0	0	0								Sh Ad	1.0	0.1
ry	0	0	0	elbow flexion-extension	1.0	0	0.001	1.0	0	0.001	Eb Fx	1.0	0.1
rz	0	0	0								Eb Ex	1.0	0.1
Position Controller `Foot_r`													
Position				Joint angle							MTU		
	P_χ	I_χ	D_χ	Name	$P_{\theta\lambda}$	$I_{\theta\lambda}$	$D_{\theta\lambda}$	P_θ	I_θ	D_θ	Name	κ	$u^{\mathrm{coc}}_{\mathrm{ref}}$
x	0.015	0.01	0	hip flexion-extension	1.5	1	0.0035	0	0	0	Hp Fx	1.0	0.1
y	0.015	0.01	0								Hp Ex	1.0	0.1
z	0.015	0.01	0	hip abduction-adduction	1.5	1	0.0035	0	0	0	Hp Ab	1.0	0.1
rx	0	0	0								Hp Ad	1.0	0.1
ry	0	0	0	knee flexion-extension	1.5	1	0.0035	0	0	0	Kn Fx	1.0	0.1
rz	0	0	0								Kn Ex	1.0	0.1
				ankle flexion-extension	1.5	1	0.0035	0	0	0	An Fx	1.0	0.1
											An Ex	1.0	0.1

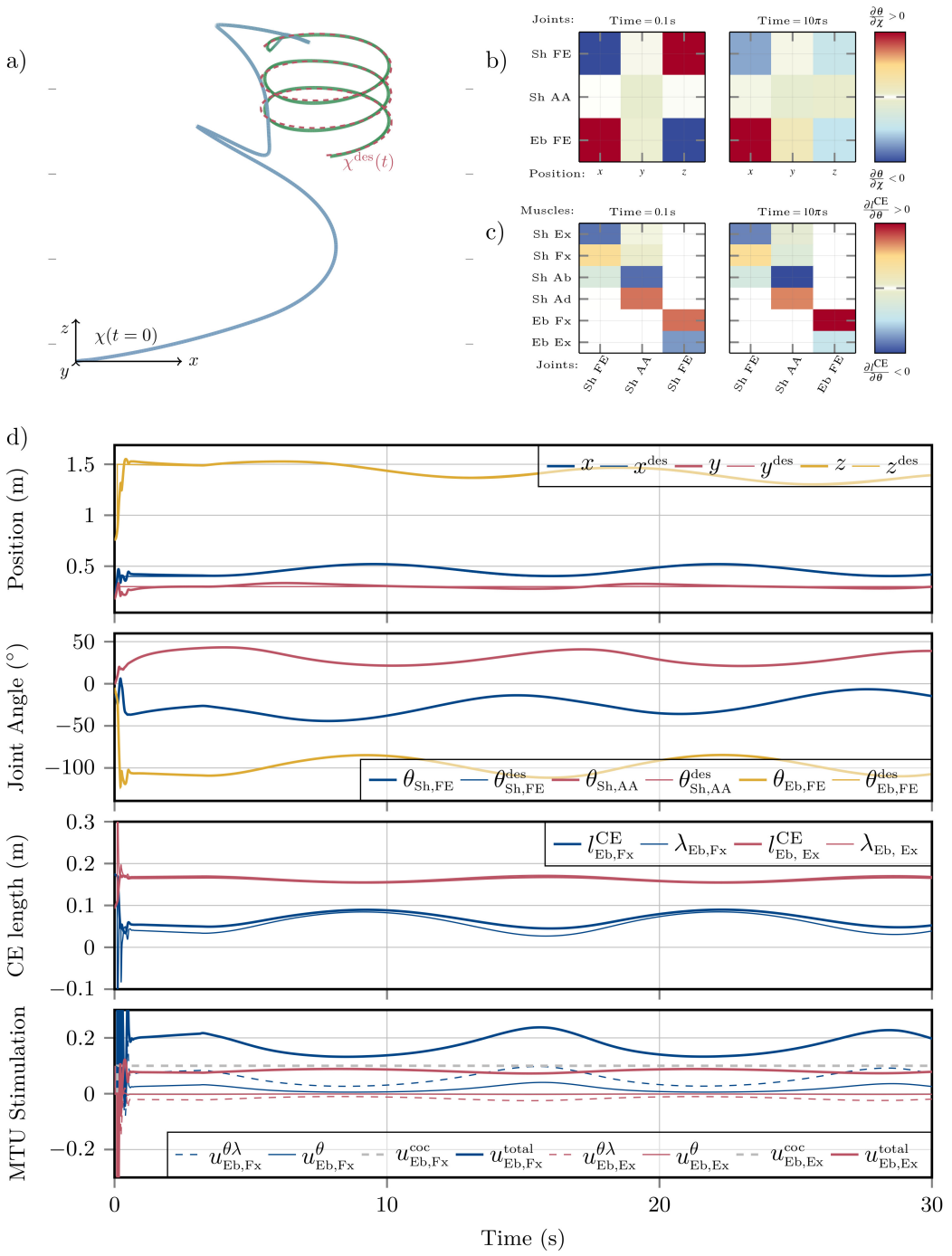


Figure 7.3: Simulation results of moving control of the hand: a) 3D trajectories of the hand (blue→green for $t \uparrow$) and the desired position (red dashed) in the x-z plane. In b) and c) snapshots of the Jacobian matrices $J^{\theta\chi}$ (b) and $J^{\lambda\theta}$ (c) are shown. The colour coding in c) represents how a joint angle influences the position of the hand. Note, that the Jacobian entry for the Sh FE joint and z-direction changes its sign dependent on the state. In d) the absolute position of the hand, the joint angles of the arm and CE-lengths and MTU stimulations for the Eb MTUs are shown. After some short overshooting, the hand approaches the desired position and stays close to it throughout the simulation.

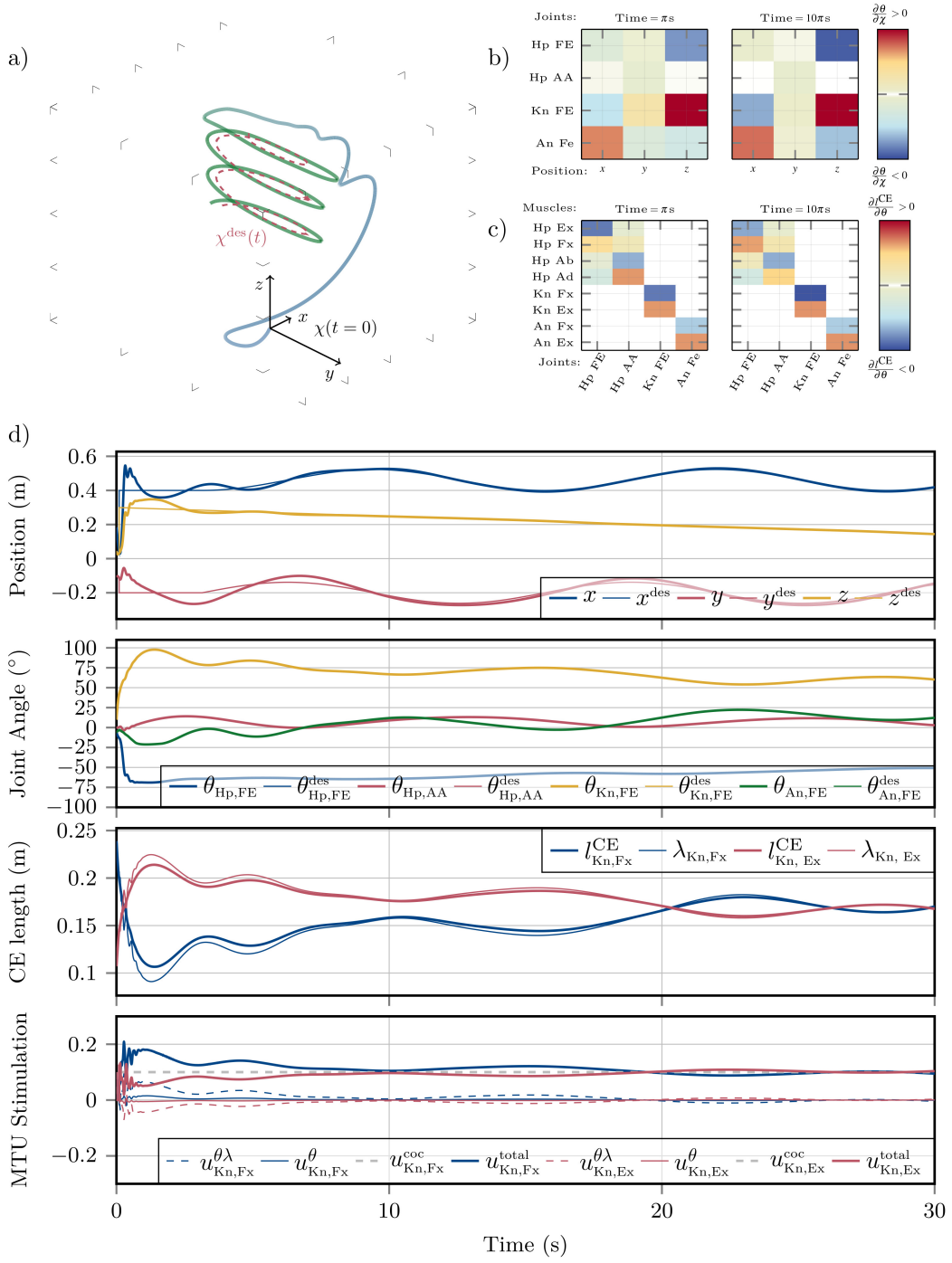


FIGURE 7.4: Simulation results of moving control of the foot: a) 3D trajectories of the foot (blue→green for $t \uparrow$) and the desired position (red dashed) in an isometric projection. In b) and c) snapshots of the Jacobian matrices $J^{\theta\chi}$ (b) and $J^{\lambda\theta}$ (c) are shown. The colour coding in c) represents how a joint angle influences the position of the hand. In d) the absolute position of the hand, the joint angles of the arm and CE-lengths and MTU stimulations for the Eb MTUs are shown. After some acceptable overshooting, the foot approaches the desired position and stays close to it throughout the simulation. Note that the desired z-position is not constant as demanded in 7.3. This is due to an unintended use of the simulation software that technically violates the laws of physics. Still, the motion generated by the control architecture shows an asymptotic stable behaviour that spirals downwards to follow $\boldsymbol{\chi}^{\text{des}}(t)$.

Generalisation towards complex combined movements 8

In the previous chapters of this part, it was demonstrated that the control architecture is capable of generating coordinated movements in an isolated setup and for elementary controls of joint angles and positions. In this chapter, these findings are used for the simulation of more complex and combined movements towards a generalisation to arbitrary movements. This is demonstrated for the two application cases of a human car ingress motion in Sec. 8.1 and of a forensic case analysis in Sec. 8.2. For both application cases a combination of angle, torque and position control is used.

In the first application example of a synthesised human car ingress motion, the control architecture is configured in such a way that the model's kinematics perform an ingress motion into a car without the violation of boundary restrictions, such as collision avoidance with the head and the door frame. The initial position of the model is a free stable stance and the subsequent movement is divided into the movement primitives of grabbing sectionthe steering wheel, lifting the leg above the sill and placing it into the footwell, diving the head below the door frame and a placement of the buttock on the seat in the car. The initial free standing beneath the car is realised exactly as described in Sec. 6.1.1 and the execution of the movement primitives by a combination of position and angle control of the limbs as described in the Secs. 5 7 . The simulations are validated with the kinematic results of an experimental study. This makes the method viable to predict changes in the DHM's kinematics due to a virtual change of the car's design, i.e. the door frame height. Some of the content of this application example have already been accepted for publication by Walter et al. (2021b).

In the second application example of a forensic case analysis, it is investigated whether it is possible that the found body pose of a corpse in a bathtub is the outcome of a slip and fall into the tub or if another person must have been involved as a culprit. It is, of course, not possible to fully unveil the truth of the past, but at least this approach forms a novel method to investigate the range of possibilities of human kinematics in an only partly known environment. To realise the movement, initially a free standing (according to Sec. 6.1.1) is performed, followed by an angle controlled forward bending and a position controlled grasping movement towards the tap. Additionally, to model a *transient ischemic attack* (TIA), at a certain time instance, all controllers are switched off and therefore the MTU stimulations are set to zero. By variations of the initial conditions, i.e. the standing position relative to the bath tub, a probability can be estimated, whether the final position of the corpse can be met by accident. The outcomes of this application example are in preparation for a manuscript. Here, only the method for the controller setups are presented that lead to the movement of a fall.

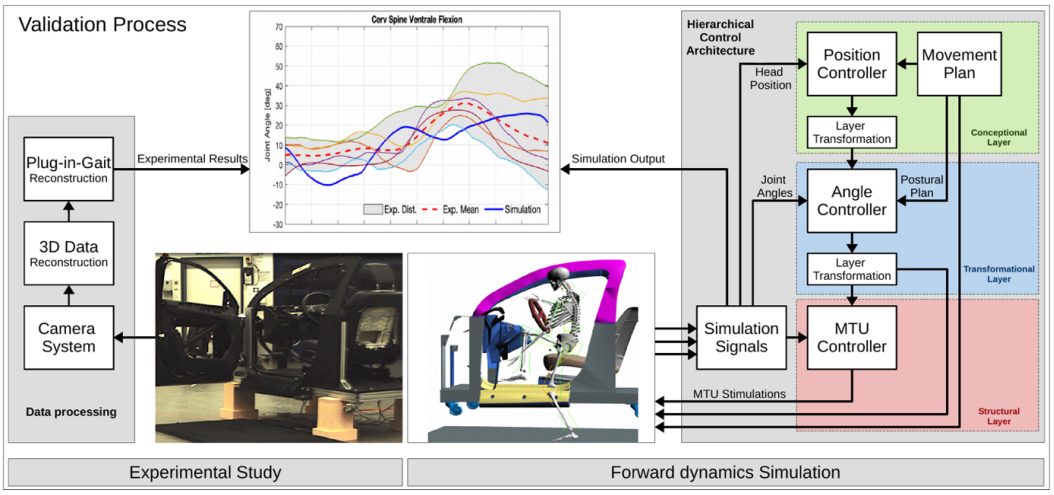


Figure 8.1: Overview of the validation process for the car ingress motion. Separated from each other simulations and experiments are performed to obtain kinematic data of an car ingress motion. The movement plan that is used in the conceptional layer of the control architecture is heuristically learned such that the model does not violate any boundary restrictions and that the synthesised kinematic data has a plausible agreement to the experimental data.

8.1 Application example digital engineering: Car ingress ergonomics

In this application example a human car ingress motion is synthesised. The DHM that is used in this example is exactly the same as described in Sec. 4 and in Appendix C. The simulation model of the car is based on the geometry of a real existing car model of an 'SUV' type (see Fig. 8.1). The ingress motion is achieved by dividing the movement task into single movement primitives in different conceptional spaces and configuring the control architecture to execute those. The movement task is then heuristically learned to not violate any of the boundary restrictions of: i) collision avoidance (right foot and sill), ii) grabbing position of the hand at the steering wheel, iii) compliance with a minimum of head clearance (head and door frame), iv) placement of the right foot on the footwell in the car, v) placement of the buttock on the seat in the car and vi) keeping a stable position. Additionally the simulation results are validated on experimental data from a separated parameter study of real humans getting into the car Walter et al. (2021b). An overview of the validation process is displayed in Fig. 8.1.

In a next step, the simulation model of the car was changed to mimic a pre-production design change of a higher door frame. This changes the definition of the boundary restrictions and by re-learning the movement in terms of adjustments of the movement task, valid ingress motions can be found for the new design.

8.1.1 Controller configurations for synthesising a car-ingress motion

To synthesise the car-ingress motion, a combination of angle, position and torque control is used. Initially the model freely stands besides the car, exactly as described in Sec. 6. The actual ingress motion is divided into movement primitives, which are separately controlled either by angle or position control of the limbs. The single movement primitives closely correspond to the boundary restrictions from above and are: i) grabbing the steering wheel with the right hand, ii) lifting the right foot above the sill, iii) placing the foot on the

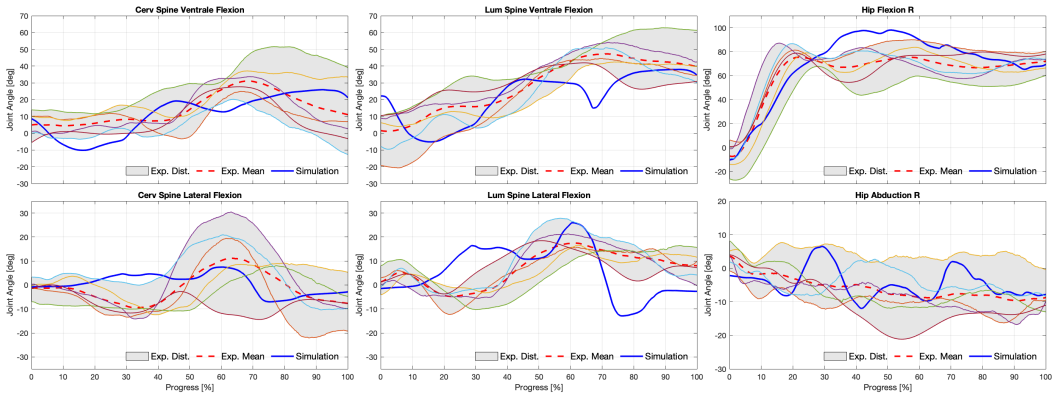


FIGURE 8.2: Validation of the ingress motion: The kinematic trajectories of the Ce, Lu and Hp joint angles of the synthesised movement (solid blue lines) are compared to experimental data (thin coloured lines) (Walter et al., 2021b). Even though the simulated motion does not always stay within the corridor that is spanned by the motion data obtained from the experimental measurements, it is considered valid here. Leaving the corridor is tolerable, as the synthetic motion is a successful individual motion by its own, i.e. the boundary restrictions were not violated.

footwell in the car, vi) pulling the body inside the car, v) diving the head below the door frame, v) placement of the buttock on the seat in the car and vi) keeping a stable position. To execute these movement primitives a total of four angle controllers, each for the left and right upper and lower extremities, one position controller for the head and one torque controller for upright standing are implemented. The control parameters for the joint and torque controllers are exactly the same as those in the Chpts. 5 and 6. The head position controller's transformational and structural layer parameter are the same as those in Sec. 7 for the foot controller and its conceptional parameters are $P_\chi = \text{diag}(2.0) \in \mathrm{R}^{4\times4}$, $I_\chi = \text{diag}(0.0) \in \mathrm{R}^{4\times4}$ and $D_\chi = \text{diag}(0.1) \in \mathrm{R}^{4\times4}$. A complete overview of the desired values that are set to the respective conceptional layers of the controllers is displayed in the Gantt-chart in Fig. 8.4 at the end of this section.

8.1.2 Simulation results of the car ingress

With the controllers set-up as described above, the simulations are ready to be performed. As a simulation result, the kinematic data in the form of joint angle trajectories of a valid ingress motion are displayed in Fig. 8.2 together with a corridor that is spanned by the kinematics data from the experimental measurements from Walter et al. (2021b). The whole motion was thereby normalised over time and scaled to a progress parameter $P \in [0\%..., 100\%]$. For details on the experimental setup and the recordings, please refer to Walter et al. (2021b). The simulation results have an overall agreement in shape and amplitudes and, even though they do not always stay within the experimental data's corridor, show the outcome of a valid ingress motion. This is an impressing result, especially when it is kept in mind that the Hp joint is missing its physiological capability of rotation. To further improve the validity of a simulation of this kind, the biofidelity of the model itself must be considered to be increased.

After some small modifications to the movement plan (see Fig. 8.4), i.e. a small time shift of 0.1s and a 5cm higher placement of the desired position of the head position controller, a second simulation was performed for the car model with the higher door frame. This is a very small change within the movement plan and showcases not only the sensibility of such an asynchronous and complex movement but also makes one humble on the sophis-

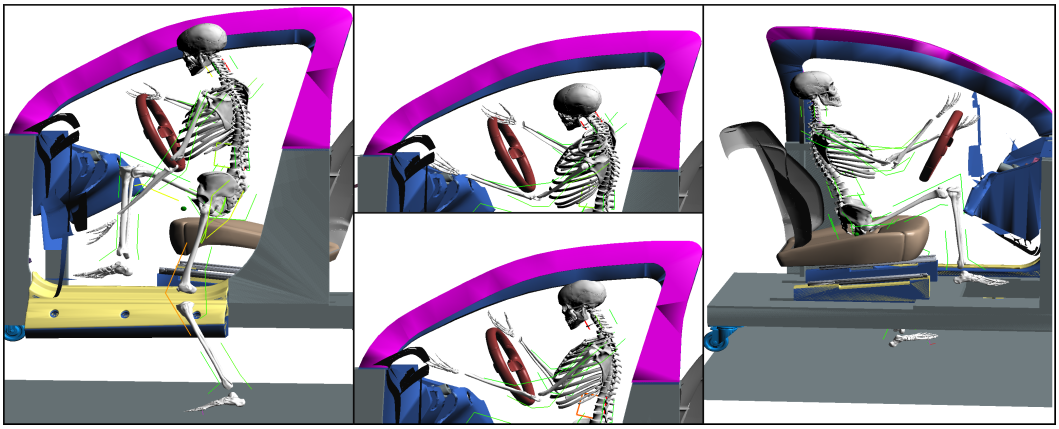


Figure 8.3: Phases and movement primitives of the simulated car ingress motions. First, the right leg is lifted above the sill and the right arm grabs the steering wheel (left picture $P = 34\%$). In the middle picture ($P = 65\%$) the transition of the head through the door is shown as a comparison of the two door frame heights. The movement plan was adapted in the control architecture according to available headspace. In the right picture ($P = 100\%$) the final position in the car is shown.

ticated solution nature found that lets a real human effortlessly slip in. The differences of both simulations are displayed in Fig. 8.3.

The simulation results of the second simulation can now serve some interesting purposes. As they are of purely synthetic nature, i.e. there are no experimental data that drove the simulation in any kind, the internal states of the model reveal a potential change of biophysical or ergonomic properties due to the virtual design change. This can help, e.g. in the early development process of a car, where no physical mock-up yet exists. For the here presented example of a higher door frame, the simulation results showed that significantly less work has to be done, especially in lateral and ventral flexion of the lumbar spine. For a detailed analysis of the results, please refer to Walter et al. (2021b).

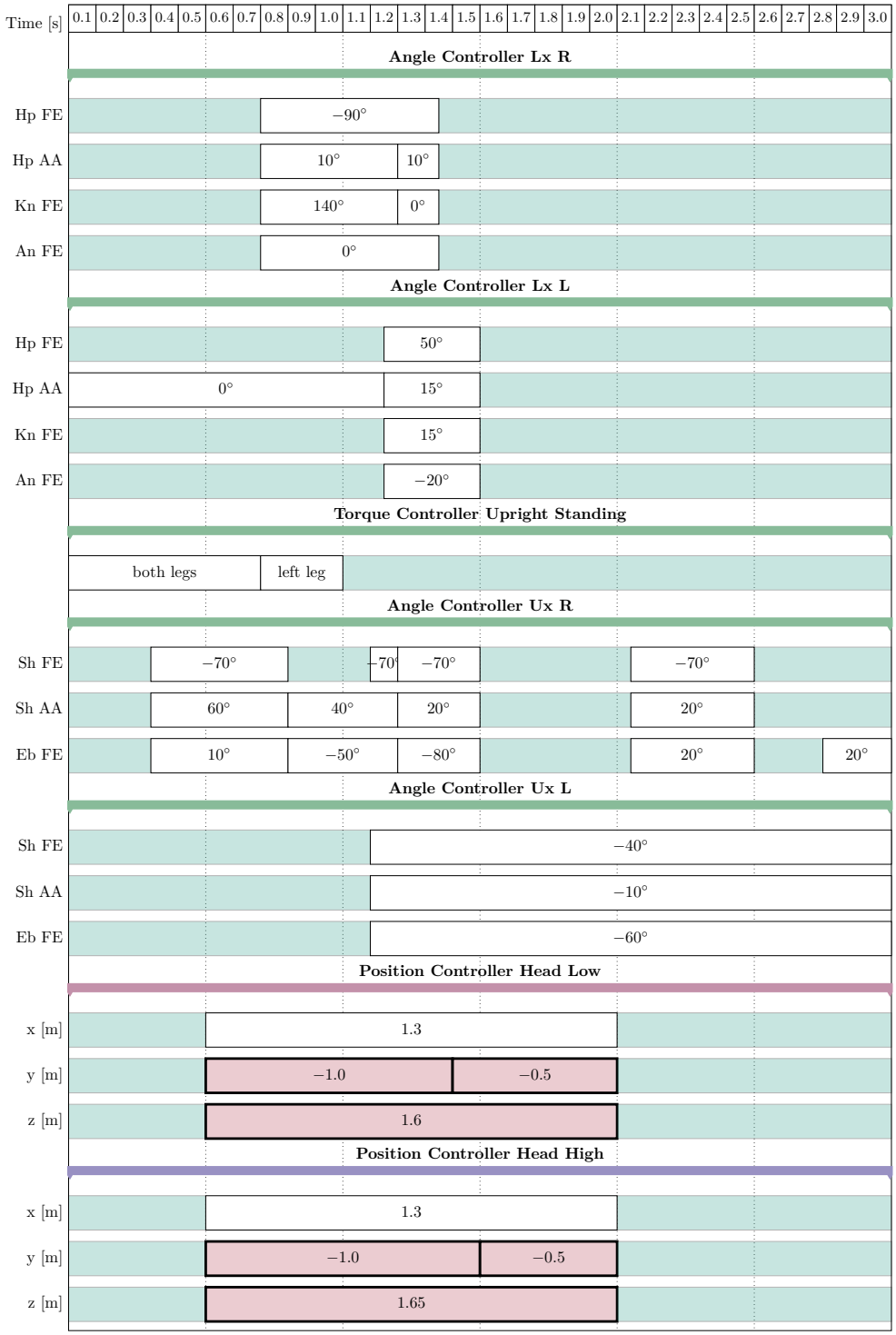


FIGURE 8.4: Gantt-chart of the movement plan for the ingress motion. For the displayed times, the desired values for the conceptional controllers are set. The opaque blue areas correspond to a disabling of the respective controller during these times. The reference co-contraction stimulation contribution is not shown and is either set to $u_{\text{ref}}^{\text{coc}} = 0.0$, $u_{\text{ref}}^{\text{coc}} = 0.1$ or $u_{\text{ref}}^{\text{coc}} = 0.3$. The only change in the movement plan to realise the 'higher' simulation is in the timing and desired position of the position controller of the head. This is highlighted in the red blocks in the respective position controller movement plans.

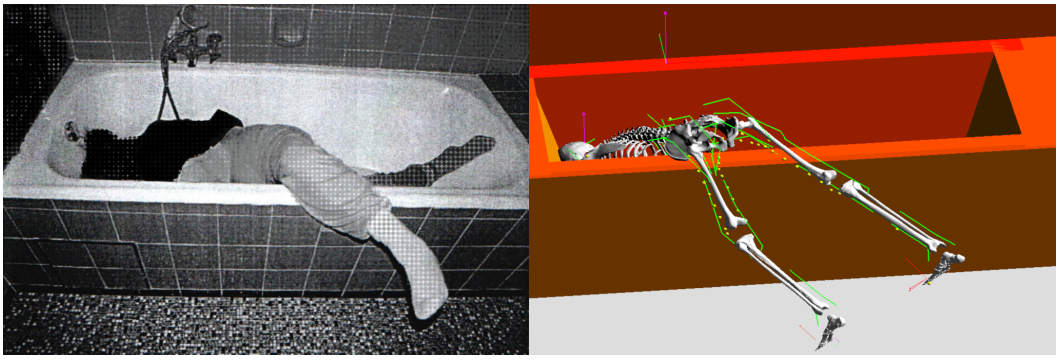


Figure 8.5: Application example forensic case analysis: The control architecture is used to synthesize the movement of a fall into a bathtub. By variations on the movement plan and the initial conditions a probability can potentially derived whether an unusual body pose (left picture) can be the outcome of a tragic fall. The simulation screenshot sown on the right does not exactly match the body posture on the left. It is imaginable thou that the right leg might slip into the tub when the trunk rotates, e.g. due to water in the tub.

8.2 Application example forensics: Mean crime or tragic fall

The control architecture presented in this thesis provides a powerful tool to the field of biomechanics as it allows to intuitively synthesize a very wide range of movements. One key factor here is that the architecture is self-driven, i.e. it does not rely on any kind of experimental data that drives the movement. This is due to the possibility it provides to formulate the movement task in different conceptional spaces heuristically. The MTU stimulation generation, which requires a solution to the model's redundancy, autonomously follows from the architecture's design, as described in Part II. This all now allows to synthesise a wide range of human movements, including those of uncertain circumstances, and to investigate the model's internal dynamics.

As a further application example, in this section, the architecture is used to synthesise the movement of an old woman's fall into a bathtub. This is an attempt towards helping to answer the question whether a specific body pose of a corpse is possibly the outcome of a tragic fall or it is more likely that another person as a culprit was involved[1]. In Fig. 8.5 the body pose that was found at the scene is shown together with the converged pose of a simulation. Among other things, the unusual body pose of the women raised suspiciousness towards a crime. The court was in doubt that the found body pose and the injuries at the head occurred by chance, although it was known that the old woman could have suffered a TIA and was prone to falling. A TIA can be associated with brain dysfunction in a circumscribed area caused by a regional reduction in blood flow, i.e. by ischemia (Coutts, 2017). Symptoms can include gait and balance dysfunction despite having no obvious physiological impairments (Batchelor et al., 2015).

To perform the movement of a fall into a bathtub, the control architecture was configured with four different controllers and a model of a TIA was included. The controllers are a torque controller for the legs that performs a stable standing and a squat motion, an angle controller that controls the pose of the trunk and the head and two position controllers, each for the movement of a hand. In the Sec. 8.2.1 the movement plan and the TIA model are presented and in Sec. 8.2.2 the simulation results are compared to some forensic findings.

[1]It is emphasised at this point that the here presented study is not meant to be used for drawing conclusions for any criminal case, but solely to demonstrate the general methods approach using the presented control architecture. Some aspects of the used model are not formulated sharp enough to allow straightforwardly drawing conclusions from the simulations results (see Sec. 8.2.4).

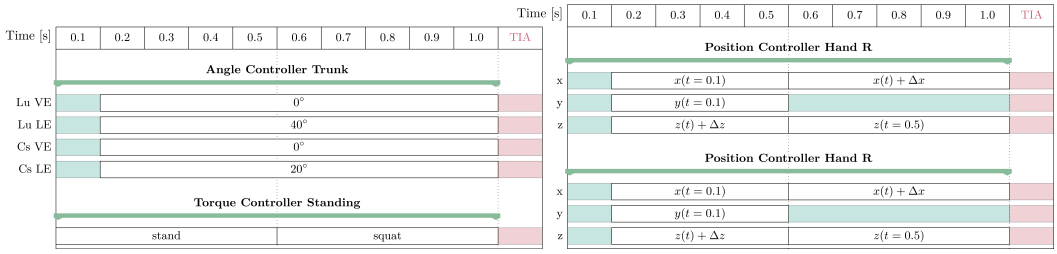


FIGURE 8.6: The Movement plan of the synthesised fall motion reflects the everyday task of preparing to soak laundry in the bathtub. Initially a stable standing is performed by torque control of the legs, the trunk is bend forwards and the hands are lifted. Then a squat is initiated and the hands are brought front for grasping the handles. At the time $t^{\mathrm{TIA}} = 1\,\mathrm{s}$ a TIA disturbs the movement plan with the consequence of a fall into the bathtub. The position control is hereby generated only relative to the model itself and does not use information from the environment. This makes the movement plan a good candidate for variations in the initial condition of the simulation, as the movement will always be executed the same until a contact occurs.

8.2.1 Movement plan and TIA model

The movement plan for the fall movement is based on a normal all-day task that the old woman may have undertaken: She was known to soak her laundry in the bathtub before putting it into the washing machine. The simulation model's movement thus is aimed to mimic the movement of squatting and bending over the tub and reaching for the tap handles. This is achieved by the use of four controllers in parallel: i) a torque controller that acts on the legs to perform stable standing and squatting (see Chpt. 6), ii) an angle controller to control the forward bending of the trunk (see Chpt. 5) and iii) two position controllers for the hands to grab the tap (see Chpt. 7).

The grasping movement is implemented as a 'model-internal' movement, i.e. it does not rely on any information of the model's extra-personal space. This is done by a special way in defining the postural plan. At a certain time $t_0^{\mathrm{Pos}} = 0.1\,\mathrm{s}$, the desired positions in x and y directions of the hand are stored and set as desired positions. With this, the controller asks to hold the hands at the same $x - y$ coordinates. At the same time the desired z direction is set to the current z position of the hand plus a very small offset of $\Delta z = 10^{-3}$. This has the effect that the hand and is always driven upwards in z direction:

$$\boldsymbol{\chi}_{\mathrm{L/R}}^{\mathrm{des}} = \begin{bmatrix} x_{\mathrm{L/R}}^{\mathrm{des}} \\ y_{\mathrm{L/R}}^{\mathrm{des}} \\ z_{\mathrm{L/R}}^{\mathrm{des}} \end{bmatrix} = \begin{bmatrix} x_{\mathrm{L/R}}(t = t_0) \\ y_{\mathrm{L/R}}(t = t_0) \\ z_{\mathrm{L/R}}(t) + \Delta z \end{bmatrix} \qquad \text{for } t \geq t_0^{\mathrm{Pos}}.$$

At a second time of $t_2^{\mathrm{Pos}} = 0.5\,\mathrm{s}$, a forward movement of the hand is engaged. For the left hand the desired x position is set to $x_{\mathrm{r}}^{\mathrm{des}} = x_r(t) + \Delta x$ with $\Delta x = 10^{-3}$, the y direction is released from control and the height is held constant $z_r^{\mathrm{des}} = z_r(t = t_2^{\mathrm{Pos}})$. The left hands desired x_r^{des} and z_r^{des} position are set to the respective current positions of the right hand, while the height is kept constant. This aims to bring the left hand to the right hands position.

To model the TIA, it is simply assumed that a sudden loss of balance control occurred with the consequence of an uncontrolled fall into the tub. For simplicity, at certain time $t^{\mathrm{TIA}} = 1.0\,\mathrm{s}$ all stimulation contributions of the controllers are switched off. The complete movement plan and the occurrence of the TIA are displayed in Fig. 8.6.

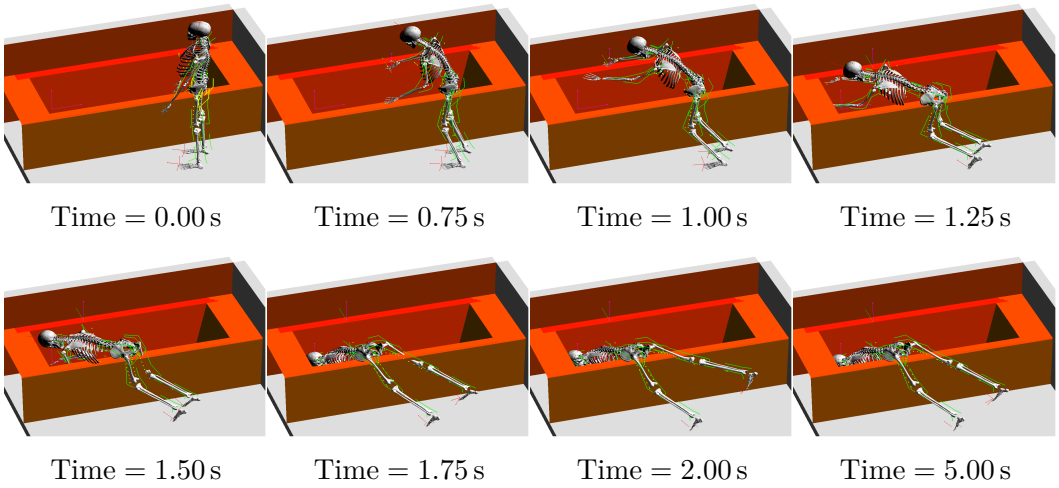


Figure 8.7: Time series of the fall motion. During the grasping, squatting and forward bending movement a modelled TIA strikes and causes the model to fall into the bathtub. The dynamics of the fall lead to an upward swing of the left leg at $t \approx 2\,\mathrm{s}$ that converges to the shown end position at time $t = 5\,\mathrm{s}$.

8.2.2 Bathtub model and contacts

The bathtub was modelled in accordance to the footage that was available from the scene (Fig. 8.5) and corresponds to a generic tub. The walls of the bathtub are defined as contacting surfaces that interact with a discrete set of specified contact points on the model (see yellow balls in Fig. 8.5 and Fig. 8.7). These contact points are implemented in arrays at the pelvis, thigh and thorax bodies, as well as specific points at the left and right arms and the head. These specific points correspond to areas where the forensics analysis identified injuries of the tissue. The contact forces follow the description from Sec. 2.4 and Appendix B. The contact parameters correspond to a hard and slippery surface.

8.2.3 Simulation results of the fall

The movement plan from the last section (Fig. 8.6) successfully synthesised a fall motion into the bathtub model. The movement mostly occurs as planned, as initially a stable stand is performed, then the hands are raised, the body is bend forward and a squat is initiated. After the hands grasp forward, the TIA strikes and the MTU stimulations are switched off. As a consequence, the model collapses into the tub. With the defined contact points the women's model interacts with the bathtub, first at the left forearm then with the head. The right leg surprisingly swings upwards after landing on the side of the tub, while the left leg remains closer to the ground. It is imaginable that a bit of rotation in the trunk can yield the unusual pose of the corpse from Fig. 8.5. A time series of the animation is displayed in Fig. 8.7.

As the kinematic validation, based on the overall visual impression of the fall movement, passed its first test, the simulation model's internal dynamics are worth being investigated. Therefore the forces of impact as well as their timing that occurred during the fall are analysed. Together with the end position of the body it is of great interest, whether the contacts with the bathtub can be the reason for injuries that where found at the autopsy. The forward dynamics approach that is presented here allows to measure the forces of the discrete specified contact points on the model. These forces correspond to the impact of the accelerated body parts with their respective masses. Relating these force with clinical

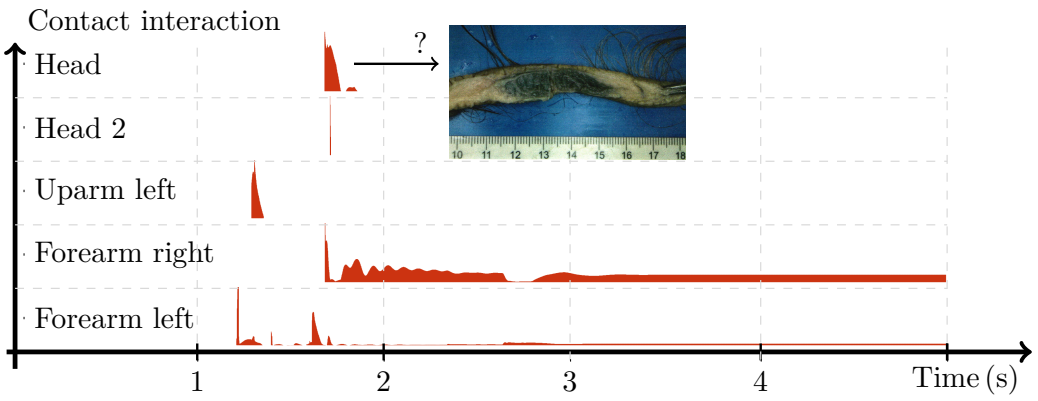


FIGURE 8.8: Contact interactions during the fall motion. At first the right forearm contacts the inner wall and slips down so that the up arm contacts the tub. As the head dives deeper into the tub the right forearm touches the side and the ground and the head hits the bath tub wall. The model then comes to a rest with residual force acting on the forearms.

injuries brings its own challenge along but is potentially very rewarding when undertaking studies like this. In Fig. 8.8 the contact interactions of the model's head (back and side), its left forearm and up arm and the right up arm are displayed. The force amplitudes are hereby normalised.

8.2.4 Model limitations and outlook

Especially in legal problems that impact a humans fate, a keen eye must be kept on the significance of results. Biomechanics analysis in the form of forward dynamics simulations are still rarely seen in legal questions and one reason is the complexity not only of human behaviour but also of its 'model'. As it is not possible to model a real human, always some kind of surrogate model must be used in such a simulation study. The used model itself influences the conclusions that can be drawn from the simulation's results (Pinter et al., 2012). The DHM that is used in this application case is a simplified model as well, e.g., only 2 MTUs are modelled for each DoF and the Hp and Sh joint cannot perform their internal or external rotations. This missing Hp movement capability, for example, has an impact on the possible end positions that can be achieved in the here shown simulations. A DHM with increased biofidelity, e.g. Hp rotation, or a real human, has a bigger *range of motion* (RoM) and a wider range of possible end positions after a fall.

Also the used implementation of discrete contact interactions with the environment has its limitations. With such an approach of a limited number of contact points, the question of: "where exactly did the head hit the wall?" can never be predictively answered. Additionally, each contact interaction involves an acceleration of the model and has an influence on the movement to come. With a discrete and finite set of contact points it is hard to obtain simulation results that reflect the exact physics of the visual impression that the animation suggests. A fine array of discrete contact points or a continuous *surface-surface* contact algorithm can sufficiently resolve this issue in future studies.

Part IV

Discussion

Discussion and outlook 9

The presented hierarchical control architecture showed to be a practical approach to actuate the used DHM in the examples of joint angle, joint torque and position control. Due to the hierarchical architecture, only a few intuitive changes in the conceptional layer led to the synthetisation of different muscle-actuated movements, including the complex combined movements of a car-ingress motion and a falling motion into a bathtub. These examples hint in favour of the architecture to be capable of synthesising a wide range of arbitrary movements. The internal biophysical data of the DHM, which is generated along with the simulation of the movement, is meant to predict the consequences for a real human moving.

With the separation of the control system into a low-dimensional planning space ('conceptional layer') and a high-dimensional actuation space ('structural layer') (see Fig. 3.1), the problem of movement planning was eased. With the presented hierarchical control architecture, movement planning can occur in a conceptional context only, without having to consider the properties of the biological actuators. i.e. muscles and tendons. The muscle actuation is achieved by Jacobian-based layer transformations that translate the conceptionally obtained movement plan to muscle stimulations. These transformations resolve the redundancies of the different layer's coordinate spaces, by using anatomical and tissue knowledge. The 'transformational layer' hereby serves as a standardised pathway that receives a 'postural plan' from the 'conceptional layer' and provides inputs for the 'structural layer'.

The Jacobian matrices, which are used for the transformations, are always presented in closed-form. This gives structural insights into the required calculations and reveals exactly which sensor signals and body properties are needed for resolving the redundancy. Precisely since resolving the muscle-joint redundancy is not unique, additional DoFs on an uncontrolled manifold remain. In control, this uncontrolled manifold allows to fulfil additional criteria, such as joint-wide co-contraction, along—and not interfering—with the execution of the planned movement (see Fig. 6.2).

Additionally, on each layer, PID-controllers are implemented to ensure a robust and stable task fulfilment. While the overall modular design of the hierarchical control architecture allows to use other control ideas instead, the choice to use PID-controllers in this work has several reasons: i), the low-level λ-controller that forms the basis of the hierarchical control architecture in the structural layer has the form of a P(D)-controller (3.4). By generalisation of this principle, PID-controllers are used in the remaining layers. ii), a PID-controller is a very simple and generic linear system that is easy to comprehend (Wescott, 2000). For all simulations performed in this thesis, a heuristic tuning of the control parameters was sufficient. A system-theoretical stability analysis or automated optimisation wasn't necessary.

iii), the PID-controlled closed-loop system has a satisfactory performance, while still being structurally resolved. Even in their cascaded, hierarchical implementation, the modular use of the PID-controllers does not overcomplicate the architecture's design. Also, all calculations are expressed in closed-forms and all signal pathways can be retraced (see Fig. 3.4). This is an important feature of the hierarchical control architecture that allows to engage validation experiments in future studies. In its current state, the hierarchical control can already be used for the synthesis of various complex movements of muscle-actuated systems.

9.1 Steps towards a validation and biological identification of the hierarchical control architecture

Approaching a validation of the hierarchical control architecture as a model of biological motor control must be considered in a twofold way, at least. First, how, and in which detail, does the model's behaviour compare to its real world, biological pendant? And second, how is the model, or its parts, represented by biological structures?

Regarding the first consideration, it must be kept in mind that the presented hierarchical control architecture is a feedback system that relies on a mathematical model of the biological system, e.g. the DHM from Sec. 4. In computational studies, the validity of the control architecture can only be observed when embedded in the whole closed-loop system that incorporates both, the dynamics of the control system and of the DHM. Any model limitation of the DHM (see Sec. 9.2) strongly impacts the behaviour of the closed-loop system and thereby influences the outcome of a check of validity. By increasing the biofidelity and validity of the used DHM in an isolated validation check, the focus of the closed-loop validation moves closer towards the control structure. For validation, real world experiments must be conducted in parallel to a computational movement synthesis, both with the same task formulation and boundary restriction. By comparing the model's output trajectories to the phenomenological experimental measurements, a quality estimation of the model can be made. In an iterative process, the model's quality can potentially be increased by adapting its parameters. If a model still cannot explain some of the phenomenological observations, its structure must be questioned, altered, augmented or eventually discarded. Investigating internal states in addition to the system's output can help in refining this process. However, this is only possible, when the model's structure is closely related to the biological system and only for signals that can be experimentally measured. It is a fundamental problem of biological motor control that, due to the sheer amount and complexity of neural connections, higher-level control signals cannot easily be extracted or deciphered from the biological system. Although the intention of a movement, for example, can be correlated with an increased activity in certain brain regions (Desmurget et al., 2009), finding its origin is a chase of the philosophical question of what consciousness is (Crick and Koch, 1990; Chalmers, 2000). The simple mathematical notion of a postural plan $\boldsymbol{\theta}^{\mathrm{des}}(t)$ that is introduced in the hierarchical control architecture (see Figs. 3.1 and 3.4), for example, may be impossible to find in the same form in biological measurements. This is, because the presented hierarchical control architecture is a simplified model approach that is structurally more resolved than the grey matter that represents the control structures in the biological system. For a validation of the general principles used by the hierarchical control architecture, at first, *electromyography* (EMG) measurements of the muscular activity of all muscles in an AAS should be performed. The synergistic activations in this AAS can then be compared to the synthesized prediction of the hierarchical control architecture, which is the outcome of a Jacobian-based layer transformation. A high correlation in the synergised stimulation patterns could hint towards similar underlying calculations in the biological system.

Regarding the second consideration for validation, even if the phenomenological observations of the biological system are sufficiently reflected by the computational model,

biological structures must be identified that are capable of performing the required 'calculations'. For the presented hierarchical control architecture, this means that neural circuities must be identified that are capable of performing the required calculations, e.g. of the Jacobian transformation (3.16). While such calculations have been proposed by Pellionisz and Llinás (1985) to be potentially contributed to the cerebellum, a more decentralised implementation in the form of weighted neuronal connections within the spinal cord may also be plausible (compare to Windhorst (2007); Eccles (1967); Gao et al. (1996); Doya (2000)). A hint towards such a decentralisation lies in the modular nature of the Jacobian-based transformations. An angle-length Jacobian matrix, for example, naturally consists of blocks for the single limbs and, in each of these block, a single matrix column corresponds to the movement of a single joint with all relevant muscles arranged in the non-zero matrix rows (compare, e.g. Fig. 7.3 c) and Fig. 7.4 c)). The respective matrix entries of multiple muscles in an AAS have different signs, depending on the muscle's function on the joint. When contributed to the spinal cord, the different signs may implicitly correspond to the local implementation of inhibitory interneurons. In more complex movements that partly rely on visual feedback, e.g. position control, a more centralised biological 'implementation' in higher centres of the CNS is plausible. Although, general open question remain, e.g. see Karniel (2011); Kawai et al. (2015); Gao et al. (1996).

With the in-silico applications in Pt. III of this dissertation, the presented hierarchical control architecture accomplished its first rudimentary test towards validation. It showed to provide appropriate MTU stimulation signals to actuate the used DHM, simply as the resulting MTU forces and joint torques, respectively, successfully executed the desired movements. A set of inappropriate MTU stimulations, on the other hand, could not have led to a successful synthetisation of human upright stance as presented in Sec. 6.1.1. This hints towards a plausibility of the general hierarchical approach using cascaded control laws and Jacobian-based transformations. Still, as the achieved performance of upright stance does not fully reflect the behaviour of a real human (Günther et al., 2011, 2012), further validation measures are necessary to improve the architecture's parametrisation and to refine its design.

9.2 Model limitations of the DHM

The control architecture was tested for a full-body—but simplified—musculo-skeletal model. Regarding the DHM, there are at least three important simplifications in the context of this dissertation.

The first simplification of the DHM is the reduced number of muscles with only one antagonistic pair per mechanical DoF and no bi-articular muscles[1]. Additionally, each muscle is represented by only one motor unit (Haeufle et al., 2014a). Both reduces the redundancy in the model, i.e. only thirty-six muscle stimulation signals are required to control the full-body model instead of several thousands of α-motoneuron signals in the real human body (de Luca and Contessa, 2012). While the control architecture would allow to include more redundant and also bi-articular muscles, the possible benefit, e.g. for decoupling of parallel tasks (Latash, 2012; Hsu and Scholz, 2012) cannot be evaluated with the current DHM.

The second simplification addresses the mechanical DoFs of the DHM. Especially the Hp joints, the Sh joints and the spine are drastically simplified. The Hp and Sh joints are modelled with only having two DoFs each and the spine is a rigid structure with the only DoFs in the cervical and lumbar region. Advancing the model to have three DoFs in the Hp and the Sh is still very simplistic, but leads to a much wider range of motion of the DHM. Breaking up the stiff connections of the single vertebrae of the spine requires to

[1] Technically, the Hp joint functions as a bi-articular joint, as a single Hp MTU spans over the two DoFs of Hp FE and AA.

introduce additional muscles and models of related biological tissue, such as ligaments and intervertebral discs, e.g. according to Mörl et al. (2020); Rupp et al. (2015); Karajan et al. (2013).

The third simplification of the DHM considers biological sensors. In the biological system all sensor information is provided by a multitude of sensor organs, e.g. see Windhorst (2007); Mileusnic et al. (2006); Blecher et al. (2018). These organs are formed by specialised cells and their efferent output inherits the non-linear dynamics that are intrinsic to biology. To have a more detailed model of the reality, dynamical models of the proprioceptive biological sensor organs should be integrated in the DHM, e.g. a model of the muscle spindle based on Mileusnic et al. (2006). As the muscle spindle's efferent neural connections forms a basis of biological motor control by the mono-synaptic reflex, a refined sensor model may lead towards a refined model of the 'λ-controller'.

9.3 Control limitations with potential modular improvements to the architecture's design

There are several aspects of biological motor control that are not captured by the hierarchical control architecture in the form in which it is introduced in this dissertation. This includes questions on intermittent control (Gawthrop et al., 2011; Loram et al., 2011), state estimation and motor prediction (Wolpert and Ghahramani, 2000; Wolpert and Flanagan, 2001), decision making (Wolpert and Landy, 2012), learning, adaptation and optimisation (Shadmehr and Mussa-Ivaldi, 1994; Shadmehr et al., 2010; Mussa-Ivaldi, 1999; Wochner et al., 2020)

Even if these aspects are not considered in this work, the hierarchical control architecture is structurally prepared to do so. Its modular design (see Fig. 3.4) in general allows to implement technical solutions of neurological hypotheses and to analyse their applicability for complex movements in forward dynamics simulations.

Intermittent control, for example, can be engaged by introducing signal processing blocks, similar to the delay blocks in the Figs. 2.1, 3.2 and 3.4, which hold a sampled signal's value constant, until a new sample is taken that updates the signal. Critical parameters for stability hereby include the sample-time duration in which the signal is held constant until it is newly updated. An event-driven approach for updating the signal may help in maintaining stability, when the sample-time is considered in the stability analysis, as e.g. achieved by Ibuki et al. (2014) for a technical camera-based system.

State estimation and prediction is a broad field that can be approached from different starting points. By motor prediction, for example, the control performance can be increased and voluntary and external body movements can be distinguished, see also Wolpert and Flanagan (2001). From a system-theoretic point of view, a *Luenberger*-type observer system may be the most simple and straightforward implementation (Luenberger, 1971). The general idea is to implement a copy of the controlled musculo-skeletal system (i.e. the observer) within the feedback controller and to feed the same stimulation input to this observer. The model in the observer then generates an estimation or prediction of the resulting movement, which is compared to the actual generated movement of the musculo-skeletal system. This closes the observer feedback loop and, by a stabilising parametrisation, the estimation of the observer system approaches the exact behaviour of the real model. A big advantage of having such an observer system implemented is that otherwise unmeasurable signals are available within the control system. For complex non-linear dynamics that are intrinsic to biological motion, non-linear approaches seem appropriate, e.g. see Zeitz (1987); Reif and Unbehauen (1999). Another promising approach to engage predictive control of biological motion is the use of *model predictive control* (MPC), e.g. see Morari and Lee (1999); Allgöwer and Zheng (2012); Berberich et al. (2020). The presented hierarchical control architecture hereby introduces the advantage that a MPC approach can be implemented

on the highest, conceptional layer, while still considering low-level, structural constraints (Koehler et al., 2021), such as minimal MTU stimulations.

In the presented control approach it is mostly assumed that the required sensor information are directly available. Moreover, some state information of the system may not be directly available from a 'biological sensor' and is moreover the outcome of sensor fusion or multisensory integration (Elmenreich, 2002). Also the Jacobian-based transformations that are used in the hierarchical control architecture can be used in a reversed direction to obtain estimations of higher-level system information from lower-level sensor signals. In combination with an observer system as discussed above, a sophisticated control structure can be constructed that is capable of judging the quality of single contributing sources of a sensory signal and thereby to compensate misleading information. By enabling further sensor signals for the conceptional planning, or in structural reflex models, the versatility and performance of the control architecture may be increased.

Biological motion is driven by feedforward and feedback mechanisms. Several studies suggest that the contribution of feedforward signals increase during learning, as e.g. outlined by Seidler et al. (2004) and tested by Haeufle et al. (2012); Müller et al. (2015). The presented hierarchical control architecture, as a feedback controller system, lacks most feedforward mechanisms and moreover remodelled the 'open-loop' stimulation contribution of the hybrid λ-controller (Kistemaker et al., 2007; Bayer et al., 2017; Günther and Ruder, 2003) to be a higher-level, closed-loop stimulation contribution. In this implementation and according to Seidler et al. (2004); Adams (1971); Pratt et al. (1994), the hierarchical control architecture must be considered to resemble an unskilled child. It is an exciting challenge to include adaptive learning methods in the architecture. This can start with simple self-tuning techniques, e.g. see Gawthrop (1986) and lead towards more complex methods of learning of action through an adaptive combination of motor primitives (Thoroughman and Shadmehr, 2000).

For the simulation tasks investigated here, also a heuristic tuning of the PID parameters was possible and successful. The heuristic parameter tuning, on the other hand, may not be ideal, as there are ambiguities of the parameters in the cascaded controller setup. This problem may become difficult to infeasible for models with more DoFs or for more complex or dynamic movements. A systematic optimisation of the control parameters is worth being investigated in future. However, the use of optimisation techniques introduces the question of what an optimal behaviour is (compare to Wochner et al. (2020)).

Furthermore, the stabilising effect of the control architecture on such a non-linear system, e.g. the DHM, cannot be guaranteed yet and therefore still relies on tests via simulations. However, stability has been mathematically proven in a simplified model (Brändle et al., 2020) for the structural layer and—from all simulations investigated here—seems practically achievable even in the hierarchical architecture.

The DHM considers critical muscular non-linearities and elasticities, which are expected to be difficult properties in the sense of control (Brändle et al., 2020) but central for understanding human movement control (van Soest and Bobbert, 1993; Pinter et al., 2012; Stollenmaier et al., 2020). It is remarkable that, despite all the linear (Taylor) approximations in the Jacobians (3.9,3.26, 3.36) and the the quasi-static assumptions for the relation between muscle-internal stiffnesses (3.23), the control architecture is able to handle these non-linear elastic characteristics of the DHM.

9.4 Current state and future applications of the hierarchical control architecture

In its current state, the hierarchical control architecture forms a powerful technical tool for synthesising biological movements. The provided examples of upright standing, joint and position control can be used to create a pool of movement primitives. When some of

these movement primitives are successively combined, complex movements can be generated. Two examples thereof have already been presented in Sec. 8.1 by the synthetisation of a car ingress motion and in Sec. 8.2 by a forensic case analysis. By further utilising the architecture for this purpose, more movements of various kind can be synthesized and the internal states of the DHM can be subsequently evaluated to extract ergonomics or medical measures. Following the example of Schmitt (2003), with the hierarchical control architecture a broad range of complex movement sequences may be investigated, e.g. from athletic disciplines like running or cycling. Applying optimisation techniques may lead to a refinement of an athlete's technique to increase performance or to lower the risk of injuries. The same approach may be applicable for investigating diverse body plans (animal morphologies), for evaluating processes of manual labour and for analysing the effects of prosthetic devices on the human body.

Putting this to the next step, a prosthetic device may not only be evaluated as aforementioned but also be driven by the hierarchical control architecture, when the device itself is actuated by artificial muscles, e.g. Wolfen et al. (2018); Acome et al. (2018). That is, in the same way as the architecture is used in this dissertation to actuate a DHM, it can be used to actuate a real-life artificial-muscle-actuated robotic system. A robotic system that is driven by artificial muscles has the potential to perform more human-like motions and thereby to improve human-robotic interactions. As a required step, Jacobian matrices similar to (3.16), (3.27) and (3.36) are required, which are dependent on the characteristics of the artificial muscle-type. It would be most interesting to implement learning based approaches for obtaining these Jacobians for a robotic system. Especially, as the required morphological information that is needed for its calculation was revealed in this dissertation.

A third field of application for the hierarchical control architecture arises from its modular design. When implementing one or another of the ideas mentioned in Sec. 9.3, the presented hierarchical control architecture may serve as a tool to investigate the general organisation of biological motor control. Especially when validated on experimental data, the architecture's capability of movement synthetisation allows to test neurological hypothesis on a muscle-actuated system and therefore to deduce the biological design. A first example thereof are the results presented in Sec. 6 and in Walter et al. (2021a). There, the general hypothesis of a torque-based planning process, which is also assumed by Günther and Wagner (2016); Rozendaal and van Soest (2005); Alexandrov et al. (2001); Edwards (2007), was bolstered by applying it to a muscle-actuated system. While this does not yet represent a proof for the biological system's organisation, it showcases by success that the general principle is indeed viable.

In addition to all this, by the use of the presented hierarchical control architecture, new questions on the organisation of biological motor control arise. The choice of PID control parameters, as a simple example, has a significant impact on the dynamics of the executed movement. That is, higher feedback gains can lead to a faster execution of the movement and a more reserved set of (PID) parameters to a slower and smoother behaviour (Wescott, 2000). It is thereby imaginable that, to execute a desired movement, not only the movement plan as an input to the controller is adapted, but also the controllers' parameters. On the other hand, these control parameters strongly influence the stability of the closed loop movement system. Interestingly, even a parametrisation close to destabilisation can lead to plausible movements: While holding the hand in the air, a destabilisation, e.g. by the D-part, evokes an oscillation. By 'controlled destabilisation' a waving movements is realised. This is remarkable, as the movement is then initiated by changing a single parameter.

Part V

Appendix

A.1 Notes on rotations in 3D-space and other representations

In this section, different approaches to obtain such a rotation matrix - and to extract the 3 individual angles from a given rotation matrix, are presented. The different representations each have advantages and drawbacks. The *Euler* and *Cardan* representations, which are also implemented in some parts of `demoa`, e.g., have singularities at certain angle configurations. Using *exponential coordinates* or *quaternions* is mathematically more stable in this regard, but they may not be as intuitive to be mapped on anatomical angles. The improved mathematical stability can be traced back to the fact that *exponential coordinates* or *quaternions* use four parameters to describe the rotation, while the *Euler* and *Cardan* representations only use three. It is a fundamental topological fact that singularities can never be eliminated in 3D rotations when using only 3 parameters for parametrization Murray et al. (1994), it is yet possible to avoid them, e.g., by early detection and switching to another convention where the singularities are located differently (Henze, 2002).

Quaternions Quaternions are a representation closely related to the exponential coordinates, which are based on Euler's theorem 2.1.1 (see Sec. 2.1.2). As such, the four elements of a quaternion are also commonly known as *Euler parameters* (Sarabandi and Thomas, 2019). A quaternion can be written as

$$Q = q_0 + q_1 i + q_2 j + q_3 k, \tag{A.1}$$

where $q_i \in \mathbb{R}$ are the *Euler parameters* and i, j and k are generalized imaginary numbers. Following *Rodrigues' formula* (2.6) the rotation matrix $\boldsymbol{R}$, for a given angle θ and axis $\boldsymbol{\omega} = \begin{bmatrix}\omega_1 & \omega_2 & \omega_3\end{bmatrix}^T$, takes the form

$$\boldsymbol{R} = \begin{bmatrix} c + \omega_1^2(1-c) & \omega_1\omega_2(1-c) - \omega_3 s & \omega_1\omega_3(1-c) + \omega_2 s \\ \omega_2\omega_1(1-c) + \omega_3 s & c + \omega_2^2(1-c) & \omega_2\omega_3(1-c) - \omega_1 s \\ \omega_3\omega_1(1-c) - \omega_2 s & \omega_3\omega_2(1-c) + \omega_1 s & c + \omega_3^2(1-c) \end{bmatrix}, \tag{A.2}$$

where $c = \cos(\theta)$ and $s = \sin(\theta)$. By defining the *Euler parameters*, or the elements of a quaternion as

$$q_0 := \cos(\tfrac{\theta}{2}) \tag{A.3}$$
$$q_1 := \omega_1 \sin(\tfrac{\theta}{2}) \tag{A.4}$$
$$q_2 := \omega_2 \sin(\tfrac{\theta}{2}) \tag{A.5}$$
$$q_3 := \omega_3 \sin(\tfrac{\theta}{2}), \tag{A.6}$$

the rotation matrix $\boldsymbol{R}$ can then be written as

$$\boldsymbol{R} = \begin{bmatrix} 2(q_0^2 + q_1^2) - 1 & 2(q_1 q_2 - q_0 q_3) & 2(1_1 q_3 + q_0 q_2) \\ 2(q_1 q_2 + q_0 q_3) & 2(q_0^2 + q_2^2) - 1 & 2(q_1 q_3 - q_0 q_1) \\ 2(1_1 q_3 - q_0 q_2) & 2(q_2 q_3 + q_0 q_1) & 2(q_0^2 + q_3^2) - 1 \end{bmatrix}. \tag{A.7}$$

This equation gives a way to calculate a rotation matrix from quaternions.

For calculating a quaternion from a rotation matrix, several approaches exist. The most straightforward way is to firstly calculate the Euler axis $\boldsymbol{\omega}$ and the rotation angle θ according to (2.7) and (2.8) and then the quaternion elements according to (A.3)-(A.6). This is not the only way to calculate quaternions from a rotation matrix, though. Especially when numerical aspects are considered, multiple variations for obtaining robust representations of quaternions have been proposed. Instead of listing those different representations here, it is referred here to a survey on quaternion computation Sarabandi and Thomas (2019). As their conclusion, the algebraic method of *Cayley* is among the simplest and yet the best in terms of time and error performance. It is designed to obtain double-quaternion representations for 4D rotations, but can be simplified to the 3D-case. For $r_{ij} \in \mathbb{R}$ being the elements of a rotation matrix $\boldsymbol{R} \in SO(3)$, the quaternions by *Cayley's method* are given by

$$q_0 = \tfrac{1}{4}\sqrt{(r_{11} + r_{22} + r_{33} + 1)^2 + (r_{32} - r_{23})^2 + (r_{13} - r_{31})^2 + (r_{21} - r_{12})^2} \tag{A.8}$$
$$q_1 = \tfrac{1}{4}\sqrt{(r_{32} - r_{23})^2 + (r_{11} - r_{22} - r_{33} + 1)^2 + (r_{13} + r_{31})^2 + (r_{21} + r_{12})^2} \tag{A.9}$$
$$q_2 = \tfrac{1}{4}\sqrt{(r_{13} - r_{31})^2 + (r_{21} + r_{12})^2 + (r_{11} - r_{22} - r_{33} + 1)^2 + (r_{32} + r_{23})^2} \tag{A.10}$$
$$q_3 = \tfrac{1}{4}\sqrt{(r_{21} - r_{12})^2 + (r_{13} + r_{31})^2 + (r_{32} - r_{23})^2 + (r_{11} - r_{22} - r_{33} + 1)^2}. \tag{A.11}$$

This representation has the advantage that it contains information from all entries of the rotation matrix $\boldsymbol{R}$ and does not contain any divisions. It is also well-defined in terms of obtaining partial derivatives of the quaternion elements with respect to the rotation matrix entries (Sarabandi and Thomas, 2019).

Euler/Cardan The representations of rotations using *Euler* or *Cardan* angles may be the most straightforward to understand and are considered to have the most intuitive relation to physiological angles in biomechanical studies. This may also be why the ISB recommends the usage of *Euler* or *Cardan* angles, although in 3D not even the clinical definitions are consistent (Sarabandi and Thomas, 2019; Wu et al., 2002, 2005; Sinclair et al., 2012). In contrast to the *exponential coordinates* and *quaternions* from above that use 4 parameters, *Euler* and *Cardan* only use 3 parameters for the parametrization of the rotation. This makes them prone to singularities at certain angle configurations.

Euler and *Cardan* angles have in common, that they use three individual elementary rotations around the three Cartesian coordinate axes by the three angles α, β and γ. For a coordinate system with the basis $\boldsymbol{e} = \begin{bmatrix} e_x & e_y & e_z \end{bmatrix}^T$, the three elementary rotations are

given by

$$\boldsymbol{R}_x(\alpha) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\alpha) & -\sin(\alpha) \\ 0 & \sin(\alpha) & \cos(\alpha) \end{bmatrix}, \quad \boldsymbol{R}_y(\beta) = \begin{bmatrix} \cos(\beta) & 0 & \sin(\beta) \\ 0 & 1 & 0 \\ -\sin(\beta) & 0 & \cos(\beta) \end{bmatrix} \tag{A.12}$$

$$\text{and} \quad \boldsymbol{R}_z(\gamma) = \begin{bmatrix} \cos(\gamma) & -\sin(\gamma) & 0 \\ \sin(\gamma) & \cos(\gamma) & 0 \\ 0 & 0 & 1 \end{bmatrix}. \tag{A.13}$$

With these elementary rotations, there exist multiple conventions on the axes and the order of rotation. An elementary rotation can thereby be performed around a original axis or around an already rotated axis. In the *Cardan* convention, all individual rotations are performed on different axes, e.g, $X-Y-Z$ or $Y-X-Z$[1], while in *Euler* convention the first and the last rotation are performed around the same axis, e.g. $Z-X-Z$[1]. The rotation matrix of the final 3D rotation is then obtained by multiplying the three elementary rotations. As an example the Rotation matrix $\mathbb{R}_{yxz}$ for the $Y-X-Z$ *Cardan* convention for given angles α, β and γ is obtained by

$$\boldsymbol{R}_{yxz}(\alpha,\beta,\gamma) = \boldsymbol{R}_y \cdot \boldsymbol{R}_x \cdot \boldsymbol{R}_z \tag{A.14}$$

$$= \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\alpha) & -\sin(\alpha) \\ 0 & \sin(\alpha) & \cos(\alpha) \end{bmatrix} \cdot \begin{bmatrix} \cos(\beta) & 0 & \sin(\beta) \\ 0 & 1 & 0 \\ -\sin(\beta) & 0 & \cos(\beta) \end{bmatrix} \cdot \begin{bmatrix} \cos(\gamma) & -\sin(\gamma) & 0 \\ \sin(\gamma) & \cos(\gamma) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{A.15}$$

$$= \begin{bmatrix} c_\alpha \cdot c_\gamma + s_\alpha \cdot s_\beta \cdot s_\gamma & s_\alpha \cdot s_\beta \cdot c_\gamma - c_\alpha s_\gamma & s_\alpha \cdot c_\beta \\ c_\beta \cdot s_\gamma & c_\beta \cdot c_\gamma & -s_\beta \\ c_\alpha \cdot s_\beta \cdot s_\gamma - s_\alpha \cdot c_\gamma & s_\alpha \cdot s_\gamma + c_\alpha \cdot s_\beta \cdot c_\gamma & c_\alpha \cdot c_\beta \end{bmatrix}, \tag{A.16}$$

with $s_\phi = \sin(\phi)$ and $c_\phi = \cos(\phi)$. With equations of the kind of (A.16) it is also possible to retrieve the rotation angles α, β and γ from a rotation matrix $\boldsymbol{R}$. The inverse rotation of, e.g, $\boldsymbol{R}_{yxz}(\alpha,\beta,\gamma)$ is given by $\boldsymbol{R}^{-1}_{yxz}(\alpha,\beta,\gamma) = \boldsymbol{R}_{zxy}(-\alpha,-\beta,-\gamma)$, which can simply be verified by inverting (transposing) (A.14).

It is important to note that, regardless of which *Cardan* or *Euler* convention is used, these representations of rotations are prone to encounter singularities at certain angle configurations. This is a fundamental topological fact for all representation of 3D-rotation that uses only three parameters Murray et al. (1994). In `demoa` the problem is handled by an early detection of being close to such a singularity and then switching from the *Cardan* $Y-X-Z$ to the *Euler* $Z-X-Z$ convention, which have their singularities at different spots (Henze, 2002).

A.2 Body velocities and the adjoint transformation matrix

In Sec. 2.1.5 the notion of *spatial velocities* $\boldsymbol{\nu}^{\mathcal{A}}_{\mathcal{A}p_B}$ was presented as the velocity of the point $\boldsymbol{p}_B$ relative to the frame $\mathcal{A}$ (subscript) as viewed from the frame $\mathcal{A}$ (superscript). Another representation of rigid body velocity is the *body velocity* $\nu^{\mathcal{B}}_{\mathcal{A}p_B}$, namely, the relative velocity of the point $\boldsymbol{p}_B$ and the frame $\mathcal{A}$ as viewed from the body frame $\mathcal{B}$. It can be constructed by simple calculation by inverting the following identity (Murray et al., 1994)

$$\nu^{\mathcal{A}}_{\mathcal{A}p_B} = \boldsymbol{G}^{\mathcal{A}\mathcal{B}} \cdot \nu^{\mathcal{B}}_{\mathcal{A}p_B}, \tag{A.17}$$

[1]The $Y-X-Z$ *Cardan* and the $Z-X-Z$ *Euler* conventions are used in `demoa` (Henze, 2002)

which, by utilising (2.22)-(2.23), further yields

$$\begin{aligned} \nu^{\mathcal{B}}_{\mathcal{A}p_B} &= \boldsymbol{G}^{\mathcal{B}\mathcal{A}} \cdot \nu^{\mathcal{A}}_{\mathcal{A}p_B} && \text{(A.18)} \\ &= \underbrace{\boldsymbol{G}^{\mathcal{A}\mathcal{B}^{-1}} \cdot \dot{\boldsymbol{G}}^{\mathcal{A}\mathcal{B}}}_{:=\hat{\boldsymbol{V}}^{\mathcal{B}}_{\mathcal{A}\mathcal{B}}} \cdot \underbrace{\boldsymbol{G}^{\mathcal{A}\mathcal{B}^{-1}} \cdot \boldsymbol{p}^{\mathcal{A}}_B}_{=\boldsymbol{p}^{\mathcal{B}}_B} && \text{(A.19)} \\ &= \hat{\boldsymbol{V}}^{\mathcal{B}}_{\mathcal{A}\mathcal{B}} \cdot \boldsymbol{p}^{\mathcal{B}}_B. && \text{(A.20)} \end{aligned}$$

And thus, the homogeneous representation of the body velocity is

$$\hat{\boldsymbol{V}}^{\mathcal{B}}_{\mathcal{A}\mathcal{B}} = \boldsymbol{G}^{\mathcal{A}\mathcal{B}^{-1}} \cdot \dot{\boldsymbol{G}}^{\mathcal{A}\mathcal{B}} = \begin{bmatrix} \boldsymbol{R}^{\mathcal{A}\mathcal{B}^T}\dot{\boldsymbol{R}}^{\mathcal{A}\mathcal{B}} & \boldsymbol{R}^{\mathcal{A}\mathcal{B}^T}\dot{\boldsymbol{t}}^{\mathcal{A}\mathcal{B}} \\ 0 & 0 \end{bmatrix} := \begin{bmatrix} \hat{\boldsymbol{\Omega}}^{\mathcal{B}}_{\mathcal{A}\mathcal{B}} & \boldsymbol{v}^{\mathcal{B}}_{\mathcal{A}\mathcal{B}} \\ 0 & 0 \end{bmatrix} \in se(3) \subset \mathbb{R}^{4\times 4} \tag{A.21}$$

and the corresponding body velocity coordinates are

$$\boldsymbol{v}^{\mathcal{B}}_{\mathcal{A}\mathcal{B}} = \begin{bmatrix} \boldsymbol{R}^{\mathcal{A}\mathcal{B}^T} \cdot \dot{\boldsymbol{t}}^{\mathcal{A}\mathcal{B}} \\ \left(\boldsymbol{R}^{\mathcal{A}\mathcal{B}^T} \cdot \dot{\boldsymbol{R}}^{\mathcal{A}\mathcal{B}}\right)^{\vee} \end{bmatrix} = \begin{bmatrix} \boldsymbol{\nu}^{\mathcal{B}}_{\mathcal{A}\mathcal{B}} \\ \boldsymbol{\omega}^{\mathcal{B}}_{\mathcal{A}\mathcal{B}} \end{bmatrix}. \tag{A.22}$$

Again by using the respective definitions the body velocity and the spatial velocity can be transformed into each other. This can also be interpreted as a change of the reference coordinate frame:

$$\hat{\boldsymbol{V}}^{\mathcal{A}}_{\mathcal{A}\mathcal{B}} = \boldsymbol{G}^{\mathcal{A}\mathcal{B}}\hat{\boldsymbol{V}}^{\mathcal{B}}_{\mathcal{A}\mathcal{B}}\boldsymbol{G}^{\mathcal{A}\mathcal{B}^{-1}} \qquad \text{and} \qquad \hat{\boldsymbol{V}}^{\mathcal{B}}_{\mathcal{A}\mathcal{B}} = \boldsymbol{G}^{\mathcal{A}\mathcal{B}^{-1}}\hat{\boldsymbol{V}}^{\mathcal{A}}_{\mathcal{A}\mathcal{B}}\boldsymbol{G}^{\mathcal{A}\mathcal{B}}. \tag{A.23}$$

For transforming the coordinate representations $\boldsymbol{v}^{\mathcal{A}}_{\mathcal{A}\mathcal{B}} \in \mathbb{R}^6$ and $\boldsymbol{v}^{\mathcal{B}}_{\mathcal{A}\mathcal{B}} \in \mathbb{R}^6$ a convenient matrix operation is introduced that is an equivalent operation to (A.23) for vectors in $\mathbb{R}^6$:

$$\begin{bmatrix} \boldsymbol{\nu}^{\mathcal{A}}_{\mathcal{A}\mathcal{B}} \\ \boldsymbol{\omega}^{\mathcal{A}}_{\mathcal{A}\mathcal{B}} \end{bmatrix} = \underbrace{\begin{bmatrix} \boldsymbol{R}^{\mathcal{A}\mathcal{B}} & \overbrace{\hat{\boldsymbol{T}}^{\mathcal{A}\mathcal{B}} \cdot \boldsymbol{R}^{\mathcal{A}\mathcal{B}}}^{(\hat{=}(\boldsymbol{t}^{\mathcal{A}\mathcal{B}})^{\wedge})} \\ 0 & \boldsymbol{R}^{\mathcal{A}\mathcal{B}} \end{bmatrix}}_{:=\mathbf{Ad}^{\mathcal{A}\mathcal{B}}_G} \cdot \begin{bmatrix} \boldsymbol{\nu}^{\mathcal{B}}_{\mathcal{A}\mathcal{B}} \\ \boldsymbol{\omega}^{\mathcal{B}}_{\mathcal{A}\mathcal{B}} \end{bmatrix} \quad \Leftrightarrow \quad \boxed{\boldsymbol{v}^{\mathcal{A}}_{\mathcal{A}\mathcal{B}} = \mathbf{Ad}^{\mathcal{A}\mathcal{B}}_G \cdot \boldsymbol{v}^{\mathcal{B}}_{\mathcal{A}\mathcal{B}}.} \tag{A.24}$$

The matrix

$$\boxed{\mathbf{Ad}^{\mathcal{A}\mathcal{B}}_G := \begin{bmatrix} \boldsymbol{R}^{\mathcal{A}\mathcal{B}} & \hat{\boldsymbol{T}}^{\mathcal{A}\mathcal{B}} \cdot \boldsymbol{R}^{\mathcal{A}\mathcal{B}} \\ 0 & \boldsymbol{R}^{\mathcal{A}\mathcal{B}} \end{bmatrix} \in \mathbb{R}^{6\times 6}, \text{ with } \mathbf{Ad}^{\mathcal{A}\mathcal{B}^{-1}}_G = \begin{bmatrix} \boldsymbol{R}^{\mathcal{A}\mathcal{B}^T} & -\boldsymbol{R}^{\mathcal{A}\mathcal{B}^T}\hat{\boldsymbol{T}}^{\mathcal{A}\mathcal{B}} \\ 0 & \boldsymbol{R}^{\mathcal{A}\mathcal{B}^T} \end{bmatrix} = \mathbf{Ad}^{\mathcal{A}\mathcal{B}}_{G^{-1}}} \tag{A.25}$$

is called the *adjoint transformation matrix* and it can be used to change the velocity coordinates from the body frame $\mathcal{B}$ to the inertial (spatial) frame $\mathcal{A}$. This exactly corresponds to, first, transforming the homogeneous representation $\hat{\boldsymbol{V}}^{\mathcal{B}}_{\mathcal{A}\mathcal{B}}$ to the frame $\mathcal{A}$ by $\hat{\boldsymbol{V}}^{\mathcal{A}}_{\mathcal{A}\mathcal{B}} = \boldsymbol{G}^{\mathcal{A}\mathcal{B}}\hat{\boldsymbol{V}}^{\mathcal{B}}_{\mathcal{A}\mathcal{B}}\boldsymbol{G}^{\mathcal{A}\mathcal{B}^{-1}}$ (see (A.23)) and then extracting the velocity coordinates $\boldsymbol{v}^{\mathcal{A}}_{\mathcal{A}\mathcal{B}}$ with the $\vee$-operator:

$$\hat{\boldsymbol{V}}^{\mathcal{A}}_{\mathcal{A}\mathcal{C}} = \hat{\boldsymbol{V}}^{\mathcal{A}}_{\mathcal{A}\mathcal{B}} + \hat{\boldsymbol{V}}^{\mathcal{A}}_{\mathcal{B}\mathcal{C}} \qquad \text{or} \qquad \boldsymbol{v}^{\mathcal{A}}_{\mathcal{A}\mathcal{C}} = \boldsymbol{v}^{\mathcal{A}}_{\mathcal{A}\mathcal{B}} + \boldsymbol{v}^{\mathcal{A}}_{\mathcal{B}\mathcal{C}}. \tag{A.26}$$

$$\hat{\boldsymbol{V}}^{\mathcal{A}}_{\mathcal{A}\mathcal{C}} = \hat{\boldsymbol{V}}^{\mathcal{A}}_{\mathcal{A}\mathcal{B}} + \boldsymbol{G}^{\mathcal{A}\mathcal{B}} \cdot \hat{\boldsymbol{V}}^{\mathcal{B}}_{\mathcal{B}\mathcal{C}} \cdot \boldsymbol{G}^{\mathcal{A}\mathcal{B}^{-1}} \qquad \text{or} \qquad \boldsymbol{v}^{\mathcal{A}}_{\mathcal{A}\mathcal{C}} = \boldsymbol{v}^{\mathcal{A}}_{\mathcal{A}\mathcal{B}} + \mathbf{Ad}^{\mathcal{A}\mathcal{B}}_G \cdot \boldsymbol{v}^{\mathcal{B}}_{\mathcal{B}\mathcal{C}}. \tag{A.27}$$

A.3 Equations of motion

In this section, the calculations of the system matrices of the equations of motion () from Sec. 2.1.7 are presented. Further details on the algorithms for obtaining the equations of motion can be found in Legnani et al. (1996b,a) and Henze (2002).

The system's equations of motion are

$$\boldsymbol{M}\ddot{\boldsymbol{q}} + \boldsymbol{c}(\boldsymbol{q}, \dot{\boldsymbol{q}}) = \boldsymbol{\tau} \in \mathbb{R}^{n_{\mathrm{DoF}}}. \tag{A.28}$$

The mass matrix $\boldsymbol{M} \in \mathbb{R}^{n_{\mathrm{DoF}} \times n_{\mathrm{DoF}}}$ can be obtained by following

$$\boldsymbol{M}_{i,m} = \mathrm{trace}\left(\sum_{h=\max(i,m)}^{N} L_i^{\mathcal{W}} J_h^{\mathcal{W}} L_m^{\mathcal{W}\,T} \right) \tag{A.29}$$

and the vector $\boldsymbol{c} \in \mathbb{R}^{n_{\mathrm{DoF}}}$ by

$$\boldsymbol{c}_m = \mathrm{trace}\left(\sum_{h=m}^{N} (\tilde{\boldsymbol{H}}_{0,h} - \boldsymbol{H}_g^{\mathcal{W}}) J_h^{\mathcal{W}} L_m^{\mathcal{W}\,T} \right) - L_m^{\mathcal{W}} \otimes \sum_{h=m}^{N} \boldsymbol{\Phi}_m^{\mathcal{W}}, \tag{A.30}$$

with

$$\begin{aligned} \tilde{\boldsymbol{H}}_{0,h} &= \sum_{i=1}^{h} \left(V_{0,i-1}^{\mathcal{W}} V_i^{\mathcal{W}} - V_i^{\mathcal{W}} V_{0,i-1}^{\mathcal{W}} \right) + V_{0,h}^{\mathcal{W}\,2} \\ &= \tilde{\boldsymbol{H}}_{0,h-1} + \left(V_{0,h-1}^{\mathcal{W}} V_h^{\mathcal{W}} - V_h^{\mathcal{W}} V_{0,h-1}^{\mathcal{W}} \right) + V_{0,h}^{\mathcal{W}\,2}. \end{aligned} \tag{A.31}$$

The operator $\odot$ hereby denotes the *pseudo scalar product* between 4×4 matrices (Legnani et al., 1996a), which is defined as

$$A \otimes B := a_{3,2} b_{3,2} + a_{1,3} b_{1,3} + a_{2,1} b_{2,1} + a_{1,4} b_{1,4} + a_{2,4} b_{2,4} + a_{3,4} b_{3,4}. \tag{A.32}$$

Note, that the matrix equations (A.29)-(A.31) have the dimension $n_q = n_{\mathrm{dof}}$ and that the respective matrix entries are summed up by the individuals of the homogeneous representations for the single bodies. These matrices are obtained by the motion composition rules (2.34-2.36), which are based on equations (2.15), (2.25) and (2.31) and are easily generalised and formulated in an algorithm (Legnani et al., 1996b,a).

The contact model B

[1] In three-dimensional computer simulations, a common and most simple mathematical description of a contact between a body and a surface is to define a point fixed on the body, and approximate the surface by a plane. By then calculating, at any mechanical state of the body, the instantaneous distance between the point and the plane, a contact event between the point and the plane can be detected. In case of such an event, and with likewise knowing the movement relative to each other from the mechanical state, the force $\boldsymbol{f}$ and torque $\backslash\tau$ acting on the body (here indicated as the 'from'-body) can be modelled and calculated in terms of the body's and the plane's coordinates and velocities, as well as parameters describing the mechanical properties of the contacting and deforming materials. If the plane is fixed to another body (the 'to'-body) , the parameters of point-to-plane contacts reflect lumped properties of both bodies, and the contact force and torque on the 'from'-body simply act with inverted signs ($-\boldsymbol{f}$ and $-\backslash\tau$) on the 'to'-body, according to the principle of *actio = reactio*. Fig. B.1a illustrates such a contact situation: The contact plane is in the `demoa` implementation the x-y-plane at $\mathtt{z}=0$ of the coordinate system $\mathcal{K}_0$ (on the 'from'-body), and the origin of the coordinate system $\mathcal{K}_1$ (on the 'to'-body) is the point continuously checked for contact with $\mathcal{K}_0$'s x-y-plane. The vectors $\vec{r}$ and $\vec{v}$ of relative displacement and velocity, respectively, are depicted along with their respective projections onto $\mathcal{K}_0$'s x-y-plane (indicated by '$\parallel$') and z-axis (indicated by '$\perp$').

A two-dimensional (point-to-line) contact model approach formulated earlier (Günther, 1997; Günther and Ruder, 2003) has been used as a starting point for the three-dimensional contact model implementation in `demoa`. As body-body contacts in the real world always occur at finite surfaces, the three-dimensional `demoa` implementation gained a new part that takes, beyond the relative linear movement of the origins of $\mathcal{K}_1$ and $\mathcal{K}_0$, the angular movement of both contacting systems relative to each other into account. Accordingly, a contact torque $\backslash\tau$ is modelled in addition to the contact force $\boldsymbol{f}$. The contact model in `demoa` discriminates, firstly, the conjugate contact model states 'contact' from 'no contact' by continuously identifying the position of $\mathcal{K}_1$'s origin along $\mathcal{K}_0$'s z-axis (normal to the contact plane). Fig. B.1b gives a survey of the decision scheme to determine the overall state of a contact interaction. Secondly, it discriminates the conjugate states 'stick' from 'slip', for linear and angular movements separately, based on the tangential (linear) velocity of $\mathcal{K}_1$'s origin relative to $\mathcal{K}_0$'s origin and the torsional (angular) velocity of $\mathcal{K}_1$ relative to $\mathcal{K}_0$'s x-y-plane (see Fig.B.2), respectively, as well as the modelled force acting tangentially to

[1] The description of the contact model, which is presented here was initially written by Henze (2002) in German and was then translated to English by M. Günther for the manuscript by Walter et al. (2021a).

$\mathcal{K}_0$'s x-y-plane and the torsional torque in this plane, respectively, with all state transitions being reversible.

To maximise the physical and mathematical consistency of detecting and handling both contact and stick-slip events while modelling multiple contact interactions, `demoa` offers deploying the well-tried *Shampine-Gordon* predictor-corrector algorithm '*de*' (Shampine and Gordon, 1975) for integrating a system of ordinary differential equations in a modified (Henze, 2002) version. The modified '*de*' algorithm identifies, by quadratic polynomial interpolation backwards in time for each event detected, the time order of such (contact and stick-slip) events that occur between the instant t_{in} when '*de*' is called with an initial state vector of the neuro-musculo-mechanical system, including all discrete contact and stick-slip states, and a user-fixed instant t_{out} when '*de*' is required to return an updated state vector that fulfils the likewise user-fixed absolute and relative accuracies. Events of reversible transitions between 'contact' and 'no contact' as well as between 'stick' and 'slip' are detected, and the discrete state values accordingly and properly switched, as illustrated in Fig. B.1b, by a root-finding function that is called by the modified version of '*de*' before any '*de*'-internal (predictor or corrector) step. Additionally, the origin of a stick coordinate system $\overline{\mathcal{K}}_1$ (Fig. B.2) is fixed in $\mathcal{K}_0$'s x-y-plane at the position where contact of the origin of $\mathcal{K}_1$ has been detected by interpolation. The event detection, ordering, and handling part of the modified '*de*' algorithm guarantees that, if an event earlier than those already found is newly detected, the integration is always restarted at the instant of the earliest event found so far. Elastic stick forces and torques are then calculated depending on the linear and angular displacements of $\mathcal{K}_1$ relative to $\overline{\mathcal{K}}_1$, and friction contributions depending on their relative linear and angular velocities. The 'slip' and 'stick' force laws for modelling the contact interaction force $\vec{f}$ and torque τ as acting on the 'from'-body are explained in the following. The (tangential force and torsional torque) limit values of maximum static friction, which determine the stick-slip transition, are parametrised by the coefficients of static friction, μ_c and $\mu_{c\phi}$:

$$\begin{aligned} f_c &= \mu_c \cdot |f_\perp| \\ \tau_c &= \mu_{c\phi} \cdot |f_\perp| \quad . \end{aligned} \tag{B.1}$$

In case of contact, the normal (to $\mathcal{K}_0$'s x-y-plane) component $f_\perp$ (i.e., the projection onto $\mathcal{K}_0$'s z-axis) of the contact force $\vec{f}$ is always calculated from the normal component $\mathrm{r}_\perp$ of the distance vector of the origins of the contact coordinate systems $\mathcal{K}_0$ and $\mathcal{K}_1$, and the time derivative $\mathrm{v}_\perp$ of $\mathrm{r}_\perp$, that is, the projection of the relative velocity onto $\mathcal{K}_0$'s z-axis:

$$f_\perp = \kappa_\perp \cdot \mathrm{r}_\perp + \rho_\perp^{01} \cdot \mathrm{v}_\perp - \rho_\perp^{11} \cdot \mathrm{r}_\perp \cdot \mathrm{v}_\perp \quad . \tag{B.2}$$

Here, $\kappa_\perp$ is the normal stiffness of the contact interaction, and $\rho_\perp^{01}$ and $\rho_\perp^{11}$ are the damping coefficient and the non-linear damping strength, respectively, of normal deformation rates. If, according to the decision scheme based on Eq. (B.1) as shown in Fig. B.1b, the contact has switched to the 'slip' state, the corresponding tangential (projection onto $\mathcal{K}_0$'s x-y-plane of contact) force vector $\boldsymbol{f}_\parallel$ of dynamic friction and the normal torque component $\tau_\perp$ modelling torsional friction are determined by (i) the normal force component $f_\perp$, (ii) the projection vector $\vec{v}_\parallel$ onto the contact plane (i.e., components tangential to it) of the difference $\vec{v} = \vec{v}_{\mathcal{K}_1} - \vec{v}_{\mathcal{K}_0}$ between the linear velocity vectors of the origins of $\mathcal{K}_1$ ($\vec{v}_{\mathcal{K}_1} = \dot{\vec{r}}_{\mathcal{K}_1}$) and $\mathcal{K}_0$ ($\vec{v}_{\mathcal{K}_0} = \dot{\vec{r}}_{\mathcal{K}_0}$), and (iii) the projection $\xi_\perp$ onto $\mathcal{K}_0$'s z-axis of the angular velocity ξ of $\mathcal{K}_1$ relative to $\mathcal{K}_0$, respectively. In that, the (tangential) dynamic friction force can be modelled as consisting of both a contribution linearly proportional to velocity and a *Coulomb* component, whereas the torsional torque is solely made of a *Coulomb* component:

$$\begin{aligned} \boldsymbol{f}_\parallel &= \sigma \cdot \vec{v}_\parallel + \mu \cdot |f_\perp| \cdot \frac{\vec{v}_\parallel}{|\vec{v}_\parallel|} \\ \tau_\perp &= \mu_\phi \cdot |f_\perp| \cdot \frac{\xi_\perp}{|\xi_\perp|} \quad . \end{aligned} \tag{B.3}$$

Table B.1: The model used here has in total eight foot-ground contact points defined (four on each foot), with the positions of the origins of the reference coordinate systems $\mathcal{K}_0^{\text{a-d,l/r}}$ given below. All contacts use the same set of parameters (listed below) and produce forces w.r.t the 'from' body's coordinate frame $\mathcal{K}_1^{\text{w}}$ that is fixed to the origin of the world.

$\kappa_\perp \left[\frac{\text{N}}{\text{m}}\right]$	$\rho_\perp^{01}[\,]$	$\rho_\perp^{11} \left[\frac{\text{N·s}}{\text{m}^2}\right]$	$\sigma[\,]$	$\mu\ [\,]$	$\text{v}_c \left[\frac{\text{m}}{\text{s}}\right]$	$\mu_\phi[\,]$
100000.0	100.0	100000.0	0.0	0.7	0.001	0.0
$\xi_c \left[\frac{\text{rad}}{\text{s}}\right]$	$\kappa_\parallel \left[\frac{\text{N}}{\text{m}}\right]$	$\rho_\parallel \left[\frac{\text{N·s}}{\text{m}}\right]$	$\mu_c[\,]$	$\kappa_\phi \left[\frac{\text{N·m}}{\text{rad}}\right]$	$\rho_\phi \left[\frac{\text{N·m}}{\text{rad}}\right]$	$\mu_{c\phi}$ [m]
99.0	50000.0	50.0	0.8	200.0	20.0	99.0

$\mathcal{K}_0$	x [m]	y [m]	z [m]
$\mathcal{K}_0^{\text{a,l/r}}$	−0.1085	−0.0391	−0.0361
$\mathcal{K}_0^{\text{b,l/r}}$	−0.1085	0.0391	−0.0361
$\mathcal{K}_0^{\text{c,l/r}}$	0.0678	−0.0391	−0.0361
$\mathcal{K}_0^{\text{d,l/r}}$	0.0678	0.0391	−0.0361

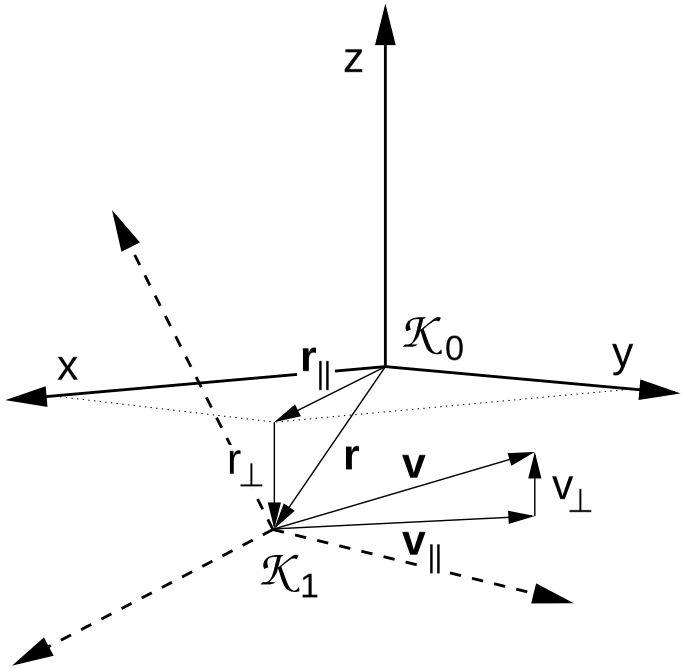
Here, μ and μ_ϕ are the coefficients of dynamic friction, and σ is the frictional damping coefficient. If, in contrast, the contact has switched to the 'stick' state, the tangential force $\boldsymbol{f}_\parallel$ always acts to drive $\mathcal{K}_1$—with the 'to'-body attached—back to the stick contact point (origin of $\overline{\mathcal{K}}_1$)—with the 'from'-body attached—, which is fixed to $\mathcal{K}_0$'s x-y-plane of contact in the 'stick' state, at the position where $\mathcal{K}_1$'s origin projected onto the contact plane at the instant when the 'stick' event occurred. Then, $\boldsymbol{f}_\parallel$ is determined by the displacement $\vec{r}_\parallel - \overline{\vec{r}}_\parallel$ of the projection vector $\vec{r}_\parallel$ of the contact point ($\mathcal{K}_1$'s origin) onto the contact plane from the respective projection vector $\overline{\vec{r}}_\parallel$ of the stick coordinate system $\overline{\mathcal{K}}_1$ (Fig. B.2), which is fixed in the contact plane, and the vector of the tangential linear velocity vector $\overline{\vec{v}}_\parallel = \dot{\vec{r}}_\parallel - \dot{\overline{\vec{r}}}_\parallel$ (note: *different* from $\vec{v}_\parallel$ in (ii) above) of the contact point within the contact plane:

$$\boldsymbol{f}_\parallel = \kappa_\parallel \cdot (\vec{r}_\parallel - \overline{\vec{r}}_\parallel) + \rho_\parallel \cdot \overline{\vec{v}}_\parallel \quad . \tag{B.4}$$

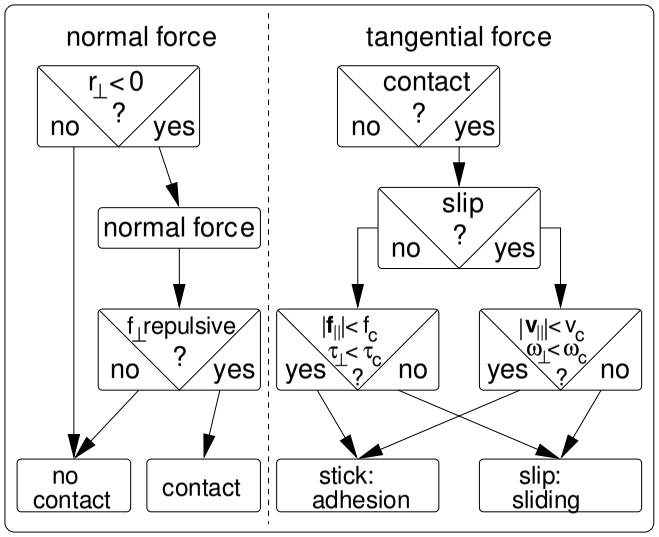
Here, $\kappa_\parallel$ is the tangential stiffness of the contact interaction, and $\rho_\parallel$ is the damping coefficient of the tangential deformation rate. Moreover, if $\mathcal{K}_1$ rotates in the 'stick' state relative to $\overline{\mathcal{K}}_1$, a restoring torsional torque around the axis normal to the contact plane ($\overline{\mathcal{K}}_1$'s $\bar{\text{z}}$-axis and $\mathcal{K}_0$'s z-axis aligning) acts between $\mathcal{K}_1$ and $\mathcal{K}_0$:

$$\tau_\perp = \kappa_\phi \cdot \Delta\xi + \rho_\phi \cdot \xi_\perp \quad . \tag{B.5}$$

Here, $\xi_\perp$ is the component perpendicular to the contact plane of the angular velocity \$\xi$ of $\mathcal{K}_1$ relative to $\mathcal{K}_0$, ρ_ϕ the corresponding angular damping coefficient, and κ_ϕ the angular stiffness of torsion. The symbol $\Delta\xi = \xi - \overline{\xi}$ is the torsional angular excursion of $\mathcal{K}_1$ (relative to $\mathcal{K}_0$) from its (torsion) angle $\overline{\xi} = \overline{\phi} + \overline{\psi}$ determined at the instant of the 'stick' event, that is, $\mathcal{K}_1$'s instantaneous torsional excursion relative to $\mathcal{K}_1$'s orientation $\overline{\mathcal{K}}_1$ as fixed at the 'stick' event. The calculation of the instantaneous torsion angle $\xi = \phi + \psi$ is illustrated in Fig. B.2. The used parameters for this contact model and the position of the contact points on the model's foot bodies are listed in Table B.1.



(a) The symbol definitions for the contact situation. The coordinate system $\mathcal{K}_0$ (fixed to the 'from'-body) represents the contact plane, and the origin of $\mathcal{K}_1$ (fixed to the 'to'-body) the potentially contacting point. $\vec{r}$ and $\vec{v}$ symbolise position and velocity vectors, respectively. Indices '$\parallel$' and '$\perp$' symbolise a vector projection onto $\mathcal{K}_0$'s x-y-plane and perpendicular to it (onto its z-axis), respectively. Here, the 'from'- and the 'to'-body are indeed in contact with each other, as the position of the $\mathcal{K}_1$ origin is at $\mathsf{z} < 0$ measured in the $\mathcal{K}_0$ system, i.e., below $\mathcal{K}_0$'s x-y-plane intersecting at $\mathsf{z} = 0$ (the contact plane). According to the principle of *actio = reactio*, the contact force vector $\boldsymbol{f}$ (not shown here) is calculated as acting on the 'from'-body (with $\mathcal{K}_0$) at the position of the point—the origin of $\mathcal{K}_1$—that contacts the plane, and the inverted force vector $(-\boldsymbol{f})$ is set to act on the 'to'-body (with $\mathcal{K}_1$) at the exact same position.



(b) The states of a modelled point-to-plane ($\mathcal{K}_1$–$\mathcal{K}_0$) contact force interaction according to Fig. B.1a, and the transition criteria (decision scheme) for calculating the contact force vector $\boldsymbol{f}$ (i.e., normal component $f_\perp > 0$ and tangential components $\boldsymbol{f}_\parallel$) in this 'to'-body–'from'-body interaction. The critical values of tangential (linear) and torsional (angular) velocities for slip-to-stick transitions, $v_c = 0.01\,\mathrm{m\,s^{-1}}$ and $\xi_c = \infty$ (with this, stick-to-slip disabled here), respectively, as well as, for stick-to-slip transitions, tangential force and torsional (indicated by '$\perp$') torque magnitudes, $f_c = \mu_c \cdot f_\perp$ and $\tau_c = \mu_{c\phi} \cdot f_\perp$, respectively, with the coefficient of static friction $\mu_c = 0.8$ (stick-to-slip disabled, thus, $\mu_{c\phi} > 0$ arbitrary), are parameters for modelling reversible transitions. The state of a contact interaction changes from 'stick' to 'slip' (based on f_c and τ_c), the 'slip' event, if *either* of the depicted criteria is *broken*, i.e., the transition to 'slip' occurs through a logical *or* connective of the 'stick' requirement. In contrast, the slip-to-stick transition (based on v_c and ξ_c), the 'stick' event, occurs through a logical *and* connective of 'back to stick' requirement, i.e., *both* depicted criteria of approaching low velocities must be *fulfilled*. If there is no contact (anymore), the stick-slip state of a contact interaction is initialised to 'slip'.

Figure B.1: Contact model symbol definition and state diagram

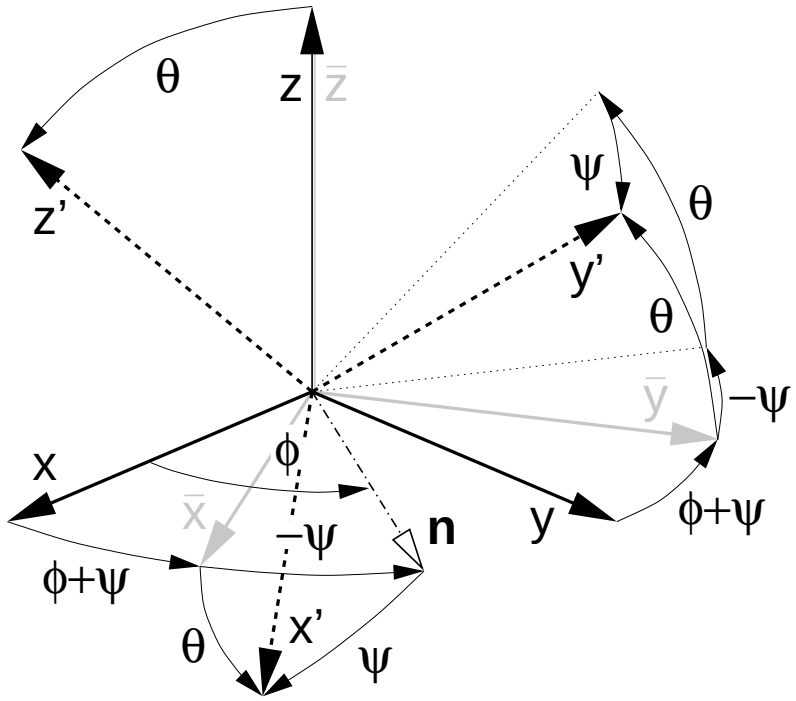
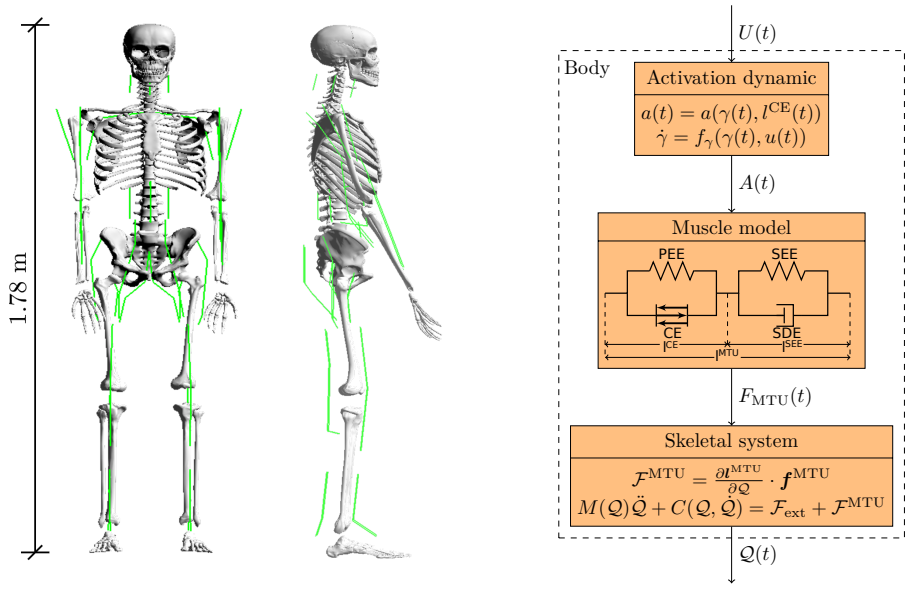


FIGURE B.2: The calculation of the torsion angle $\xi = \phi + \psi$ and the stick coordinate system $\overline{\mathcal{K}}_1$ ($\bar{\mathsf{x}}$-$\bar{\mathsf{y}}$-$\bar{\mathsf{z}}$) at the instant of a 'stick' event: separating the tilt from the torsion contribution to an overall rotation of the primed coordinate system $\mathcal{K}_1$ (dashed axes) relative to the unprimed system $\mathcal{K}_0$ (solid axes). The rotations are expressed in terms of Euler angles executed in the order ϕ, θ, ψ. The tilt is the (second) angular rotation by θ around the n-axis, the latter being the intermediate x-axis after the first rotation around the initial (unprimed) z-axis by ϕ. The last (third) rotation by ψ is then executed around the already finally fixed (primed) z'-axis. The x-y-plane of the (solid, unprimed) $\mathcal{K}_0$ system is the contact plane, in which to lie the n-axis of tilting can be assumed and used for parametrising stick-slip interaction when applying the Euler angle description of three-dimensional rotations. Regardless of the angular orientation of $\mathcal{K}_1$ relative to $\mathcal{K}_0$, the position of the origin of $\mathcal{K}_1$ in relation to the contact plane determines (a) whether both coordinate systems (bodies) are in contact at all and, therefore, mechanically interact by a force $\vec{f}$ and a torque $\boldsymbol{\tau}$, as well as (b) where in the contact plane a unique stick coordinate system $\overline{\mathcal{K}}_1$ is located for modelling a restoring visco-elastic tangential force and torsional torque in the 'stick' state. In fact, $\overline{\mathcal{K}}_1$ is fixed at the instant when the 'stick' event occurs: its origin is chosen to be located in the contact plane at the vector $\overline{\vec{r}}_{\parallel}$ of the projection of $\mathcal{K}_1$'s origin onto the contact plane—here, for simplicity of the illustration at $\mathcal{K}_0$'s origin—, the orientation of $\overline{\mathcal{K}}_1$'s $\bar{\mathsf{z}}$-axis is chosen to always align with the normal vector of the contact plane ($\mathcal{K}_0$'s z-axis), and the Euler angle sum $\overline{\xi} = \overline{\phi} + \overline{\psi}$ is calculated for determining later torsional angular excursions $\Delta\xi = \xi - \overline{\xi}$ relative to the 'stick' condition.



(a) (b)

FIGURE C.1: (a) Frontal and side view of the visualization of the DHM. The green lines show the muscle geometry. (b) Structure of the model: the motor command $\boldsymbol{u}(t) \in \mathbb{R}^{n_{\text{MTU}}}$ is fed into the model of activation dynamics (Hatze, 1977; Rockenfeller and Günther, 2018), which relates the neuronal stimulation to muscular activity $\boldsymbol{a}(t) \in \mathbb{R}^{n_{\text{MTU}}}$ that drives the MTUs (Haeufle et al., 2014a). The MTUs produce forces $\boldsymbol{f}^{\text{MTU}}(t) \in \mathbb{R}^{n_{\text{MTU}}}$ that act as joint torques by their respective moment arms $\frac{\partial \boldsymbol{l}^{\text{MTU}}}{\partial \mathcal{Q}}$. In combination with external forces, this results in a movement of the DoFs $\mathcal{Q}(t) \in \mathbb{R}^{n_{\text{DoF}}}$ of the body.

The musculoskeletal model *allmin* consists of $n_{\text{RGB}} = 15$ rigid bodies (see Table C.1). The rigid bodies are connected via 14 joints (see Table C.2) including $n_{\text{DoF}} = 20$ degrees of freedom. Each *degree of freedom* (DoF) (except for the Wrs) is controlled by an AAS being congruent with the concept of *elementary biological drives*, as described by Schmitt et al. (2019). The musculoskeletal model is actuated by $n_{\text{MTU}} = 36$ MTUs (see Table C.3-C.5 and Figure C.1a). The model is implemented in the simulation software `demoa` (`C/C++`).

C.1 Model parameters

Table C.1: List of all bodies included in the model with their mechanical properties with m: mass, r_x,r_y: radius in x and y direction, h_z: height in z direction, $\mathbf{d}_1$: distance proximal joint to the body's centre of mass and $\mathbf{d}_2$: distance centre of mass to distal joint. The spine body has an underlying curvature based on Kitazaki and Griffin (1997). The allover body dimensions are based on data describing a 50th percentile male from NASA (1978).

Body Name	m [kg]	r_x [m]	r_y [m]	h_z [m]	$\mathbf{d}_1$ $[m]$	Child	$\mathbf{d}_2$ $[m]$
Pelvis (world)	10.2516	0.1224	0.1643	0.18783	[0,0,0]	Spine	[0.000557293, 0.0000, 0.12213]
						Thigh (l/r)	$[0.0147, \pm 0.0796, -0.0657]$
Spine	33.2397	0.1224	0.1643	0.4166	[-0.00055, 0.0000, -0.2083]	Head	[0.00055, 0.0000, 0.2083]
						Uparm (l/r)	$[0.00677703, \pm 0.1816, 0.10507988]$
Head	4.8869	0.0993	0.0778	0.278194	[-0.0092, 0.0000, -0.11]	-	-
Uparm (l/r)	2.1631	0.0495	–	0.3065	[0.0000, 0.0000, 0.1456]	Forearm (l/r)	[0.0000, 0.0000, -0.1609]
Forearm (l/r)	1.3389	0.0477	–	0.2725	[0.0000, 0.0000, 0.1117]	Hand (l/r)	[0.0000, 0.0000, -0.1608]
Hand (l/r)	0.5252	0.028	0.089	0.192	[0.0000, 0.0000, 0.0574]	-	-
Thigh (l/r)	8.1719	0.0947	–	0.4347	$[0.0000, \mp 0.0188, 0.1782]$	Shank (l/r)	[0.0000, 0.0000, -0.2565]
Shank (l/r)	3.3541	0.0597	–	0.4239	$[0.0000, \mp 0.0059, 0.1865]$	Foot (l/r)	[0.0000, 0.0000, -0.2374]
Foot (l/r) $*$	1.0172	0.0398	–	0.272	[-0.0656, 0.0000, 0.0402]	-	-

Table C.2: List of all joints included in the model.

Name	Type	Movement	RoM [°]
Lumbar spine	Universal	left/right	$[-30 \dots 30]$
Lumbar spine	Universal	flexion/extension	$[0 \dots 30]$
Cervival spine	Universal	left/right	$[-30 \dots 30]$
Cervival spine	Universal	flexion/extension	$[-30 \dots 30]$
Shoulder (Right)	Universal	abduction/adduction	$[-10 \dots 60]$
Shoulder (Right)	Universal	flexion/extension	$[-100 \dots 10]$
Ellbow (Right)	Revolute	flexion/extension	$[-120 \dots 10]$
Wrist (Right)	Revolute	flexion/extension	$[0 \dots 0]$
Shoulder (Left)	Universal	abduction/adduction	$[-10 \dots 60]$
Shoulder (Left)	Universal	flexion/extension	$[-100 \dots 10]$
Ellbow (Left)	Revolute	flexion/extension	$[-120 \dots 10]$
Wrist (Left)	Revolute	flexion/extension	$[0 \dots 0]$
Hip (Right)	Universal	flexion/extension	$[-120 \cdots -10]$
Hip (Right)	Universal	abduction/adduction	$[-10 \dots 70]$
Knee (Right)	Revolute	flexion/extension	$[-1 \dots 120]$
Ankle (Right)	Revolute	flexion/extension	$[-20 \dots 40]$
Hip (Left)	Universal	flexion/extension	$[-120 \dots 10]$
Hip (Left)	Universal	abduction/adduction	$[-10 \dots 70]$
Knee (Left)	Revolute	flexion/extension	$[-1 \dots 120]$
Ankle (Left)	Revolute	flexion/extension	$[-20 \dots 40]$

Table C.3: Muscle routing parameters: Origin R_O, Deflection Point 1 R_{DF1} and 2 R_{DF2} and Insertion R_I relative to their parent body. All numbers in this table are rounded to three decimal digits. Muscle names: Eb Fx, Eb Ex, An Ex, An Fx, Hp Ab, Hp Ad, Hp Fx, Hp Ex, Ce lEx, Ce vEx, Ce vEx, Ce vFx, Kn Ex, Kn Ex, Lu lEx, Lu vEx, Lu vEx, Lu vFx, Sh Ex, Sh Fx, Sh Ab, Sh Ad. The muscle routing parameters have been optimised to allow a reasonable RoM of the respective joints.

Name	R_O [m]			Parent	R_{DF1} [m]			Parent	R_{DF2} [m]			Parent	R_I [m]			Parent
Lu lEx	−0.028	0.000	0.108	Pelvis	−0.040	0.000	0.110	Pelvis	−0.042	0.000	−0.131	Spine	−0.032	0.000	−0.129	Spine
Lu lFx	0.018	0.000	0.101	Pelvis	0.088	0.000	0.089	Pelvis	0.069	0.000	−0.106	Spine	0.009	0.000	−0.120	Spine
Lu vEx	−0.005	0.050	0.104	Pelvis	−0.005	0.050	0.104	Pelvis	−0.011	0.050	−0.124	Spine	−0.011	0.050	−0.124	Spine
Lu vFx	−0.005	−0.050	0.104	Pelvis	−0.005	−0.050	0.104	Pelvis	−0.011	−0.050	−0.124	Spine	−0.011	−0.050	−0.124	Spine
Ce lEx	−0.054	0.000	0.199	Spine	−0.054	0.000	0.199	Spine	−0.056	0.000	−0.070	Head	−0.056	0.000	−0.070	Head
Ce lFx	0.043	0.000	0.175	Spine	0.043	0.000	0.175	Spine	0.044	0.000	−0.080	Head	0.044	0.000	−0.080	Head
Ce vEx	−0.006	0.050	0.187	Spine	−0.006	0.050	0.187	Spine	−0.006	0.050	−0.075	Head	−0.006	0.050	−0.075	Head
Ce vFx	−0.006	−0.050	0.187	Spine	−0.006	−0.050	0.187	Spine	−0.006	−0.050	−0.075	Head	−0.006	−0.050	−0.075	Head
Hp Ex (l/r)	−0.075	±0.080	0.025	Pelvis	−0.075	±0.090	−0.095	Pelvis	−0.075	∓0.019	0.121	Thigh	−0.020	∓0.009	0.031	Thigh
Hp Fx (l/r)	0.065	±0.040	0.101	Pelvis	0.075	±0.040	0.021	Pelvis	0.015	∓0.019	0.101	Thigh	0.015	∓0.019	0.020	Thigh
Hp Ab (l/r)	−0.025	±0.120	0.050	Pelvis	0.000	±0.152	−0.030	Pelvis	−0.030	±0.040	0.035	Thigh	−0.020	±0.030	0.005	Thigh
Hp Ad (l/r)	0.000	0.000	0.000	Pelvis	−0.010	±0.010	−0.100	Pelvis	−0.005	∓0.035	0.090	Thigh	0.000	∓0.020	0.010	Thigh
Kn Fx (l/r)	−0.050	0.000	0.000	Thigh	−0.050	0.000	−0.108	Thigh	−0.059	0.000	0.106	Shank	−0.030	0.000	0.100	Shank
Kn Ex (l/r)	0.040	0.000	0.000	Thigh	0.030	0.000	0.253	Thigh	0.030	0.000	0.050	Shank	0.030	0.000	0.050	Shank
An Ex (l/r)	−0.050	0.000	−0.025	Shank	−0.050	0.000	−0.175	Shank	−0.125	0.000	0.050	Foot	−0.125	0.000	0.050	Foot
An Fx (l/r)	0.030	0.000	−0.025	Shank	0.030	0.000	−0.175	Shank	0.030	0.000	0.050	Foot	0.030	0.000	0.050	Foot
Sh Ex (l/r)	−0.069	±0.182	0.113	Spine	−0.050	0.000	0.125	Uparm	−0.017	0.000	0.000	Uparm	−0.017	0.000	0.000	Uparm
Sh Fx (l/r)	0.022	±0.182	0.139	Spine	0.022	±0.182	0.139	Spine	0.017	0.000	0.000	Uparm	0.017	0.000	0.000	Uparm
Sh Ab (l/r)	−0.026	±0.242	0.135	Spine	−0.026	±0.242	0.135	Spine	0.000	±0.017	0.000	Uparm	0.000	±0.017	0.000	Uparm
Sh Ad (l/r)	−0.024	0.000	0.126	Spine	0.007	±0.125	0.103	Spine	0.000	∓0.040	0.125	Uparm	0.000	∓0.017	0.000	Uparm
Eb Fx (l/r)	0.025	0.000	0.000	Uparm	0.030	0.000	−0.050	Uparm	0.030	0.000	0.014	Forearm	0.024	0.000	−0.100	Forearm
Eb Ex (l/r)	−0.025	0.000	0.000	Uparm	−0.049	0.000	−0.160	Uparm	−0.048	0.000	0.100	Forearm	−0.024	0.000	0.000	Forearm

Table C.4: Muscle-specific actuation parameters, with $F^{\max}$: maximum isometric force, $l^{\mathrm{CE,opt}}$: optimal length of the CE, ΔW^{asc}: width of normalized bell curve in ascending branch of the force-length relationship, $l^{\mathrm{SEE},0}$ rest length of the SEE, $l^{\mathrm{CE,init}}$: initial length of the CE. Muscle names: *elbow* (Eb) Fx, *elbow* (Eb) Ex, *ankle* (An) Fx, *ankle* (An) Fx, *hip* (Hp) Ab, *hip* (Hp) Ad, *hip* (Hp) Fx, *hip* (Hp) Ex, *cervical joint* (Ce) lFx, *cervical joint* (Ce) lEx, *cervical joint* (Ce) vFx, *cervical joint* (Ce) vEx, *knee* (Kn) Fx, *knee* (Kn) Ex, *lumbar joint* (Lu) lFx, *lumbar joint* (Lu) lEx, *lumbar joint* (Lu) vFx, *lumbar joint* (Lu) vEx, *shoulder* (Sh) Ab, *shoulder* (Sh) Ad, *shoulder* (Sh) Fx, *shoulder* (Sh) Ex. The muscle-specific parameters are based on literature values of single muscles and have been further tuned for a functionality of the overall model, accounting for the consolidation of multiple muscles into single MTUs (Bayer et al., 2017; Kistemaker et al., 2006; Günther, 1997).

	$F^{\max}$ [N]	$l^{\mathrm{CE,opt}}$ [m]	ΔW^{asc}	$l^{\mathrm{SEE},0}$ [m]
Eb Fx	1420.0	0.1885	1.0	0.1845
Eb Ex	1550.0	0.171	0.525	0.18
An Fx	3000.0	0.15	1.0	0.133
An Ex	3000.0	0.13	1.0	0.115
Hp Ab	2000.0	0.18	1.0	0.121
Hp Ad	2000.0	0.204	0.75	0.136
Hp Fx	5000.0	0.195	1.0	0.135
Hp Ex	5000.0	0.192	1.0	0.191
Ce lFx	5000.0	0.07	1.5	0.01
Ce vEx	5000.0	0.05	1.5	0.01
Ce vFx	5000.0	0.046	1.5	0.01
Ce lFx	5000.0	0.062	1.5	0.01
Kn Fx	6000.0	0.258	0.525	0.112
Kn Ex	6000.0	0.264	1.0	0.28
Lu lFx	15000.0	0.2	1.5	0.11
Lu vEx	15000.0	0.09	1.5	0.02
Lu vFx	15000.0	0.09	1.5	0.02
Lu lFx	15000.0	0.075	1.5	0.04
Sh Ab	6000.0	0.12	1.0	0.08
Sh Ad	6000.0	0.225	1.0	0.12
Sh Fx	10000.0	0.1	1.0	0.073
Sh Ex	6000.0	0.165	1.0	0.105

Table C.5: Muscle non-specific actuation parameters for the muscles and the activation dynamics.

	Parameter	Unit	Value	Source	Description
CE	ΔW^{des}	[ ]	0.45	similar to Bayer et al. (2017); Kistemaker et al. (2006)	width of normalized bell curve in descending branch, adapted to match observed force-length curves
	$\nu^{\mathrm{CE,des}}$	[ ]	1.5	Mörl et al. (2012)	exponent for descending branch
	$\nu^{\mathrm{CE,asc}}$	[ ]	3.0	Mörl et al. (2012)	exponent for ascending branch
	$A^{\mathrm{rel,0}}$	[ ]	0.2	Günther (1997)	parameter for contraction dynamics: maximum value of A^{rel}
	$B^{\mathrm{rel,0}}$	[1/s]	2.0	Günther (1997)	parameter for contraction dynamics: maximum value of B^{rel}
	$\mathcal{S}^{\mathrm{ecc}}$	[ ]	2.0	van Soest and Bobbert (1993)	relation between $F(v)$ slopes at $v^{\mathrm{CE}} = 0$
	$\mathcal{F}^{\mathrm{ecc}}$	[ ]	1.5	van Soest and Bobbert (1993)	factor by which the force can exceed F^{isom} for large eccentric velocities
PEE	$\mathcal{L}^{\mathrm{PEE,0}}$	[ ]	0.95	Günther (1997)	rest length of PEE normalized to optimal length of CE
	ν^{PEE}	[ ]	2.5	Mörl et al. (2012)	exponent of F^{PEE}
	$\mathcal{F}^{\mathrm{PEE}}$	[ ]	2.0	Mörl et al. (2012)	force of PEE if l^{CE} is stretched to ΔW^{des}
SDE	D^{SDE}	[ ]	0.3	Mörl et al. (2012)	dimensionless factor to scale $d^{\mathrm{SDE,max}}$
	R^{SDE}	[ ]	0.01	Mörl et al. (2012)	minimum value of d^{SDE} (at $F^{MTU} = 0$), normalized to $d^{\mathrm{SDE,max}}$
SEE	$\Delta U^{\mathrm{SEE,nll}}$	[ ]	0.0425	Mörl et al. (2012)	relative stretch at non-linear linear transition
	$\Delta U^{\mathrm{SEE,l}}$	[ ]	0.017	Mörl et al. (2012)	relative additional stretch in the linear part providing a force increase of $\Delta F^{\mathrm{SEE,0}}$
	$\Delta F^{\mathrm{SEE,0}}$	[N]	$0.4\,F^{\mathrm{max}}$		both force at the transition and force increase in the linear part
activation dynamics	M_H	[1/s]	11.3	Kistemaker et al. (2006)	time constant for the activation dynamics
	γ_c	[mol/l]	1.37e-4	Kistemaker et al. (2006)	constant for the activation dynamics
	ρ_0	[l/mol]	5.27e4	Kistemaker et al. (2006)	constant for the activation dynamics
	a_0	[ ]	0.005	Günther (1997)	resting active state for all activated muscle fibers
	ν	[ ]	3	Kistemaker et al. (2006)	constant for the activation dynamics

D.1 Simulation software and solver

To perform the here presented simulations, the C/C++ simulation environment *demoa* is used (Henze, 2002; Mörl et al., 2012; Rupp et al., 2015). The simulation software *demoa* uses homogeneous matrix representations (Hartenberg and Denavit, 1955) to algorithmically set-up the mechanical equations of motion (2.33), as described by (Legnani et al., 1996b,a), using the C/C++ library *eigen* (Guennebaud et al., 2010). The integration method used within *demoa* for the mechanical equations of motion (2.33), the activation dynamics (2.38), and the MTU contraction dynamics is based on the Shampine-Gordon algorithm '*de*' (Shampine and Gordon, 1975), which has been slightly modified (Henze, 2002) to allow event handling (root finding), '*derf*'. The integration parameters are the same for all simulation studies from Pt. III. The maximum simulation step size is set to 0.001 s with relative and absolute error boundaries of 10^{-6}.

The integration in the I part of the PID-controllers is performed using Euler's method (first order Runge-Kutta) and the differentiation in the $\boldsymbol{\tau}$-PID-controller in Sec. 6 is performed by discrete differentiation based on the integration step time.

D.2 `demoa` variables of the homogeneous rigid body matrices

In this section the homogeneous representation matrices $\boldsymbol{G}$, $\hat{\boldsymbol{\Xi}}$ and $\hat{\boldsymbol{V}}$ of the positions, twists and velocities of rigid bodies, as described in Sec. 2.1, are related to the notation of the respective variable names in `demoa`. Additionally it is sketched how the input parameters of `demoa` are used for the calculation of these matrices. The syntax for each specific block is well described in the documentation of `demoa`. The input parameters of `demoa` that defines the anatomy of the musculo-skeletal system are provided by the user in the form of a structured list of parametrised text-blocks in a `.dsim`-file of the single model-elements.

The model-elements of interest for this section are those of the bodies and those of the joints. A body is parametrised by its inertia properties and its geometry. The geometry is specified by defining coordinate frames that are located at the joints that connect the body to its environment. Other coordinate frames are attached to the rigid body's COM. As described in Sec. 2.1.3, the relative configuration, i.e. rotation and translation, of two frames can be described by the homogeneous representation $\boldsymbol{G} \in SE(3) \subset \mathbb{R}^{4\times4}$ of rigid body transformations (2.11).

- Rigid body transformation matrices $\boldsymbol{G}$ as in (2.15) correspond to `M`-matrices in `demoa`.

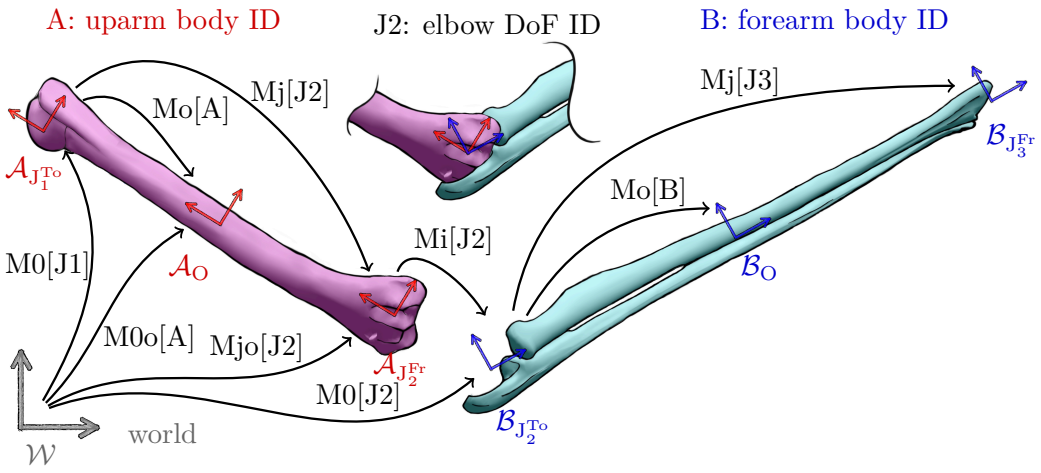


Figure D.1: Coordinate frame transformations at a joint in `demoa`. Each joint has a 'To'-body and a 'From'-body at which the joint coordinate frames are attached, e.g. $\mathcal{A}_{\mathrm{JFr}}$ and $\mathcal{B}_{\mathrm{JTo}}$. The rigid body transformation $\boldsymbol{G}^{\mathcal{A}_{\mathrm{JFr}}\mathcal{B}_{\mathrm{JTo}}}$ describes the joint DoFs and is stored in the `Mi[J]`-matrix. The matrix `Mj[J]` describes the distance from the previous joint to the current joint, the matrix `Mo[A]` the distance to the body's COM. The matrices `M0[J]`, `M0o[J]` and `Mj0[J]` are `Mi[J]`, `Mo[J]` and `Mj[J]` transformed to the inertial frame $\mathcal{W}$. By following the composition rule (2.15) and starting at the first body, the kinematic chain can be successively build up.

There are various useful `M`-matrices predefined in `demoa`. In Fig. D.1 an exemplary elbow joint is shown with arrows displaying the transformations of each those `M`-matrices.

The matrices `Mo` and `Mj` are directly obtained from the body model-block parameters in the provided `.dsim`-file.

The `Mi` matrices are obtained as follows: The input parameters of a joint define the movement of the respective joint DoFs. This information is stored in the `Li`-matrix. This matrix thereby fixes and defines the screw axis of a joint and thereby restricts its DoFs. As such `Li` is referred to as a *restriction matrix* in `demoa`.

- Rigid body twists $\hat{\boldsymbol{\Xi}}$ as in (2.20) correspond to `L`-matrices in `demoa`.

According to (2.21), a rigid body twist can be used for the calculation of $\boldsymbol{G}$ by a matrix exponential utilising *Rodrigues' formula* (2.6) (Legnani et al., 1996b,a; Murray et al., 1994; Lynch and Park, 2017). In exactly this way the `demoa`-matrix `Mi` is obtained.

From Sec. 2.1.5 it is known that the spatial velocity $\hat{\boldsymbol{V}}$ of a rigid body corresponds to a twist.

- Rigid body spatial velocities $\hat{\boldsymbol{V}}$ as in (2.22) correspond to `W`-matrices in `demoa`.

According to (2.28), in `demoa` the velocity is directly obtained by multiplying `Li` by the velocity `qd` of the DoF.

Implementation of the hierarchical control architecture E

To execute a simulation with the here presented hierarchical control architecture, it is implemented in the simulation framework `demoa` (CBB in-house code). The biophysical sensor information and other system knowledge that is needed within the control architecture, such as the actual joint angles or the moment arms of the MTUs are therefore obtained, or calculated during run-time, from the system's state that is integrated over time in `demoa`.

The object-oriented implementation allows to execute different controllers in the different control spaces of joint angles, joint torques and limb positions in parallel. With this it is possible, for example, to simultaneously control the movement of the joint angles of a leg and the position of a hand of a model.

In this chapter, the implementation strategy of the control architecture is described by listing the respective control parameters and quickly outlining the initialization routine and the functions that are called during simulation runtime in `demoa`.

E.1 Object-oriented design and controller parametrisation

To enable a wide range of freedom for control in terms of control spaces, an object-oriented implementation design is used, where multiple controllers can be generated to fulfil individual control goals in parallel. Each controller thereby consists of the full hierarchy, i.e. it has it's own conceptional, transformational and structural layer. The layers are implemented to be individual `objects` of the respective `Layer-class`. Therefore the three `classes`, the `ConceptionalLayer`, the the `TransformationalLayer` and the the `StructuralLayer` are implemented. The control space of the conceptional layer has to be one that is specified by a `control_architecture`-byte:

$$\texttt{int control\_architecture} = 0\texttt{b}b_f b_\chi b_\tau b_\theta b_\lambda, \tag{E.1}$$

This `control_architecture`-byte is defined in `usrinclude.h` and specifies which different control spaces are activated in the model. It's binary form (E.1), with $b_i \in \{0, 1\}$, i.e. the i-th bit enables the i-th controller; of CE-lengths (λ), joint angles (θ), joint torques (τ), limb positions (χ) or forces (f)[1], for example,

$$\texttt{int control\_architecture} = 0\texttt{b}01110, \tag{E.2}$$

enables the θ-, τ- and χ-controllers in parallel. Any other combination is of course possible.

[1] Note, that the force controller as described in section 3.3.4 and specified by b_f is not fully integrated in the control architecture's implementation

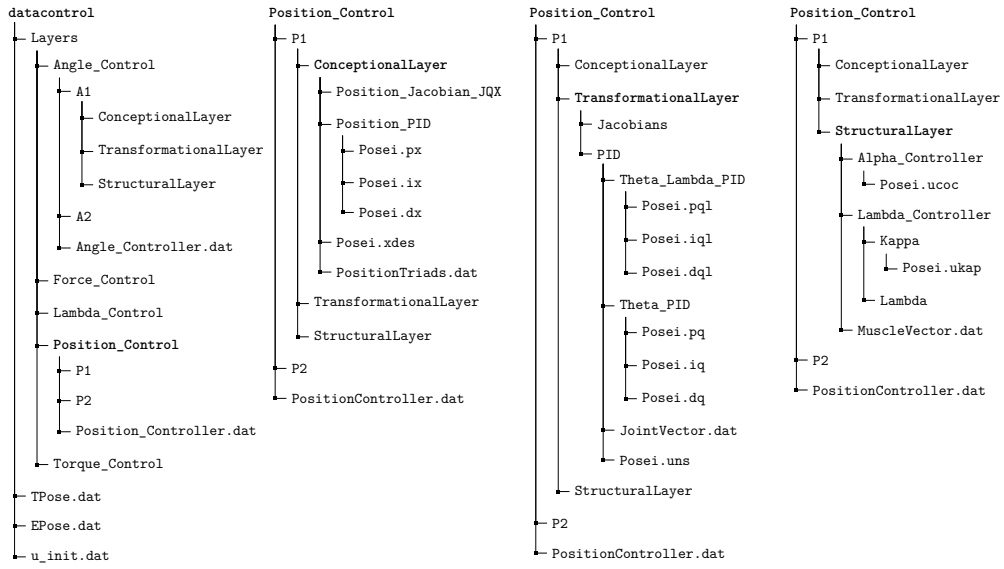


Figure E.1: Folder structure of the `/datacontrol`-folder (left). In the three folder structures on the right an exemplary folder structure for a position controller with the name `P1` is shown.

In the control parameter folder `/datacontrol` within the model folder the specified controllers are parametrized (see figure E.1 for folder structure). The general controller parameters of initial stimulation is specified in the file `u_init.dat` and the different control states in `TPose.dat` or `EPose.dat`, depending on the variable `EventFlag` variable in `usrinclude.h`. When `EventFlag=0` a time-based control is used that switches the poses automatically according to the times and poses defined in `TPose.dat`. When `EventFlag=0` an event-based control is used and to switch the poses defined in `EPose.dat` a custom code in `uEventControl.cpp` is expected.

In the subfolders in `/datacontrol/Layers`, the different control spaces are configured. Each control space has its own folder, e.g. `/datacontrol/Layers/Angle_Control` and in each of those folders a `.dat`-file that specifies the names and thereby the total number of controllers that are defined. For example, a `AngleController.dat`-file in `/datacontrol/Layers/Angle_Control/` like

```
1 //Specify Angle Controller Names Here, This comment Line is mandatory
2 A1
3 A2
```

specifies the two angle controllers with the names `A1` and `A2`, respectively. For each of such specified controllers, a subfolder with the same name must be present that contains the parameters of the individual layers of the specified controllers. The layer parameters have to be inside another set of subfolders with the names `/ConceptionalLayer`, `/TransformationalLayer` and `/StructuralLayer`. In figure E.1 the folder structure is displayed and in the follwing subsections the parameters for the individual layers are described.

E.1.1 The conceptional layer parameters: `/ConceptionalLayer`

Each conceptional layer has by choice one of the control spaces of: i) a set joint angles, ii) a set of joint torques or iii) the configuration in 3D-space of a specified coordinate space.

In general a PID-controller is implemented for this control space and a Jacobian matrix is used for the transformation to the transformational layer, as described in theory in Sec. 3.2. For its parametrisation, therefore three information must be provided by the user: i) the control space must be specified, e.g. which joints are to be controlled, ii) the desired states for this control space and iii) the PID parameters for the controller. The Jacobian matrices for transformations are usually calculated automatically during runtime, optionally they can be read in for each pose, though. The respective flag has to be set in `usrinclude.h`.

E.1.1.1 Angle controller parameters

A conceptional angle controller, as described in theory in section 3.3.1 takes the trivial form of providing the postural plan $\boldsymbol{\theta}^{\mathrm{des}}(t)$ directly. It therefore has in fact no PID-controller and no Jacobian transformation implemented. Still, the control space has to be defined in the file `JointVector.dat` in the exemplary form:

```
1 // Comment Line
2 Joint_Shoulder->Uparm, 0,
3 Joint_Shoulder->Uparm, 1,
4 Joint_Uparm->Forearm, 0,
```

In this example the 2 DoF shoulder `joint` of type `Universal` is called `Joint_Shoulder->Uparm` in the `.dsim`-file of the `model` and the first and second lines specify the first and second joint DoFs by the 0 and 1, respectively. Additionally the desired values for the such defined control space must be provided for each pose, e.g. by `Pose1.qdes`:

```
1 // Comment Line
2 0.0,
3 777,
4 90.0,
```

This sets the desired values for the shoulder *abduction-adduction* (AA) and elbow *flexion-extension* (FE) to 0.0 and to 777 for shoulder FE. The 777 hereby sets the desired value exactly the to the current value and effectively disables the control of this control space variable. In the example here, this means that no MTU stimulations are generated to actively manipulate the shoulder FE movement. The demanded movement thus is, to bring the elbow to 90° while holding the arm proximal to the trunk. The arm is allowed to freely swing within the sagittal plane.

E.1.1.2 Torque controller parameters

The conceptional torque controller parameters, specified within it's sub-folder, for example `/TorqueController/T1/ConceptionalLayer` for a torque controller named `T1`, are the vector of controlled joint DoFs in `TorqueJointVector.dat`, the vectors of desired torques in the pose-files with the suffix `.tdes` and the PID-parameters in the subfolder `/Torque_PID`. The vectors of controlled and desired DoFs are set in an exactly same manner as for the angle controller above. The PID-subfolder contains for each pose a `.pt`, a `.it` and a `.dt` file, containing the P, I and D control parameter matrices. These control matrices are usually chosen as diagonal matrices and have a form similar to the following example of a `.pt`-file for a similar joint vector as given in the angle controller parameter section above:

```
1 // Comment Line,
2 2.000000, 0.000000, 0.000000,
3 0.000000, 0.000000, 0.000000,
4 0.000000, 0.000000, 0.750000,
```

The Jacobian matrices for transformations are usually calculated automatically during runtime, optionally they can be specified to be read in for each pose from the sub-folder `/Position_Jacobian_JQX`. The respective flag has to be set in `usrinclude.h`.

E.1.1.3 Position controller parameters

The specification of the position controller parameters follows the same approach, by setting the control space and desired values, as well as PID-parameters for the controller. The PID-matrices are all 6×6 as the positional space has three translational DoFs and three rotational. By setting a specific diagonal matrix entry to zero, the control of this coordinate direction (or rotation) can be switched off. The control space definition in the `PositionTriads.dat`-file occurs slightly different compared to the angle or torque controllers, as for the position controller a `Triad`-name must be specified together with a 'relative' `Triad`-name. These two triads should specify the start and the end of a kinematic sub-chain of the model, e.g. from the shoulder to the hand as in this example of a `PositionTriads.dat`-file:

```
1 Shoulder_Joint_Uparm
2 Ctrl_Trd_Hand_l
```

The first entry describes the name of the 'relative'-`Triad` (the start of the kinematic sub-chain) and the second of the 'control'-`Triad` (attached to the end `Body` of the kinematic sub-chain). The desired configuration of the 'control'-`Triad` is specified, for each pose, in a `.xdes` file. The only content in such a file is a `Triad`-name that exists in the model and of which the position is desired to be achieved.

The Jacobian matrices for transformations are usually calculated automatically during runtime, optionally they can be specified to be read in for each pose from the sub-folder `Torque_Jacobian_JQT`. The respective flag has to be set in `usrinclude.h`.

E.1.2 The transformational layer parameters: `/TransformationalLayer`

No matter which conceptional control space is chosen, the transformational layer has always the same form and must thus be specified in the same way for each controller. The transformational layer's control parameters are specified in the subfolder `/TransformationalLayer` within the controller's folder. As in this layer two PID-controllers on the control space of joint angles are active, the joint angle DoFs have to be provided as a vector in the `JointVector.dat`-file. In this file a list of `joint`-names with a DoF number must be given, exactly as described in Sec E.1.1.1 for the conceptional angle control parameters. Note, that the transformational layer's joint vector must be chosen in accordance with the conceptional layer's output joint vector. That is, for angle and torque controllers, the joint-vectors of the conceptional and the transformational layers must contain the same DoFs (although the order does not matter). A transformational layer's joint vector of a position controller must contain all those joints that connect the 'relative' `Triad`'s body to the 'control' `Triad` 's body. To control a hand relative to it's shoulder, for example, the shoulder, elbow and wrist joints are involved and must be specified in the respective `JointVector.dat`-file of the transformational layer. The PID-parameters for the hierarchical $\theta-\lambda$ controller and for the direct θ-controller are chosen exactly as described above, e.g. for the torque PID controller in Sec. E.1.1.2. The same holds for the Jacobian matrices, when they are specified to be read in (`usrinclude.h`).

Additionally in this layer the joint-wide co-contraction parameters ζ^{θ} can be set (refer to Sec. 3.2.4 for the theory of joint-wide co-contraction). For each pose and for each specified joint, a co-contraction parameter can be specified in the respective `.uns`-file of the pose, e.g. the file contents

```
1 // Comment Line,
2 0.0,
3 0.0,
4 0.0,
```

disables the joint wide co-contraction for a controller, similar to the example from section E.1.1.1.

E.1.3 The structural layer parameters: /StructuralLayer

Within the structural layer that is parametrised in the /StructuralLayer subfolder, the low-level CE-length (λ-) controller and the reference co-contraction ζ^u are parametrised. For this it is mandatory to specify a vector of MTUs that are involved in the current controller. The file MuscleVector.dat therefore must contain a list of those MTU-names that are spanned around the joints that are specified in the transformational layer's paremtrisation. Sticking to the same example with the joint vector from section E.1.1.1, a corresponding MTU-vector may look like:

```
1 // Comment Line,
2 M_schulterstrecker
3 M_schulterbeuger
4 M_schulterabduktion
5 M_schulteraduktion
6 M_ellbogenbeuger
7 M_ellbogenstrecker
```

In the subfolder Alpha_Controller, for each specified MTU a co-contraction parameter $\zeta^u \in (0\,1]$ must be set. In the subfolder /Lambda_Controller/Kappa, the P-parameter matrix of the λ-controller is specified as a list that forms the diagonal entries of $\boldsymbol{P}_\lambda$. And in the subfolder /Lambda_Controller/Lambda, desired CE-lengths, i.e. λ, are set for each pose in the respective .ulam-files. Note, that λ only needs to be set if for the control space λ is chosen, i.e. the bit b_λ is set in the control_architecture variable in usrinclude.h.

E.2 Initialisation

In the function UserSetMKSInit() in usrsetmks.cpp, the main controller object (that is usually accessed by myMotControl) is initialised. This instance manages all individual controllers and can be used for communication with the controller's layer-objects. In it's initialisation (see Lst. E.3), the basic controller configurations of the control_architecture -byte from usrinclude.h and the Tpose.dat or the Epose.dat file in /datacontrol-folder are interpreted. Also, some of the biophysical sensor signals and system features, such as the matrix of moment arms are calculated here, globally (see Lsts. E.1, E.6 and E.2). This means that the moment arm matrix, for example, is calculated for every DoF and for every MTU that are defined in the model and the respective order of demoa is used.

With the control_architecture interpreted, the controller configurations in /datacontrol /Layers can be efficiently interpreted. Therefore the names of the controllers and their total numbers in their respective control spaces are read in and, for each control space and each controller, the respective Layer-objects are created. The controllers are hereby designed, such that they act on a subset of the DoFs of the model. This means, that they require only a part of the information, e.g., moment arms, that was calculated globally before. To obtain this information a kind of *cookie-cutter-technique* is used, using masking-permutation matrices. These masking-permutation matrices combine the two functions of a permutation and a masking (*cookie-cutting*). With such a matrix, the local DoFs of the Layer-object can be mapped and ordered onto the DoFs of the whole model. As such, the (transposed) matrix can be used to map a vector in 'global' DoF-coordinates to the 'local', Layer-space, for example to extract and to reorder a sub-matrix of moment arms used in a transformational layer from the globally known moment arm matrix calculated by the main controller-object (myMotControl).

Conceptional layer In the initialisation of an ConceptionalLayer-object (see Lst. **??**), at first the control-space file is read, JointVector.dat for angle controllers, JointTorqueVector .dat for torque controllers and PositionTriads.dat for position controllers. In a next step,

the output space is calculated, which is trivial for angle and torque controllers, as the DoFs that are manipulated are the same. For the output-space of a conceptional position controller, the kinematic sub-chain from the 'relative'-`Triad` to the 'control'-`Triad` is analysed and all DoFs within this sub-chain build the output-space of the layer.

By knowing the output-space of the conceptional layer, the masking-permutation matrix can be calculated that can be used for the mapping of 'global'-`model` variables to 'local'-`layer` variables (see paragraph above). Then, all the PID-matrices for all poses are read in and the Jacobian matrices are calculated. Additionally, all other control variables that are used in the `ConceptionalLayer` implementation are initialised and allocated.

Transformational layer In the initialisation of an `object` of the `TransformationalLayer -class` (see Lst. E.22), similar as for the conceptional layer, the control-space file is read in that has always the form of a joint-DoF vector defined in `JointVector.dat`. Again, the permutation to the 'global'-`model` DoF-space is calculated to obtain a 'local'-`Layer` DoF-space. This 'local'-space of a `TransformationalLayer object` is the compared to the 'local'-output-space of the respective `ConceptionalLayer`. These two spaces must contain the same DoFs, if not, an error appears and the initialisation is aborted.

Similarly, the MTUs that are involved in manipulating the specified joints can be identified from the 'local' DoF-space, by analysing the anthropometry of the musculoskeletal model. With this, a 'local' MTU-space is obtained and a respective masking-permutation matrix can be calculated to project into the 'global' `model` MTU-space.

Finally, the PID-parameter matrices are read for all poses and the Jacobian matrices are calculated initially. Additionally, all other control variables that are used in the `TransformationallLayer` implementation are initialised and allocated.

Structural layer In the `StructuralLayer` initialisation (see Lst. E.36), the MTU-vector is read in from the `MuscleVector.dat`-file and compared to the 'local' MTU-vector predicted by the respective `TransformationalLayer`. These two MTU-vectors must contain exactly the same MTUs in order to ensure a proper communication between the transformational and the structural layer. When a mismatch is detected, an error message appears and the initialisation is aborted. For a well-defined MTU-vector, a masking-permutation matrix is calculated for a mapping between the 'global' `model` MTU-space and the 'local' `Layer` MTU-space. Then, the low-level λ-control parameters are read in, namely the κ-parameters and the reference co-contractions $\boldsymbol{u}_{\text{ref}}^{\text{coc}}$. The desired CE-lengths λ are only read for a controller with the control space of CE-lengths, specified by the `control_architecture byte` with the λ-bit $b_\lambda = 1$ in (E.1). All other control variables that are needed are allocated and initialised (see Lst. E.36).

E.3 Runtime routines of the control algorithms

During simulation runtime, in each simulation step, the biophysical sensor data must be updated and provided to the control architecture, the PID-control laws must be executed and the Jacobian matrices must be calculated based on the actual system state. Which control relevant state data of the system must be provided to the control architecture hereby depends on the types of controllers that are active, as specified with the `control_architecture` byte (E.1). For any higher-level control that requires the full hierarchy, each a conceptional, a transformational and a structural layer are needed. In addition, in each simulation step, the actual `control_state` is checked and when a new state is detected (either by time or by an event), the specified PID parameters for all layers and desired values for the conceptional control are updated.

```
1       #ifdef CONTROL
2       if (dbg_uode) cout << "Info(): UserODE:  CONTROL" << endl;
3       if(calcJacs & 0b0011 ) {//Angle and Torque Jacobians
```

```
4               Eigen::VectorXd Lce_vec = Eigen::VectorXd::Zero(nMUSC);
5               Eigen::VectorXd Fmtu_vec = Eigen::VectorXd::Zero(nMUSC);
6               Eigen::MatrixXd dLseedLce_mat = Eigen::MatrixXd::Zero(nMUSC,nMUSC);
7               Eigen::MatrixXd dFcedLce_mat = Eigen::MatrixXd::Zero(nMUSC,nMUSC);
8               Eigen::MatrixXd MA_dof  = Eigen::MatrixXd::Zero(nMUSC, n_dof);
9               Eigen::VectorXd Tht_is_glb = Eigen::VectorXd::Zero(n_dof);
10              Eigen::VectorXd Tht_d_glb = Eigen::VectorXd::Zero(n_dof);
11
12              UpdateMomentArms(MA_dof);
13              for (int i = 0; i < nMUSC; ++i){
14                      dLseedLce_mat(i,i)=myMuscles[i].dLseedLce();
15                      Lce_vec(i)=myMuscles[i].get_lCE(); // //Lce_vec=myMuscles[i].get_lCE();
16                      if(calcJacs & 0b0010){
17                              dFcedLce_mat(i,i)=myMuscles[i].dFcedLce(); //if TORQUE CRL
18                              Fmtu_vec(i)=myMuscles[i].get_fMTU();
19                      }
20              }
21              for (int i = 0; i < n_dof; ++i){
22                      Tht_is_glb(i)=q[i];
23                      Tht_d_glb(i)=qd[i];
24              }
25              myMotControl.set_Mamat(MA_dof);
26              myMotControl.set_DlseeDlce(dLseedLce_mat);
27              myMotControl.set_Lcevec(Lce_vec);
28              myMotControl.set_Tht_is_glb(Tht_is_glb);
29              myMotControl.set_Tht_d_glb(Tht_d_glb);
30              if(calcJacs & 0b0010){
31                      myMotControl.set_dFcedLce(dFcedLce_mat);//if TORQUE CTRL
32                      myMotControl.set_Fmtuvec(Fmtu_vec);//if TORQUE CTRL
33              }
34      }
35      if (dbg_uode) cout << "Info(): UserODE:  CONTROL2" << endl;
36      myMotControl.UpdateLayers(t, z, zd);
37      Eigen::VectorXd mystim = Eigen::VectorXd::Zero(nMUSC);
38      mystim = myMotControl.UpdateStim(t, ndof, nMUSC, z, zd);
39      #endif
```

Here the moment arms calculation is executed for the global system, as well as other controller relevant information, such as the matrices $\frac{\partial \boldsymbol{l}^{\mathrm{CE}}}{\partial \boldsymbol{l}^{\mathrm{SEE}}}$ (used for the Jacobian $J^{\lambda\theta}$ (3.9)), $\frac{\partial \boldsymbol{f}^{\mathrm{CE}}}{\partial \boldsymbol{l}^{\mathrm{CE}}}$ (used for the Jacobian $J^{\theta\tau}$ (3.36)). The two main functions to update the controller variables in this this code-block are `UpdateLayers()` and `UpdateStim()` in lines 39 and 41 (both defined in `ucontrol.cpp` in the module `u_control` and listed in Lst. E.4 and Lst. E.5), where the first updates the layer's variables to the current system's state and the latter executes the PID-controllers and the Jacobian-based transformations to eventually generate the stimulation signal for the MTUs (3.6). Both functions, `UpdateLayers()` and `UpdateStim()` further call individual layer functions that straightforwardly follow the mathematical design of the control architecture from Pt. II. The most important conceptional layer functions are given in Sec. E.5.3, those of the transformational layer in E.5.4 and those of the structural layer in E.5.5.

E.4 Different ways to set desired control states

In this section different ways to parametrise the input of desired values to the controlarchitecture are presented. As an example, the joint angle controlled movements of the lower limbs from Sec. 5.1 are used. For the remaining control spaces an analoug approach can be used and, of course, a user is not limited to the shown examples but encouraged to find own solutions.

The desired movement in Sec. 5.1 is specified in terms of discrete set-points of desired joint angles as follows:

$$\boldsymbol{\theta}^{\mathrm{des}}_{\mathrm{Upex}} = \begin{cases} [\ 0.0 \quad 0.0 \quad 0.0 \quad 0.0\ ]^T, & t \in (\ 0.0\mathrm{s} \ \ldots \ 1.5\mathrm{s}\ ] \\ [\ -40.0 \quad 30.0 \quad 110.0 \quad -20\ ]^T, & t \in (\ 1.5\mathrm{s} \ \ldots \ 3.0\mathrm{s}\ ] \\ [\ -110.0 \quad 10.0 \quad 30.0 \quad 30.0\ ]^T, & t \in (\ 3.0\mathrm{s} \ \ldots \ 4.5\mathrm{s}\ ] \\ [\ -10.0 \quad 40.0 \quad 70.0 \quad 0.0\ ]^T, & t \in (\ 4.5\mathrm{s} \ \ldots \ 6.0\mathrm{s}\ ] \end{cases}. \tag{E.3}$$

There are several ways to set these desired values in the simulation framework `demoa`. The most simple approach is to configure a `time-based` controller and to define the `poses` in the respective files, as outlined in section E.1. An `event-based` approach gives a more

detailed way in operating the controllers, for example, a new pose can be set in the file `usrEventcontrol.cpp` by

```
{static bool once=true; //Initially swiched off
if (t>1.5 && once){
once=false;
ctrlstate=1;
cout << "Info(): Time=" << t << " Setting Controlstate to: " << ctrlstate <<
    endl;
myMotControl.set_ControlState(ctrlstate);
}}
```

This sets the new control state `ctrlstate=1` to all controllers that are defined sets the respective parameters that are defined in `datacontrol`.

To only set a new control state for a single controller, for example the following code snippet can be used:

```
{static bool once=true; //Initially swiched off
if (t>3.0 && once){
once=false;
ctrlstate=2;
cout << "Info(): Time=" << t << " Setting Controlstate to: " << ctrlstate <<
    endl;
//myMotControl.set_ControlState(ctrlstate);
myMotControl.set_ControlState_Ang(ID_Ang_LowEx_R,ctrlstate);
}}
```

This sets the new control state `ctrlstate=2` for the angle controller with the unique ID `ID_Ang_LowEx_R`. The ID of a controller corresponds to it's occurence in the list of defined controllers, e.g. in `AngleController.dat`.

As a third alternative, the desired joint angles of a controller can be set directly without changing the control state:

```
{static bool once=true;
if (t>4.5 && once){
once=false;
DES_Ang_LowEx_R(0)=-10.0;
DES_Ang_LowEx_R(1)=40.0;
DES_Ang_LowEx_R(2)=70.0;
DES_Ang_LowEx_R(3)=0.0;
UC_Ang_LowEx_R=Eigen::VectorXd::Ones(n_Ang_LowEx_R_Ucoc)*0.1;

myMotControl.set_Ang_des(ID_Ang_LowEx_R, DES_Ang_LowEx_R);
myMotControl.set_Ang_U_coc_lcl(ID_Ang_LowEx_R, UC_Ang_LowEx_R);
}}
```

This directly sets the desired angles and the reference co-contraction for the MTUs while keeping all PID parameters. This has the advantage that no further poses have to be defined in `datacontrol`.

E.5 Source code of the control module functions

E.5.1 Functions in `usrsetmks.cpp`

Listing E.1: Algorithm for calculating the adjacency matrix

```
1 void calc_BodyJointAdjacency_mat(Eigen::MatrixXd& adjbjmat){
2
3   bool dbg_adj=false;
4   Eigen::MatrixXd Adj_BJ_mat=Eigen::MatrixXd::Zero(nrgb,nrgb);
5   const Body * BodyCount;
6   const Body * BodySearch;
7   const Body * BodyTo;
8   const Body * BodyFrom;
9   const Joint * JointCount;
10  const Joint * JointSearch;
11  const Triad * TriadTo;
12  const Triad * TriadFrom;
13  int BodyID_Count=0;
14  BodyCount=model.body(BodyID_Count);
15  while(BodyID_Count<nrgb){
16    static int TriadID_to=0;
17    static int TriadID_from=0;
18    JointCount=model.body(BodyID_Count)->joint();
19    static  int JointID_search=0;
20    static  int JointID_count=0;
21    JointSearch = model.joint(JointID_search);
22    while(JointCount->name() != JointSearch->name()){
23        JointID_search++;
24        JointSearch = model.joint(JointID_search);
25      }
26    JointID_count=JointID_search;
27    TriadFrom=JointCount->from();
28    TriadTo=JointCount->to();
29    if( TriadFrom->parentname()=="world" || TriadTo->parentname()=="world") {
30      BodyID_Count++;
31      continue;
32    }
33    BodyTo=TriadTo->parent();
34    BodyFrom=TriadFrom->parent(); // one of these is BodyCunt?
35    static int BodyID_Search=0;
36    static int BodyID_To=0;
37    BodySearch=model.body(BodyID_Search);
38    while(BodySearch->name()!=BodyTo->name()){
39      BodyID_Search++;
40      BodySearch=model.body(BodyID_Search);
41    }
42    BodyID_To=BodyID_Search;
43    BodyID_Search=0;
44    static int BodyID_From=0;
45    BodySearch=model.body(BodyID_Search);
46    while(BodySearch->name()!=BodyFrom->name()){
47      BodyID_Search++;
48      BodySearch=model.body(BodyID_Search);
49    }
50    BodyID_From=BodyID_Search;
51    Adj_BJ_mat(BodyID_From, BodyID_To)=JointID_count;
52    Adj_BJ_mat(BodyID_To, BodyID_From)=JointID_count;
53    BodyID_Count++;
54  }
55    adjbjmat=Adj_BJ_mat;
56  return;
57 }
```

Listing E.2: Algorithm for calculating the moment arms

```
void UpdateMomentArms(Eigen::MatrixXd & madof){
  madof=Eigen::MatrixXd::Zero(nMUSC, n_dof);
  for (int muscID = 0; muscID < nMUSC; ++muscID){
    static Eigen::MatrixXd Path_OI= Eigen::MatrixXd::Zero(4,n_joints);
    static Eigen::Vector4i mbvec;
    mrvec = new Eigen::Vector4d[4];
    mbvec(0)=userforcectrl[muscID].idx0;
    mbvec(1)=userforcectrl[muscID].idxEll1;
    mbvec(2)=userforcectrl[muscID].idxEll2;
    mbvec(3)=userforcectrl[muscID].idx1;
    mrvec[0](0)=userforcectrl[muscID].r0[0];
    mrvec[0](1)=userforcectrl[muscID].r0[1];
    mrvec[0](2)=userforcectrl[muscID].r0[2];
    mrvec[0](3)=1;
    mrvec[1](0)=userforcectrl[muscID].rDF1[0];
    mrvec[1](1)=userforcectrl[muscID].rDF1[1];
    mrvec[1](2)=userforcectrl[muscID].rDF1[2];
    mrvec[1](3)=1;
    mrvec[2](0)=userforcectrl[muscID].rDF2[0];
    mrvec[2](1)=userforcectrl[muscID].rDF2[1];
    mrvec[2](2)=userforcectrl[muscID].rDF2[2];
    mrvec[2](3)=1;
    mrvec[3](0)=userforcectrl[muscID].r1[0];
    mrvec[3](1)=userforcectrl[muscID].r1[1];
    mrvec[3](2)=userforcectrl[muscID].r1[2];
    mrvec[3](3)=1;
    Path_OI= Eigen::MatrixXd::Zero(4,n_joints);
    for (int j = 0; j < 3; ++j){
        Path_OI.row(j)+=myMotControl.Dijkstra(Adj_BJ, mbvec(j)-1, mbvec(3)-1);
    }
    for (int j = 3; j > 0; --j){
        Path_OI.row(j)+=myMotControl.Dijkstra(Adj_BJ, mbvec(j)-1, mbvec(0)-1);
    }
    int n_jcrs=0;
    int n_dcrs=0;
    Eigen::VectorXd Joints_Crossed;
    Eigen::VectorXd Dofs_Crossed;
    Joints_Crossed=Eigen::VectorXd::Zero(0);
    Dofs_Crossed =Eigen::VectorXd::Zero(0);
    for (int i = 0; i < n_joints; ++i){
        if ((Path_OI.col(i)).norm() != 0 ){
            Joints_Crossed.conservativeResize(Joints_Crossed.size()+1);
            Joints_Crossed(Joints_Crossed.size()-1) = i;
            for (int ii = 0; ii < n_dof; ++ii){
                if (DofToJoint(ii)==i){
                    Dofs_Crossed.conservativeResize(Dofs_Crossed.size()+1);
                    Dofs_Crossed(Dofs_Crossed.size()-1) = ii;
                }
            }
        }
    }
    n_jcrs=Joints_Crossed.size();
    n_dcrs=Dofs_Crossed.size();
    for (int i_dof = 0;i_dof < n_dcrs; ++i_dof){
        int j=DofToJoint(Dofs_Crossed(i_dof));
        int dofID=Dofs_Crossed(i_dof);
        int lastsign=0;
        int breakingpoint=-1;
        for (int i = 0; i < 4; ++i){
            if(lastsign!=Path_OI(i,j) && lastsign!=0){
                breakingpoint=i;
                break;
            }
            else lastsign=Path_OI(i,j);
        }
        double momentarm_m_dof;
        momentarm_m_dof= MomentArm_geo(dofID, j, breakingpoint, mrvec );
        madof(muscID, dofID)=momentarm_m_dof;
    }
}
return;
}

double MomentArm_geo(int jdofid,int JointID, int BrPnt, Eigen::Vector4d * MR_vec){
  int ToID=model.getbodyidx(model.joint(JointID)->to()->parent());
  Eigen::Vector4d Li_vec_to;
  Li_vec_to(0)=Li[jdofid+1](2,1);
  Li_vec_to(1)=Li[jdofid+1](0,2);
  Li_vec_to(2)=Li[jdofid+1](1,0);
  Li_vec_to(3)=0;
  Eigen::Matrix4d M_to_jnt;
  model.joint(JointID)->to()->Mmatrix(M_to_jnt);
  Eigen::Vector4d PJnt_wd;
  PJnt_wd=M0o[ToID]*M_to_jnt.block(0,3,4,1);
  Eigen::Vector4d PJnt_to;
  PJnt_to=M_to_jnt.block(0,3,4,1);
  Eigen::Vector4d PMto_wd;
  Eigen::Vector4d PMto_to;
```

```
89    Eigen::Vector4d PMfr_wd;
90    Eigen::Vector4d PMfr_to;
91    Eigen::Vector4d V_Mto_Mfr_to;
92    Eigen::Vector4d V_unt_to;
93    Eigen::Vector4d V_PA_to;
94    Eigen::Vector4d V_AX_to;
95    Eigen::Vector4d V_MA_to;
96    Eigen::Vector4d V_MA_prl_to;
97    PMto_wd(0)=MR_vec[BrPnt](0);
98    PMto_wd(1)=MR_vec[BrPnt](1);
99    PMto_wd(2)=MR_vec[BrPnt](2);
100   PMto_wd(3)=1;
101   PMto_to=M0o[ToID].inverse()*PMto_wd;
102   PMfr_wd(0)=MR_vec[BrPnt-1](0);
103   PMfr_wd(1)=MR_vec[BrPnt-1](1);
104   PMfr_wd(2)=MR_vec[BrPnt-1](2);
105   PMfr_wd(3)=1;
106   PMfr_to=M0o[ToID].inverse()*PMfr_wd;
107           // P - point                        !!P=Jnt
108           // D - direction of line (unit length)        !!V2_unit
109           // A - point in line                  !!A=Mto/Mfr      ->P-A = -V_Jnt_Mto !!
110
111           // X - base of the perpendicular line
112
113           //      P
114           //     /|
115           //    / |
116           //   /  v
117           // A---X----->D
118
119           // (P-A).D == |X-A|
120
121           // X == A + ((P-A).D)D
122           // Desired perpendicular: X-P              !!V3
123           //source: https://stackoverflow.com/questions/5227373/minimal-perpendicular-vector-between-a-point-and-a-line
124   V_Mto_Mfr_to=PMfr_to-PMto_to;    //Muscle String Vector FROM-TO wrt TO || A-->D
125   V_unt_to=V_Mto_Mfr_to/V_Mto_Mfr_to.norm();
126   V_PA_to= PJnt_to-PMto_to; //P---A
127   V_AX_to=(V_PA_to.dot(V_Mto_Mfr_to))*V_unt_to; //Vector A--X wrt TO
128   V_MA_to = (PMto_to+V_AX_to)-PJnt_to;  //Moment Arm Vector wrt TO
129   Eigen::Vector3d Va;
130   Eigen::Vector3d Vb;
131   Vb=V_unt_to.head(3);
132   Va=V_MA_to.head(3);
133   V_MA_prl_to.head(3)= Va.cross(Vb);
134   double vmaprl_to;
135   vmaprl_to=-V_MA_prl_to.dot(Li_vec_to);
136   return vmaprl_to;
137 }
```

E.5.2 Functions in `ucontrol.cpp`

This is a test that is displayed in E.3.

Listing E.3: General control initialisation function

```
1
2 void MotorControl::Init(int ndof, int nmusc, Eigen::VectorXd& ICEopt, std::vector<std::string>& mnames, int contarch,
      int eventflag, Eigen::MatrixXd& Mamat, int calcjacs){ //#TODO redef nput vars!
3       n_dof=model.dof();
4       int n_tv=0;
5       n_musc=nmusc;
6       ctrl_arch=contarch;
7       SolverCtrl sc;
8       model.solver_controls(&sc);
9       dtstep=sc.dt;
10      event_flag=eventflag;
11      cout << "Info(): EventFlag= " << event_flag << ". Default Control State=0." << endl;
12      ControlState=0;
13      if (event_flag==0) cout << "Info(): Time-based control. Use TPose.dat to define Poses and Times!" << endl;
14      else cout << "Info(): Event-based control. Use EPose.dat to define Poses and Event-Numbers! Use uEventControl.
            cpp to define Sensors and Events!" << endl;
15
16      current_kappa.resize(n_musc,1);
17      current_kappa = kappa*Eigen::VectorXd::Ones(n_musc);
18      current_lce.resize(n_musc,1);
19      current_lce = Eigen::VectorXd::Zero(n_musc);
20      current_vce.resize(n_musc,1);
21      current_vce = Eigen::VectorXd::Zero(n_musc);
22      current_sigma.resize(n_musc,1);
23      current_sigma = sigma*Eigen::VectorXd::Ones(n_musc);
24
25      std::string uinitname;
26      uiv=new double[n_musc];
27      uivname = "datacontrol/u_init.dat";
28      cout << "Info(): Read Uinit-Filename: "<< uivname << endl; //better output
29      readVectorFromFile(n_musc, uivname, uiv,  u__init);
30      uinit.resize(n_musc,1);
31      for(int j=0; j<n_musc; j++){
32              uinit(j)=uiv[j];
33      }
34      cout << "Info(): u_init read finished" << endl;
35
36      n_tv=read_TEpose(); //incl parameter?
37      read_ControlWeights(); //Probably Rewrite // #TODO
38
39      int calcJacs=calcjacs;
40      bool CALC_JAC_F=(calcJacs & 0b1000);
41      bool CALC_JAC_X=(calcJacs & 0b0100);
42      bool CALC_JAC_T=(calcJacs & 0b0010);
43      bool CALC_JAC_Q=(calcJacs & 0b0001);
44
45      OPEN_LOOP =                             (ctrl_arch & 0b00001);
46      ALPHA_CONTROL =                         (ctrl_arch & 0b11111);  //Always?
47      STRUCTURAL_LAYER =                      (ctrl_arch & 0b11111);
48      TRANSFORMATIONAL_LAYER =        (ctrl_arch & 0b11110);
49      CONCEPTOINAL_LAYER =            (ctrl_arch & 0b11110);
50      LAMBDA_CONTROL =                        (ctrl_arch & 0b00001);
51      ANGLE_CONTROL =                         (ctrl_arch & 0b00010);
52      TORQUE_CONTROL =                        (ctrl_arch & 0b00100);
53      POSITION_CONTROL =                      (ctrl_arch & 0b01000);
54      FORCE_CONTROL =                         (ctrl_arch & 0b10000);
55      std::cout << "Info(): ctrl_arch=|" << FORCE_CONTROL << "|"<< POSITION_CONTROL << "|"<< TORQUE_CONTROL << "|" <<
            ANGLE_CONTROL << "|"<< LAMBDA_CONTROL << "|"<< endl;
56      std::cout << "Info(): ctrl_arch=|F|P|T|Q|L|" << endl;
57
58      Eigen::VectorXd common_cDoF_vec=Eigen::VectorXd::Zero(n_dof);
59      n_lbd=0;
60      n_ang=0;
61      n_trq=0;
62      n_pos=0;
63      n_frc=0;
64      n_ctr=0;
65
66      if (TRANSFORMATIONAL_LAYER)     Tht_vec=Eigen::VectorXd::Zero(n_dof);
67
68      if (LAMBDA_CONTROL){ //init Lambda Controller
69              std::string LbdCtrlFile;
70              LbdCtrlFile="datacontrol/Layers/Lambda_Control/LambdaController.dat";
71              n_lbd=read_LbdCtrlFile(LbdCtrlFile);
72              for (int i = 0; i < n_lbd; ++i) cout << "info(): lbdnames[i]= " << lbdnames[i] << endl;
73              LbdCtrl_SL = new StructuralLayer[n_lbd];
74              for (int i = 0; i < n_lbd; ++i){
75                      LbdCtrl_SL[i]=StructuralLayer(1, n_tv);
76                      cout << "////////////////////////////////////////////////////" << endl;
77                      cout << "////////INIT LAMBDA CONTROLLER: "<< lbdnames[i] <<  endl;
78                      cout << "////////////////////////////////////////////////////"<< endl;
79                      LbdCtrl_SL[i].set_PoseNames(*tpname);
80                      LbdCtrl_SL[i].set_nmtu(nmusc);
```

```
81                      LbdCtrl_SL[i].set_MtuNames(mnames);
82                      LbdCtrl_SL[i].set_ICEopt(ICEopt);
83                      LbdCtrl_SL[i].set_IDc("I");
84                      LbdCtrl_SL[i].set_SL_Folder("datacontrol/Layers/Lambda_Control/"+lbdnames[i]+"/StructuralLayer/
                            ");
85                      LbdCtrl_SL[i].set_mVec_file( "MuscleVector.dat");
86                      LbdCtrl_SL[i].set_UcocRef_Folder("Alpha_Controller/");
87                      LbdCtrl_SL[i].set_UcocRef0(u__open);
88                      LbdCtrl_SL[i].set_Sigma0(sigma);
89                      LbdCtrl_SL[i].set_Lambda_Folder("Lambda_Controller/Lambda/");
90                      LbdCtrl_SL[i].set_Lambda0(lambda0);
91                      LbdCtrl_SL[i].set_Kappa_Folder("Lambda_Controller/Kappa/");
92                      LbdCtrl_SL[i].set_Kappa0(kappa);
93                      LbdCtrl_SL[i].Init_SL();
94              }
95      }
96
97      if (ANGLE_CONTROL) //init_AngleControl
98      {
99              cout << "////////////////////////////////////////////////////" << endl;
100             cout << "////////INIT ANGLE CONTROLLER"<< endl;
101             cout << "////////////////////////////////////////////////////"<< endl;
102             std::string AngCtrlFile;
103             AngCtrlFile="datacontrol/Layers/Angle_Control/AngleController.dat";
104             n_ang=read_AngCtrlFile(AngCtrlFile);
105             for (int i = 0; i < n_ang; ++i) cout << "info(): angnames[i]= " << angnames[i] << endl;
106             AngCtrl_CL=new ConceptionalLayer[n_ang];
107             AngCtrl_TL = new TransformationalLayer[n_ang];
108             AngCtrl_SL = new StructuralLayer[n_ang];
109             for (int i = 0; i < n_ang; ++i){
110                     AngCtrl_CL[i]=ConceptionalLayer(2, n_tv);
111                     AngCtrl_CL[i].set_PoseNames(*tpname);
112                     AngCtrl_CL[i].set_CL_Folder("datacontrol/Layers/Angle_Control/"+angnames[i]+"/ConceptionalLayer
                            /");
113                     AngCtrl_CL[i].set_cs_file("JointVector.dat");
114                     AngCtrl_CL[i].set_IDc("q");
115                     AngCtrl_CL[i].Init_CL_QT();
116                     Eigen::VectorXd Ang_cVar_vec=AngCtrl_CL[i].get_cVar_vec();
117                     Eigen::VectorXd Ang_oVar_vec=AngCtrl_CL[i].get_oVar_vec();
118                     Eigen::VectorXd Ang_oVar_global=AngCtrl_CL[i].get_global_oVars();
119                     for (int i = 0; i < n_dof; ++i)if (Ang_oVar_global(i)>=common_cDoF_vec(i)) common_cDoF_vec(i)=
                            Ang_oVar_global(i);
120                     AngCtrl_TL[i]=TransformationalLayer(AngCtrl_CL[i].get_ControlSpace(), n_tv);
121                     AngCtrl_TL[i].set_dtstep(dtstep);
122                     AngCtrl_TL[i].set_PoseNames(*tpname);
123                     AngCtrl_TL[i].set_cDoFvec_CL(Ang_cVar_vec);
124                     AngCtrl_TL[i].set_nmtu(nmusc);
125                     AngCtrl_TL[i].set_MtuNames(mnames);
126                     AngCtrl_TL[i].set_TL_Folder("datacontrol/Layers/Angle_Control/"+angnames[i]+"/
                            TransformationalLayer/");
127                     AngCtrl_TL[i].set_cDoF_file( "JointVector.dat");
128                     AngCtrl_TL[i].set_mVec_file("Jacobians/MuscleVector.dat");
129                     AngCtrl_TL[i].set_IDc("q");
130                     AngCtrl_TL[i].set_PID_Tht_Folder("PID/Theta_PID/");
131                     AngCtrl_TL[i].set_PID_Tht_Lbd_Folder("PID/Theta_Lambda_PID/");
132                     AngCtrl_TL[i].set_CALC_JACS(CALC_JAC_Q);
133                     if(CALC_JAC_Q){
134                             AngCtrl_TL[i].set_Mamat_glb(Ma_mat);
135                             AngCtrl_TL[i].set_dLseedLce_glb(dLseedLce_mat);
136                             AngCtrl_TL[i].set_Lcevec_glb(Lce_vec);
137                             AngCtrl_TL[i].set_Uvec_glb(uinit);
138                     }else{
139                             AngCtrl_TL[i].set_Jac_U_Tht_Folder("Jacobians/Joint_Stimulation_Jacobian_JUQ/");
140                             AngCtrl_TL[i].set_Jac_Lbd_Tht_Folder("Jacobians/Joint_Length_Jacobian_JLQ/");
141                     }
142                     AngCtrl_TL[i].Init_TL();
143                     Eigen::VectorXd Ang_mVec_TL=AngCtrl_TL[i].get_mVec_TL();
144                     AngCtrl_SL[i]=StructuralLayer(AngCtrl_CL[i].get_ControlSpace(), n_tv);
145                     AngCtrl_SL[i].set_PoseNames(*tpname);
146                     AngCtrl_SL[i].set_mVec_TL(Ang_mVec_TL);
147                     AngCtrl_SL[i].set_nmtu(nmusc);
148                     AngCtrl_SL[i].set_MtuNames(mnames);
149                     AngCtrl_SL[i].set_ICEopt(ICEopt);
150                     AngCtrl_SL[i].set_IDc("q");
151                     AngCtrl_SL[i].set_SL_Folder("datacontrol/Layers/Angle_Control/"+angnames[i]+"/StructuralLayer/"
                            );
152                     AngCtrl_SL[i].set_mVec_file( "MuscleVector.dat");
153                     AngCtrl_SL[i].set_Kappa_Folder("Lambda_Controller/Kappa/");
154                     AngCtrl_SL[i].set_Kappa0(kappa);
155                     AngCtrl_SL[i].set_UcocRef_Folder("Alpha_Controller/");
156                     AngCtrl_SL[i].set_UcocRef0(u__open);
157                     AngCtrl_SL[i].set_Sigma0(sigma);
158                     AngCtrl_SL[i].Init_SL();
159             }
160     }
161
162     if (TORQUE_CONTROL) {
163             cout << "////////////////////////////////////////////////////" << endl;
164             cout << "////////INIT TORQUE CONTROLLER "<< endl;
165             cout << "////////////////////////////////////////////////////"<< endl;
```

```
166             std::string TrqCtrlFile;
167             TrqCtrlFile="datacontrol/Layers/Torque_Control/TorqueController.dat";
168             n_trq=read_TrqCtrlFile(TrqCtrlFile);
169             for (int i = 0; i < n_trq; ++i) cout << "info(): trqnames[i]= " << trqnames[i] << endl;
170             TrqCtrl_CL=new ConceptionalLayer[n_trq];
171             TrqCtrl_TL = new TransformationalLayer[n_trq];
172             TrqCtrl_SL = new StructuralLayer[n_trq];
173             for (int i = 0; i < n_trq; ++i){
174                     TrqCtrl_CL[i]=ConceptionalLayer(3,n_tv);
175                     TrqCtrl_CL[i].set_dtstep(dtstep);
176                     TrqCtrl_CL[i].set_PoseNames(*tpname);
177                     TrqCtrl_CL[i].set_CL_Folder("datacontrol/Layers/Torque_Control/"+trqnames[i]+"/
                            ConceptionalLayer/");
178                     TrqCtrl_CL[i].set_cs_file("TorqueJointVector.dat");
179                     TrqCtrl_CL[i].set_IDc("t");
180                     TrqCtrl_CL[i].set_CALC_JACS(CALC_JAC_T);
181                     TrqCtrl_CL[i].set_PID_Folder("Torque_PID/");
182                     TrqCtrl_CL[i].set_nmtu(nmusc);
183                     if(CALC_JAC_T){
184                             TrqCtrl_CL[i].set_Mamat_glb(Ma_mat);
185                             TrqCtrl_CL[i].set_dLseedLce_glb(dLseedLce_mat);
186                             TrqCtrl_CL[i].set_dFcedLce_glb(dFcedLce_mat);
187                     }else{
188                             TrqCtrl_CL[i].set_Jac_Folder("Torque_Jacobian_JQT/");
189                             TrqCtrl_CL[i].set_Jac_sfx("jqt");
190                     }
191                     TrqCtrl_CL[i].Init_CL_QT();
192                     Eigen::VectorXd Trq_cVar_vec=TrqCtrl_CL[i].get_cVar_vec();
193                     Eigen::VectorXd Trq_oVar_vec=TrqCtrl_CL[i].get_oVar_vec();
194                     Eigen::VectorXd Trq_oVar_global=TrqCtrl_CL[i].get_global_oVars();
195                     for (int i = 0; i < n_dof; ++i)if (Trq_oVar_global(i)>=common_cDoF_vec(i)) common_cDoF_vec(i)=
                            Trq_oVar_global(i);
196                     TrqCtrl_TL[i]=TransformationalLayer(TrqCtrl_CL[i].get_ControlSpace(), n_tv);
197                     TrqCtrl_TL[i].set_dtstep(dtstep);
198                     TrqCtrl_TL[i].set_PoseNames(*tpname);
199                     TrqCtrl_TL[i].set_cDoFvec_CL(Trq_cVar_vec);
200                     TrqCtrl_TL[i].set_nmtu(nmusc);
201                     TrqCtrl_TL[i].set_MtuNames(mnames);
202                     TrqCtrl_TL[i].set_TL_Folder("datacontrol/Layers/Torque_Control/"+trqnames[i]+"/
                            TransformationalLayer/");
203                     TrqCtrl_TL[i].set_cDoF_file( "JointVector.dat");
204                     TrqCtrl_TL[i].set_mVec_file("Jacobians/MuscleVector.dat");
205                     TrqCtrl_TL[i].set_IDc("t");
206                     TrqCtrl_TL[i].set_PID_Tht_Folder("PID/Theta_PID/");
207                     TrqCtrl_TL[i].set_PID_Tht_Lbd_Folder("PID/Theta_Lambda_PID/");
208                     TrqCtrl_TL[i].set_CALC_JACS(CALC_JAC_Q);
209                     if(CALC_JAC_Q){
210                             TrqCtrl_TL[i].set_Mamat_glb(Ma_mat);
211                             TrqCtrl_TL[i].set_dLseedLce_glb(dLseedLce_mat);
212                             TrqCtrl_TL[i].set_Lcevec_glb(Lce_vec);
213                             TrqCtrl_TL[i].set_Uvec_glb(uinit);
214                     }else{
215                             TrqCtrl_TL[i].set_Jac_U_Tht_Folder("Jacobians/Joint_Stimulation_Jacobian_JUQ/");
216                             TrqCtrl_TL[i].set_Jac_Lbd_Tht_Folder("Jacobians/Joint_Length_Jacobian_JLQ/");
217                     }
218                     TrqCtrl_TL[i].Init_TL();
219                     Eigen::VectorXd Trq_mVec_TL=TrqCtrl_TL[i].get_mVec_TL();
220                     TrqCtrl_SL[i]=StructuralLayer(TrqCtrl_CL[i].get_ControlSpace(), n_tv);
221                     TrqCtrl_SL[i].set_PoseNames(*tpname);
222                     TrqCtrl_SL[i].set_mVec_TL(Trq_mVec_TL);
223                     TrqCtrl_SL[i].set_nmtu(nmusc);
224                     TrqCtrl_SL[i].set_MtuNames(mnames);
225                     TrqCtrl_SL[i].set_ICEopt(ICEopt);
226                     TrqCtrl_SL[i].set_IDc("t");
227                     TrqCtrl_SL[i].set_SL_Folder("datacontrol/Layers/Torque_Control/"+trqnames[i]+"/StructuralLayer/
                            ");
228                     TrqCtrl_SL[i].set_mVec_file( "MuscleVector.dat");
229                     TrqCtrl_SL[i].set_UcocRef_Folder("Alpha_Controller/");
230                     TrqCtrl_SL[i].set_UcocRef0(u__open);
231                     TrqCtrl_SL[i].set_Sigma0(sigma);
232                     TrqCtrl_SL[i].set_Kappa_Folder("Lambda_Controller/Kappa/");
233                     TrqCtrl_SL[i].set_Kappa0(kappa);
234                     TrqCtrl_SL[i].Init_SL();
235             }
236     }
237
238     if (POSITION_CONTROL) {
239             cout << "////////////////////////////////////////////////////" << endl;
240             cout << "////////INIT POSITION CONTROLLER TODO: INCLUDE JACOBIAN"<< endl;
241             cout << "////////////////////////////////////////////////////"<< endl;
242             std::string PosCtrlFile;
243             PosCtrlFile="datacontrol/Layers/Position_Control/PositionController.dat";
244             n_pos=read_PosCtrlFile(PosCtrlFile);
245             PosCtrl_CL=new ConceptionalLayer[n_pos];
246             PosCtrl_TL = new TransformationalLayer[n_pos];
247             PosCtrl_SL = new StructuralLayer[n_pos];
248             for (int i = 0; i < n_pos; ++i){
249                     PosCtrl_CL[i]=ConceptionalLayer(4, n_tv);
250                     PosCtrl_CL[i].set_dtstep(dtstep);
251                     PosCtrl_CL[i].set_PoseNames(*tpname);
```

```
252                     PosCtrl_CL[i].set_CLIndex(i);
253                     PosCtrl_CL[i].set_CL_Folder("datacontrol/Layers/Position_Control/"+ posnames[i] +"/
                            ConceptionalLayer/");
254                     PosCtrl_CL[i].set_cs_file("PositionTriads.dat");
255                     PosCtrl_CL[i].set_PID_Folder("Position_PID/");
256                     PosCtrl_CL[i].set_IDc("x");
257                     PosCtrl_CL[i].set_CALC_JACS(CALC_JAC_X);
258                     if(CALC_JAC_X){ //
259                     }else{
260                             PosCtrl_CL[i].set_Jac_Folder("Position_Jacobian_JQX/");
261                             PosCtrl_CL[i].set_Jac_sfx("jqx");
262                             PosCtrl_CL[i].set_os_file("Position_Jacobian_JQX/JointVector.dat");
263                     }
264                     PosCtrl_CL[i].Init_CL_XF();
265                     Eigen::VectorXd Pos_oVar_vec=PosCtrl_CL[i].get_oVar_vec();
266                     Eigen::VectorXd Pos_oVar_global=PosCtrl_CL[i].get_global_oVars();
267                     for (int i = 0; i < n_dof; ++i)if (Pos_oVar_global(i)>=common_cDoF_vec(i)) common_cDoF_vec(i)=
                            Pos_oVar_global(i);
268                     PosCtrl_TL[i]= TransformationalLayer(PosCtrl_CL[i].get_ControlSpace(), n_tv);
269                     PosCtrl_TL[i].set_dtstep(dtstep);
270                     PosCtrl_TL[i].set_PoseNames(*tpname);
271                     Eigen::VectorXd oVarVecCL=PosCtrl_CL[i].get_oVar_vec();
272                     PosCtrl_TL[i].set_cDoFvec_CL(oVarVecCL);
273                     PosCtrl_TL[i].set_nmtu(nmusc);
274                     PosCtrl_TL[i].set_MtuNames(mnames);
275                     PosCtrl_TL[i].set_TLIndex(i);
276                     PosCtrl_TL[i].set_TL_Folder("datacontrol/Layers/Position_Control/"+ posnames[i] +"/
                            TransformationalLayer/"); //
277                     PosCtrl_TL[i].set_cDoF_file( "JointVector.dat");
278                     PosCtrl_TL[i].set_mVec_file("Jacobians/MuscleVector.dat");
279                     PosCtrl_TL[i].set_IDc("x");
280                     PosCtrl_TL[i].set_PID_Tht_Folder("PID/Theta_PID/");
281                     PosCtrl_TL[i].set_PID_Tht_Lbd_Folder("PID/Theta_Lambda_PID/");
282                     PosCtrl_TL[i].set_CALC_JACS(CALC_JAC_Q);
283                     if(CALC_JAC_Q){
284                             PosCtrl_TL[i].set_Mamat_glb(Ma_mat);
285                             PosCtrl_TL[i].set_dLseedLce_glb(dLseedLce_mat);
286                             PosCtrl_TL[i].set_Lcevec_glb(Lce_vec);
287                             PosCtrl_TL[i].set_Uvec_glb(uinit);
288                     }else{
289                             PosCtrl_TL[i].set_Jac_U_Tht_Folder("Jacobians/Joint_Stimulation_Jacobian_JUQ/");
290                             PosCtrl_TL[i].set_Jac_Lbd_Tht_Folder("Jacobians/Joint_Length_Jacobian_JLQ/");
291                     }
292                     PosCtrl_TL[i].Init_TL();
293                     Eigen::VectorXd Pos_mVec_TL=PosCtrl_TL[i].get_mVec_TL();
294                     PosCtrl_SL[i]=StructuralLayer(PosCtrl_CL[i].get_ControlSpace(), n_tv);
295                     PosCtrl_SL[i].set_PoseNames(*tpname);
296                     PosCtrl_SL[i].set_mVec_TL(Pos_mVec_TL);
297                     PosCtrl_SL[i].set_nmtu(nmusc);
298                     PosCtrl_SL[i].set_MtuNames(mnames);
299                     PosCtrl_SL[i].set_ICEopt(ICEopt);
300                     PosCtrl_SL[i].set_IDc("p");
301                     PosCtrl_SL[i].set_SL_Folder("datacontrol/Layers/Position_Control/"+posnames[i]+"/
                            StructuralLayer/");
302                     PosCtrl_SL[i].set_mVec_file( "MuscleVector.dat");
303                     PosCtrl_SL[i].set_UcocRef_Folder("Alpha_Controller/");
304                     PosCtrl_SL[i].set_UcocRef0(u__open);
305                     PosCtrl_SL[i].set_Sigma0(sigma);
306                     PosCtrl_SL[i].set_Kappa_Folder("Lambda_Controller/Kappa/");
307                     PosCtrl_SL[i].set_Kappa0(kappa);
308                     PosCtrl_SL[i].Init_SL();
309             }
310     }
311
312     if (FORCE_CONTROL)
313     {
314             cout << "////////////////////////////////////////////////////" << endl;
315             cout << "////////INIT FORCE CONTROLLER TODO: INCLUDE JACOBIAN" << endl;
316             cout << "////////////////////////////////////////////////////"<< endl;
317             std::string FrcCtrlFile;
318             FrcCtrlFile="datacontrol/Layers/Force_Control/ForceController.dat";
319             n_frc=read_FrcCtrlFile(FrcCtrlFile);
320             for (int i = 0; i < n_frc; ++i) cout << "info(): xfnames[i]= " << frcnames[i] << endl;
321             FrcCtrl_CL=new ConceptionalLayer[n_frc];
322             FrcCtrl_TL = new TransformationalLayer[n_frc];
323             FrcCtrl_SL = new StructuralLayer[n_frc];
324             for (int i = 0; i < n_frc; ++i){
325                     FrcCtrl_CL[i]=ConceptionalLayer(5,n_tv);
326                     FrcCtrl_CL[i].set_dtstep(dtstep);
327                     FrcCtrl_CL[i].set_PoseNames(*tpname);
328                     FrcCtrl_CL[i].set_CLIndex(i);
329                     FrcCtrl_CL[i].set_npos(n_frc);
330                     FrcCtrl_CL[i].set_CL_Folder("datacontrol/Layers/Force_Control/"+ frcnames[i] +"/
                            ConceptionalLayer/");
331                     FrcCtrl_CL[i].set_cs_file("ForceTriads.dat");
332                     FrcCtrl_CL[i].set_PID_Folder("Force_PID/");
333                     FrcCtrl_CL[i].set_IDc("f");
334                     FrcCtrl_CL[i].set_CALC_JACS(CALC_JAC_F);
335                     if(CALC_JAC_F){ //
336                     }else{
```

```
337                             FrcCtrl_CL[i].set_Jac_Folder("Force_Jacobian_JQF/");
338                             FrcCtrl_CL[i].set_Jac_sfx("jqf");
339                             FrcCtrl_CL[i].set_os_file("Force_Jacobian_JQF/JointVector.dat");
340                         }
341                         FrcCtrl_CL[i].Init_CL_XF();
342                         Eigen::VectorXd Frc_oVar_vec=FrcCtrl_CL[i].get_oVar_vec();
343                         Eigen::VectorXd Frc_oVar_global=FrcCtrl_CL[i].get_global_oVars();
344                         for (int i = 0; i < n_dof; ++i)if (Frc_oVar_global(i)>=common_cDoF_vec(i)) common_cDoF_vec(i)=
                                Frc_oVar_global(i);
345                         FrcCtrl_TL[i]= TransformationalLayer(FrcCtrl_CL[i].get_ControlSpace(), n_tv);
346                         FrcCtrl_TL[i].set_dtstep(dtstep);
347                         FrcCtrl_TL[i].set_PoseNames(*tpname);
348                         Eigen::VectorXd oVarVecCL=FrcCtrl_CL[i].get_oVar_vec();
349                         FrcCtrl_TL[i].set_cDoFvec_CL(oVarVecCL);
350                         FrcCtrl_TL[i].set_npos(n_frc);
351                         FrcCtrl_TL[i].set_nmtu(nmusc);
352                         FrcCtrl_TL[i].set_MtuNames(mnames);
353                         FrcCtrl_TL[i].set_TLIndex(i);
354                         FrcCtrl_TL[i].set_TL_Folder("datacontrol/Layers/Force_Control/"+ frcnames[i] +"/
                                TransformationalLayer/"); //
355                         FrcCtrl_TL[i].set_cDoF_file( "JointVector.dat");
356                         FrcCtrl_TL[i].set_mVec_file("Jacobians/MuscleVector.dat");
357                         FrcCtrl_TL[i].set_IDc("f");
358                         FrcCtrl_TL[i].set_PID_Tht_Folder("PID/Theta_PID/");
359                         FrcCtrl_TL[i].set_PID_Tht_Lbd_Folder("PID/Theta_Lambda_PID/");
360                         FrcCtrl_TL[i].set_CALC_JACS(CALC_JAC_Q);
361                         if(CALC_JAC_Q){
362                             FrcCtrl_TL[i].set_Mamat_glb(Ma_mat);
363                             FrcCtrl_TL[i].set_dLseedLce_glb(dLseedLce_mat);
364                             FrcCtrl_TL[i].set_Lcevec_glb(Lce_vec);
365                             FrcCtrl_TL[i].set_Uvec_glb(uinit);
366                         }else{
367                             FrcCtrl_TL[i].set_Jac_U_Tht_Folder("Jacobians/Joint_Stimulation_Jacobian_JUQ/");
368                             FrcCtrl_TL[i].set_Jac_Lbd_Tht_Folder("Jacobians/Joint_Length_Jacobian_JLQ/");
369                         }
370                         FrcCtrl_TL[i].Init_TL();
371                         Eigen::VectorXd Frc_mVec_TL=FrcCtrl_TL[i].get_mVec_TL();
372                         FrcCtrl_SL[i]=StructuralLayer(FrcCtrl_CL[i].get_ControlSpace(), n_tv);
373                         FrcCtrl_SL[i].set_PoseNames(*tpname);
374                         FrcCtrl_SL[i].set_mVec_TL(Frc_mVec_TL);
375                         FrcCtrl_SL[i].set_nmtu(nmusc);
376                         FrcCtrl_SL[i].set_MtuNames(mnames);
377                         FrcCtrl_SL[i].set_ICEopt(ICEopt);
378                         FrcCtrl_SL[i].set_IDc("f");
379                         FrcCtrl_SL[i].set_SL_Folder("datacontrol/Layers/Force_Control/"+frcnames[i]+"/StructuralLayer/"
                                );
380                         FrcCtrl_SL[i].set_mVec_file( "MuscleVector.dat");
381                         FrcCtrl_SL[i].set_UcocRef_Folder("Alpha_Controller/");
382                         FrcCtrl_SL[i].set_UcocRef0(u__open);
383                         FrcCtrl_SL[i].set_Sigma0(sigma);
384                         FrcCtrl_SL[i].set_Kappa_Folder("Lambda_Controller/Kappa/");
385                         FrcCtrl_SL[i].set_Kappa0(kappa);
386                         FrcCtrl_SL[i].Init_SL();
387                 }
388         }
389         n_ctr=n_lbd+n_ang+n_trq+n_pos+n_frc;
390         cout << "Info(): Total number of controllers: " << n_ctr << endl;
391         cout << "       Lambda Controller: " << n_lbd << endl;
392         cout << "       Angle Controller: " << n_ang << endl;
393         cout << "       Torque Controller: " << n_trq << endl;
394         cout << "       Position Controller: " << n_pos << endl;
395         cout << "       Force Controller: " << n_frc << endl;
396 };
```

Listing E.4: Function to update layer variables

```
1
2 void MotorControl::UpdateLayers(double t, double* z, double* zd){
3         static bool new_cstate=true;
4         static int old_ControlState=-1;
5
6         if(event_flag==0) ControlState=getTControlState(t);
7         if (ControlState!=old_ControlState){
8         new_cstate=true;
9         }else new_cstate=false;
10        old_ControlState=ControlState;
11
12        if (STRUCTURAL_LAYER) getMuscleState(n_dof, n_musc, z, zd);
13
14        if(LAMBDA_CONTROL){
15                for (int i = 0; i < n_lbd; ++i){
16                if(dbg_updateLayers) cout << "////////////////////////////////////////////////////" << endl;
17                if(dbg_updateLayers) cout << "/////////          Updating LAMBDA CONTROL" << endl;
18                if(dbg_updateLayers) cout << "////////////////////////////////////////////////////" << endl;
19                if(dbg_updateLayers) cout << "///////// # " << i << endl;
20                        if(new_cstate){
21                        LbdCtrl_SL[i].set_ControlState(ControlState);
22                        LbdCtrl_SL[i].Update_CtrlVars();
23                        }
24                        LbdCtrl_SL[i].set_lCE(current_lce);
25                        LbdCtrl_SL[i].set_vCE(current_vce);
26                }
27        }
28
29        if(ANGLE_CONTROL){
30                for (int i = 0; i < n_ang; ++i){
31                if(dbg_updateLayers) cout << "////////////////////////////////////////////////////" << endl;
32                if(dbg_updateLayers) cout << "/////////          Updating ANGLE CONTROL" << endl;
33                if(dbg_updateLayers) cout << "////////////////////////////////////////////////////" << endl;
34                if(dbg_updateLayers) cout << "///////// # " << i << endl;
35                        if(new_cstate){
36                                AngCtrl_CL[i].set_ControlState(ControlState);
37                                AngCtrl_TL[i].set_ControlState(ControlState);
38                                AngCtrl_SL[i].set_ControlState(ControlState);
39                        }
40                        AngCtrl_CL[i].Update_CtrlVars();
41                        AngCtrl_TL[i].Update_CtrlVars();
42                        AngCtrl_SL[i].Update_CtrlVars();
43                        AngCtrl_SL[i].set_lCE(current_lce);
44                        AngCtrl_SL[i].set_vCE(current_vce);
45                        if(AngCtrl_TL[i].get_CALC_JACS()){
46                                AngCtrl_TL[i].set_Mamat_lcl(Ma_mat);
47                                AngCtrl_TL[i].set_dLseedLce_lcl(dLseedLce_mat);
48                                AngCtrl_TL[i].set_Lcevec_lcl(current_lce);
49                                Eigen::VectorXd Ucoc_tmp= AngCtrl_SL[i].get_Ucoc();
50                                AngCtrl_TL[i].set_Uvec_lcl(Ucoc_tmp);
51                                AngCtrl_TL[i].calc_J_Lbd_Tht();
52                                AngCtrl_TL[i].calc_J_U_Tht();
53                        }
54                }
55        }
56
57        if(TORQUE_CONTROL){
58                for (int i = 0; i < n_trq; ++i){
59                        if(dbg_updateLayers) cout << "////////////////////////////////////////////////////" << endl;
60                        if(dbg_updateLayers) cout << "/////////          Updating TORQUE CONTROL" << endl;
61                        if(dbg_updateLayers) cout << "////////////////////////////////////////////////////" << endl;
62                        if(dbg_updateLayers) cout << "///////// # " << i << endl;
63                        if(new_cstate){
64                                TrqCtrl_CL[i].set_ControlState(ControlState);
65                                TrqCtrl_TL[i].set_ControlState(ControlState);
66                                TrqCtrl_SL[i].set_ControlState(ControlState);
67                        }
68                        TrqCtrl_CL[i].Update_CtrlVars();
69                        TrqCtrl_TL[i].Update_CtrlVars();
70                        TrqCtrl_SL[i].Update_CtrlVars();
71                        TrqCtrl_SL[i].set_lCE(current_lce);
72                        TrqCtrl_SL[i].set_vCE(current_vce);
73                        if(TrqCtrl_CL[i].get_CALC_JACS()){
74                                TrqCtrl_CL[i].set_Mamat_lcl(Ma_mat);
75                                TrqCtrl_CL[i].set_dLseedLce_lcl(dLseedLce_mat);
76                                TrqCtrl_CL[i].set_dFcedLce_lcl(dFcedLce_mat);
77                                TrqCtrl_CL[i].calc_J_Tht_Trq();
78                                TrqCtrl_TL[i].Update_Tht_is();
79                                Eigen::VectorXd Thtis_tmp= TrqCtrl_TL[i].get_Tht_is_glb();
80                                TrqCtrl_CL[i].set_oVar_is_lcl(Thtis_tmp);
81                        }
82                        if(TrqCtrl_TL[i].get_CALC_JACS()){
83                                TrqCtrl_TL[i].set_Mamat_lcl(Ma_mat);
84                                TrqCtrl_TL[i].set_dLseedLce_lcl(dLseedLce_mat);
85                                TrqCtrl_TL[i].set_Lcevec_lcl(current_lce);
86                                Eigen::VectorXd Ucoc_tmp= TrqCtrl_SL[i].get_Ucoc();
87                                TrqCtrl_TL[i].set_Uvec_lcl(Ucoc_tmp);
88                                TrqCtrl_TL[i].calc_J_Lbd_Tht();
```

```
89                              TrqCtrl_TL[i].calc_J_U_Tht();
90                      }
91              }
92      }
93
94      if(POSITION_CONTROL){
95              for (int i = 0; i < n_pos; ++i){
96                      if(dbg_updateLayers) cout << "////////////////////////////////////////////////////" << endl;
97                      if(dbg_updateLayers) cout << "/////////  Updating POSITION CONTROL" << endl;
98                      if(dbg_updateLayers) cout << "////////////////////////////////////////////////////" << endl;
99                      if(dbg_updateLayers) cout << "///////// # " << i << endl;
100                     if(new_cstate){
101                             PosCtrl_CL[i].set_ControlState(ControlState);
102                             PosCtrl_TL[i].set_ControlState(ControlState);
103                             PosCtrl_SL[i].set_ControlState(ControlState);
104                     }
105                     PosCtrl_CL[i].Update_CtrlVars();
106                     PosCtrl_TL[i].Update_CtrlVars();
107                     PosCtrl_SL[i].Update_CtrlVars();
108                     PosCtrl_SL[i].set_lCE(current_lce);
109                     PosCtrl_SL[i].set_vCE(current_vce);
110                     if(PosCtrl_CL[i].get_CALC_JACS()){
111                             Eigen::MatrixXd TESTJAC;
112                             PosCtrl_CL[i].calc_J_Pos_Ang_mat(TESTJAC);
113                             PosCtrl_CL[i].calc_J_Ang_Pos_mat(TESTJAC);
114                             PosCtrl_TL[i].Update_Tht_is();
115                             Eigen::VectorXd ThtPosis_tmp= PosCtrl_TL[i].get_Tht_is_glb();
116                             Eigen::MatrixXd PermPosDof_tmp= PosCtrl_CL[i].get_Perm_os_Mat_Mat();
117                             PosCtrl_CL[i].set_oVar_is_lcl(ThtPosis_tmp);
118                     }
119                     if(PosCtrl_TL[i].get_CALC_JACS()){
120                             PosCtrl_TL[i].set_Mamat_lcl(Ma_mat);
121                             PosCtrl_TL[i].set_dLseedLce_lcl(dLseedLce_mat);
122                             PosCtrl_TL[i].set_Lcevec_lcl(current_lce);
123                             Eigen::VectorXd Ucoc_tmp= PosCtrl_SL[i].get_Ucoc();
124                             PosCtrl_TL[i].set_Uvec_lcl(Ucoc_tmp);
125                             PosCtrl_TL[i].calc_J_Lbd_Tht();
126                             PosCtrl_TL[i].calc_J_U_Tht();
127                     }
128             }
129     }
130
131     if(FORCE_CONTROL){
132             for (int i = 0; i < n_frc; ++i){
133                     if(dbg_updateLayers) cout << "////////////////////////////////////////////////////" << endl;
134                     if(dbg_updateLayers) cout << "/////////  Updating FORCE CONTROL" << endl;
135                     if(dbg_updateLayers) cout << "////////////////////////////////////////////////////" << endl;
136                     if(dbg_updateLayers) cout << "///////// # " << i << endl;
137                     if(new_cstate){
138                             FrcCtrl_CL[i].set_ControlState(ControlState);
139                             FrcCtrl_TL[i].set_ControlState(ControlState);
140                             FrcCtrl_SL[i].set_ControlState(ControlState);
141                     }
142             }
143     }
144     return;
145 }
```

Listing E.5: Function to update total muscle stimulation

```
Eigen::VectorXd MotorControl::UpdateStim( double t, int ndof, int nmusc, double* z, double* zd){
        Stim_Lbd_mat = Eigen::MatrixXd::Zero(nmusc, n_ctr);
        Stim_Tht_mat = Eigen::MatrixXd::Zero(nmusc, n_ctr);
        Stim_Coc_mat = Eigen::MatrixXd::Zero(nmusc, n_ctr);
        Stim_Tht_Coc_mat = Eigen::MatrixXd::Zero(nmusc, n_ctr);

        static double t_old=t;
        static double dt=0;
        static int init=0;

        dt=t-t_old;
        if (dt>=dtstep) t_old=t;

        if(LAMBDA_CONTROL){
                for (int i = 0; i < n_lbd; ++i){
                        cout << "////////////////////////////////////////////////////" << endl;
                        cout << "/////////  LAMBDA CONTROL STIMULATION" << endl;
                        cout << "////////////////////////////////////////////////////" << endl;
                        cout << "/////////  #" << i << endl;
                        Stim_Lbd_mat.col(i)=LbdCtrl_SL[i].Update_U_Lbd();
                        Stim_Coc_mat.col(i)=LbdCtrl_SL[i].get_Ucoc();
                }
        }

        if(ANGLE_CONTROL){
                for (int i = 0; i < n_ang; ++i){
                        cout << "////////////////////////////////////////////////////" << endl;
                        cout << "/////////  ANGLE CONTROL STIMULATION" << endl;
                        cout << "////////////////////////////////////////////////////" << endl;
                        cout << "/////////  #" << i << endl;
                        AngCtrl_CL[i].Update_oVar();
                        Eigen::VectorXd Tht_des_global=AngCtrl_CL[i].get_global_oVar();
                        AngCtrl_TL[i].set_Tht_des(Tht_des_global);
                        AngCtrl_TL[i].Update_Tht_is();
                        AngCtrl_TL[i].Update_Tht_d();
                        AngCtrl_TL[i].Update_Tht_err();
                        AngCtrl_TL[i].Update_Tht_i(dt);
                        AngCtrl_TL[i].Update_Tht_pid();
                        AngCtrl_TL[i].Update_Tht_Lbd_pid();
                        Eigen::VectorXd Lbdtht_tmp=     AngCtrl_TL[i].Update_Lbd_Tht();
                        Eigen::VectorXd Utht_tmp =AngCtrl_TL[i].Update_U_Tht();
                        Eigen::VectorXd Uthtcoc_tmp=AngCtrl_TL[i].Update_U_Tht_Coc();
                        Stim_Tht_mat.col(n_lbd+i) = Utht_tmp;
                        Stim_Tht_Coc_mat.col(n_lbd+i)=Uthtcoc_tmp;
                        AngCtrl_SL[i].set_Utht(Utht_tmp);
                        AngCtrl_SL[i].set_Uthtcoc(Uthtcoc_tmp);
                        AngCtrl_SL[i].set_Lambda(Lbdtht_tmp);
                        Stim_Lbd_mat.col(n_lbd+i)=AngCtrl_SL[i].Update_U_Lbd();
                        Stim_Coc_mat.col(n_lbd+i)=AngCtrl_SL[i].get_Ucoc();
                }
        }

        if(TORQUE_CONTROL){
                for (int i = 0; i < n_trq; ++i){
                        cout << "////////////////////////////////////////////////////" << endl;
                        cout << "/////////  TORQUE CONTROL STIMULATION" << endl;
                        cout << "////////////////////////////////////////////////////" << endl;
                        cout << "/////////  #" << i << endl;
                        TrqCtrl_CL[i].set_Fmtu_lcl(Fmtu_vec);
                        TrqCtrl_CL[i].Update_Trq_is();
                        TrqCtrl_CL[i].Update_cVar_err();
                        TrqCtrl_CL[i].Update_cVar_d(dt);
                        TrqCtrl_CL[i].Update_cVar_d_err();
                        TrqCtrl_CL[i].Update_cVar_i(dt);
                        TrqCtrl_CL[i].Update_cVar_pid();
                        TrqCtrl_CL[i].Update_oVar();
                        Eigen::VectorXd Tht_des_global=TrqCtrl_CL[i].get_global_oVar();
                        TrqCtrl_TL[i].set_Tht_des(Tht_des_global);
                        TrqCtrl_TL[i].Update_Tht_d();
                        TrqCtrl_TL[i].Update_Tht_err();
                        TrqCtrl_TL[i].Update_Tht_i(dt);
                        TrqCtrl_TL[i].Update_Tht_pid();
                        TrqCtrl_TL[i].Update_Tht_Lbd_pid();
                        Eigen::VectorXd Lbdthttrq_tmp=  TrqCtrl_TL[i].Update_Lbd_Tht();
                        Eigen::VectorXd Utht_tmp = TrqCtrl_TL[i].Update_U_Tht();
                        Eigen::VectorXd Uthtcoc_tmp = TrqCtrl_TL[i].Update_U_Tht_Coc();
                        Stim_Tht_mat.col(n_lbd+n_ang+i) = Utht_tmp;
                        Stim_Tht_Coc_mat.col(n_lbd+n_ang+i)=Uthtcoc_tmp;
                        TrqCtrl_SL[i].set_Utht(Utht_tmp);
                        TrqCtrl_SL[i].set_Uthtcoc(Uthtcoc_tmp);
                        TrqCtrl_SL[i].set_Lambda(Lbdthttrq_tmp);
                        Stim_Lbd_mat.col(n_lbd+n_ang+i)=TrqCtrl_SL[i].Update_U_Lbd();
                        Stim_Coc_mat.col(n_lbd+n_ang+i)=TrqCtrl_SL[i].get_Ucoc();
                }
        }

        if(POSITION_CONTROL){
                for (int i = 0; i < n_pos; ++i){
```

```
89                      cout << "////////////////////////////////////////////////////" << endl;
90                      cout << "//////// POSITION CONTROL STIMULATION" << endl;
91                      cout << "////////////////////////////////////////////////////" << endl;
92                      cout << "//////// #" << i << endl;
93                      PosCtrl_CL[i].Update_Pos_is();
94                      PosCtrl_CL[i].Update_Pos_des();
95                      PosCtrl_CL[i].Update_cVar_err();
96                      PosCtrl_CL[i].Update_cVar_d(dt);
97                      PosCtrl_CL[i].Update_cVar_d_err();
98                      PosCtrl_CL[i].Update_cVar_i(dt);
99                      PosCtrl_CL[i].Update_cVar_pid();
100                     PosCtrl_CL[i].Update_oVar();
101                     Eigen::MatrixXd Perm_tmp=PosCtrl_CL[i].get_Perm_os_Mat_Mat();
102                     Eigen::VectorXd Tht_des_global=PosCtrl_CL[i].get_global_oVar();
103                     PosCtrl_TL[i].set_Tht_des(Tht_des_global);
104                     PosCtrl_TL[i].Update_Tht_d();
105                     PosCtrl_TL[i].Update_Tht_err();
106                     PosCtrl_TL[i].Update_Tht_i(dt);
107                     PosCtrl_TL[i].Update_Tht_pid();
108                     PosCtrl_TL[i].Update_Tht_Lbd_pid();
109                     Eigen::VectorXd Lbdthtpos_tmp=  PosCtrl_TL[i].Update_Lbd_Tht();
110                     Eigen::VectorXd Utht_tmp = PosCtrl_TL[i].Update_U_Tht();
111                     Eigen::VectorXd Uthtcoc_tmp=PosCtrl_TL[i].Update_U_Tht_Coc();
112                     Stim_Tht_mat.col(n_lbd+n_ang+n_trq+i) = Utht_tmp;
113                     Stim_Tht_Coc_mat.col(n_lbd+n_ang+n_trq+i)=Uthtcoc_tmp;
114                     PosCtrl_SL[i].set_Utht(Utht_tmp);
115                     PosCtrl_SL[i].set_Uthtcoc(Uthtcoc_tmp);
116                     PosCtrl_SL[i].set_Lambda(Lbdthtpos_tmp);
117                     Stim_Lbd_mat.col(n_lbd+n_ang+n_trq+i)=PosCtrl_SL[i].Update_U_Lbd();
118                     Stim_Coc_mat.col(n_lbd+n_ang+n_trq+i)=PosCtrl_SL[i].get_Ucoc();
119             }
120     }
121
122     if(FORCE_CONTROL){
123             for (int i = 0; i < n_frc; ++i){
124                     cout << "////////////////////////////////////////////////////" << endl;
125                     cout << "//////// FORCE CONTROL  STIMULATION" << endl;
126                     cout << "////////////////////////////////////////////////////" << endl;
127                     cout << "//////// #" << i << endl;
128             }
129     }
130
131     Eigen::VectorXd U_Lbd=Eigen::VectorXd::Zero(nmusc);
132     Eigen::VectorXd U_Tht=Eigen::VectorXd::Zero(nmusc);
133     Eigen::VectorXd U_coc=Eigen::VectorXd::Zero(nmusc);
134     Eigen::VectorXd U_Tht_coc=Eigen::VectorXd::Zero(nmusc);
135
136     Eigen::VectorXd U_total=Eigen::VectorXd::Zero(nmusc);
137
138     for (int i = 0; i < n_ctr; ++i){
139             U_Lbd += Stim_Lbd_mat.col(i);///22.5;
140             U_Tht += Stim_Tht_mat.col(i);
141             U_coc += Stim_Coc_mat.col(i);
142             U_Tht_coc += Stim_Tht_Coc_mat.col(i);
143     }
144
145     U_total=U_Lbd+U_Tht+U_coc+U_Tht_coc;
146
147     if(noiseflag){
148             Eigen::VectorXd Onesnmusc= Eigen::VectorXd::Ones(nmusc);
149             U_total.array()=U_total.array()*(Onesnmusc.array()+U_noise.array());
150     }
151
152     for (int i = 0; i < nmusc; ++i){
153             if(U_total(i) <= 0.01) U_total(i) = 0.01; //
154             else if(U_total(i) >= 1.0) U_total(i) = 1.0;
155     }
156
157     return U_total;
158 }
```

Listing E.6: modified Dijkstra's algorithm for the calculation of the adjacency matrix and the moment arms

```
1
2 Eigen::VectorXd MotorControl::Dijkstra(Eigen::MatrixXd A, int src, int dest){
3         int V=A.rows();
4         Eigen::VectorXd dist;
5         dist=Eigen::VectorXd::Zero(V);
6         Eigen::VectorXd sptSet;
7         sptSet=Eigen::VectorXd::Zero(V);
8         Eigen::VectorXd parent;
9         parent=Eigen::VectorXd::Zero(V);
10    for (int i = 0; i < V; i++)
11    {
12        parent(src) = -1;
13        dist(i) = INT_MAX;
14        sptSet(i) = 0;
15    }
16    dist(src) = 0;
17    for (int count = 0; count < V - 1; count++)
18    {
19        int u = DijkstraMinDist(dist, sptSet, V);
20        sptSet(u) = 1;
21        for (int v = 0; v < V; v++)
22            if (sptSet(v)==0 && A(u,v) &&
23                dist(u) + 1 < dist(v))
24            {
25                parent(v) = u;
26                dist(v) = dist(u) + 1;
27            }
28    }
29        int nJoints=A.maxCoeff()+1;
30    Eigen::VectorXd corrvectest, corrtest;
31        corrvectest=Eigen::VectorXd::Zero(nJoints);
32        corrvectest=DijkstraSolution(dist, V, parent, src, dest, corrvectest, A);
33         return corrvectest;
34 }
35
36 Eigen::VectorXd MotorControl::DijkstraPath(Eigen::VectorXd parent, int src, Eigen::VectorXd corrvectest, Eigen::
       MatrixXd A){
37        if(parent(src) ==- 1) return corrvectest;
38        if(parent(src) <= src) corrvectest(A(parent(src),src))+=1;
39        if(parent(src) >= src) corrvectest(A(parent(src),src))-=1;
40        corrvectest=DijkstraPath(parent, parent(src), corrvectest, A);
41        return corrvectest;
42 }
43
44
45 Eigen::VectorXd MotorControl::DijkstraSolution(Eigen::VectorXd dist, int n, Eigen::VectorXd parent, int src, int dest,
       Eigen::VectorXd corrvectest, Eigen::MatrixXd A){
46    int i=dest;
47    corrvectest=DijkstraPath(parent, i, corrvectest, A);
48    return corrvectest;
49 }
50
51 int MotorControl::DijkstraMinDist(Eigen::VectorXd dist, Eigen::VectorXd sptSet, int n) {
52    int min = INT_MAX;
53    int min_index=0;
54    for (int v = 0; v < n; v++)
55        if (sptSet(v) == 0 && dist(v) <= min)
56            min = dist(v), min_index = v;
57
58    return min_index;
59 }
```

E.5.3 Functions in `uconclayer.cpp`

Listing E.7: Conceptional layer initialisation function for angle and torque control

```
1
2 void ConceptionalLayer::Init_CL_QT(){
3     ControlState_old=-1;
4     ControlState=0;
5         cout << "////////////////////////////////////////////////////" << endl;
6         cout << "////////INIT CONCEPTIONAL LAYER ANG/TRQ"<< endl;
7         cout << "////////////////////////////////////////////////////"<< endl;
8
9         n_dof=model.dof();
10        n_cs= read_cs_file(cVar_vec, CL_Folder + cs_file);
11        n_os=n_cs;
12        oVar_vec=cVar_vec;
13        n_cdof=n_cs;
14        Perm_cDoF_Mat=Eigen::MatrixXd::Zero(n_cs,n_dof);
15        Perm_cDoF_Mat = calc_cDoFPermutation(cVar_vec);
16        cVar_des_array=new double**[n_tv];
17        for (int i = 0; i < n_tv; i++){
18                cVar_des_array[i]=new double*[n_cs];
19                for(int j=0; j<n_cs;j++){
20                        cVar_des_array[i][j]=new double[1];
21                }
22        }
23        csdesname = new std::string[n_tv];
24        read_cs_des_file();
25        cVar_dess = new Eigen::VectorXd[n_tv];
26        for (int i = 0; i < n_tv; ++i){
27                cVar_dess[i]=Eigen::VectorXd::Zero(n_cs);
28                cVar_dess[i]=myMotControl.ConvertToEigenMatrix(n_cs,1,cVar_des_array[i]);
29        }
30        if (ControlSpace==3)
31        {
32                P_CLarrd=new double**[n_tv];
33                D_CLarrd=new double**[n_tv];
34                I_CLarrd=new double**[n_tv];
35                for (int i = 0; i < n_tv; i++){
36                        P_CLarrd[i]=new double*[n_cs];
37                        D_CLarrd[i]=new double*[n_cs];
38                        I_CLarrd[i]=new double*[n_cs];
39                        for(int j=0; j<n_cs;j++){
40                                P_CLarrd[i][j]=new double[n_cs];
41                                D_CLarrd[i][j]=new double[n_cs];
42                                I_CLarrd[i][j]=new double[n_cs];
43                        }
44                }
45                read_PIDFiles();
46                P_CLs = new Eigen::MatrixXd[n_tv];
47                I_CLs = new Eigen::MatrixXd[n_tv];
48                D_CLs = new Eigen::MatrixXd[n_tv];
49                for (int i = 0; i < n_tv; ++i){
50                        P_CLs[i]=Eigen::MatrixXd::Zero(n_cs,n_cs);
51                        I_CLs[i]=Eigen::MatrixXd::Zero(n_cs,n_cs);
52                        D_CLs[i]=Eigen::MatrixXd::Zero(n_cs,n_cs);
53                        P_CLs[i]=myMotControl.ConvertToEigenMatrix(n_cs,n_cs,P_CLarrd[i]);
54                        I_CLs[i]=myMotControl.ConvertToEigenMatrix(n_cs,n_cs,I_CLarrd[i]);
55                        D_CLs[i]=myMotControl.ConvertToEigenMatrix(n_cs,n_cs,D_CLarrd[i]);
56                }
57                if(CALC_JACS){
58                        Eigen::MatrixXd Ma_mat_tmp;
59                        Ma_mat_tmp=Ma_mat_glb * Perm_cDoF_Mat.transpose();
60                    mVec_CL_global = Eigen::VectorXd::Zero(n_mtu);
61                    mVec_CL = Eigen::VectorXd::Zero(n_mtu);
62                    n_cmtu = 0;
63                    for (int i = 0; i < n_mtu; ++i){
64                        if((Ma_mat_tmp.row(i)).norm()!=0){
65                         mVec_CL_global(i)=i+1;
66                         mVec_CL(n_cmtu)=i+1;
67                         n_cmtu++;
68                        }
69                    }
70                    mVec_CL.conservativeResize(n_cmtu);
71                    Perm_cMtu_Mat_CL = Eigen::MatrixXd::Zero(n_cmtu,n_mtu);
72                        Perm_cMtu_Mat_CL = calc_cMtuPermutation(mVec_CL);
73                        Ma_mat_lcl=Perm_cMtu_Mat_CL*Ma_mat_tmp;
74                    dLseedLce_mat_lcl=Perm_cMtu_Mat_CL*dLseedLce_mat_glb*Perm_cMtu_Mat_CL.transpose();
75                    dFcedLce_mat_lcl=Perm_cMtu_Mat_CL*dFcedLce_mat_glb*Perm_cMtu_Mat_CL.transpose();
76                J_os_cs=Eigen::MatrixXd::Zero(n_os,n_cs);
77                    calc_J_Tht_Trq();
78                }else{
79                        J_os_csarrd=new double**[n_tv];
80                        for (int i = 0; i < n_tv; i++){
81                                J_os_csarrd[i]=new double*[n_os];
82                                for(int j=0; j<n_os;j++) J_os_csarrd[i][j]=new double[n_cs];
83                        }
84                        read_JacFiles();
```

```
85                J_os_cs=Eigen::MatrixXd::Zero(n_cs,n_cs);
86                        J_os_css = new Eigen::MatrixXd[n_tv];
87                        for (int i = 0; i < n_tv; ++i){
88                                J_os_css[i]=Eigen::MatrixXd::Zero(n_cs,n_cs);
89                                J_os_css[i]=myMotControl.ConvertToEigenMatrix(n_cs,n_cs,J_os_csarrd[i]);
90                        }
91                }
92                P_CL.resize(n_cs,n_cs);
93                D_CL.resize(n_cs,n_cs);
94                I_CL.resize(n_cs,n_cs);
95                P_CL =Eigen::MatrixXd::Zero(n_cs,n_cs);
96                D_CL =Eigen::MatrixXd::Zero(n_cs,n_cs);
97                I_CL =Eigen::MatrixXd::Zero(n_cs,n_cs);
98        }
99        sensInFlagJC.resize(n_cs,1);
100       sensInFlagJC=Eigen::VectorXd::Zero(n_cs);
101       cVar_is.resize(n_cs,1);
102       cVar_des.resize(n_cs,1);
103       cVar_err.resize(n_cs,1);
104       cVar_d_err.resize(n_cs,1);
105       cVar_d_des.resize(n_cs,1);
106       cVar_pid.resize(n_cs,1);
107       cVar_i.resize(n_cs,1);
108       cVar_d.resize(n_cs,1);
109       cVar_old.resize(n_cs,1);
110       oVar_is_lcl.resize(n_os,1);
111       cVar_is=Eigen::VectorXd::Zero(n_cs);
112       cVar_des=Eigen::VectorXd::Zero(n_cs);
113       cVar_err=Eigen::VectorXd::Zero(n_cs);
114       cVar_d_err=Eigen::VectorXd::Zero(n_cs);;
115       cVar_d_des=Eigen::VectorXd::Zero(n_cs);;
116       cVar_i=Eigen::VectorXd::Zero(n_cs);
117       cVar_d=Eigen::VectorXd::Zero(n_cs);
118       cVar_old = Eigen::VectorXd::Zero(n_cs);
119       oVar_cVar=Eigen::VectorXd(n_os);
120       oVar_is_lcl=Eigen::VectorXd(n_os);
121       Perm_os_Mat = Eigen::MatrixXd::Zero(n_os, n_dof);
122       Perm_os_Mat = calc_os_Permutation();
123
124       return;
125 };
```

Listing E.8: Conceptional layer initialisation function for position and force control

```
1
2 void ConceptionalLayer::Init_CL_XF(){
3     ControlState_old=-1;
4         cout << "////////////////////////////////////////////////////" << endl;
5         cout << "////////INIT CONCEPTIONAL LAYER"<< endl;
6         cout << "////////////////////////////////////////////////////"<< endl;
7         n_dof=model.dof();
8         n_cs = read_TrdFile(CL_Folder + cs_file);        //n_cs=6
9         cVar_vec=Eigen::VectorXd::Zero(n_cs);
10        const Body * cBdy = model.triad(cTrd_ID)->parent();
11        const Body * rBdy = model.triad(rTrd_ID)->parent();
12        const Body * tBdy;
13        int cBdy_ID=0;
14        int rBdy_ID=0;
15        int ncountbodies=1;
16        for (int i = 0; i < ncountbodies; ++i){
17                tBdy=model.body(i);
18                if(cBdy->name()==tBdy->name()) cBdy_ID=i;
19                else ncountbodies++;
20        }
21        ncountbodies=1;
22        for (int i = 0; i < ncountbodies; ++i){
23                        tBdy=model.body(i);
24                        if(rBdy->name()==tBdy->name()) rBdy_ID=i;
25                        else ncountbodies++;
26        }
27        rM_mat=Eigen::Matrix4d::Identity();
28        rM_Bdy_mat=Eigen::Matrix4d::Identity();
29        rM_Trd_mat=Eigen::Matrix4d::Identity();
30        getTriadM(rTrd_ID,  rM_Trd_mat);
31        getTriadParentBodyM(rTrd_ID, rM_Bdy_mat);
32        rM_mat=rM_Bdy_mat*rM_Trd_mat;
33        cM_mat=Eigen::Matrix4d::Identity();
34        cM_Bdy_mat=Eigen::Matrix4d::Identity();
35        cM_Trd_mat=Eigen::Matrix4d::Identity();
36        getTriadM(cTrd_ID,  cM_Trd_mat);
37        getTriadParentBodyM(cTrd_ID, cM_Bdy_mat);
38        cM_mat=rM_mat*(cM_Bdy_mat*cM_Trd_mat);
39        cTrd_des_IDs=new int[n_tv];
40        csdesname = new std::string[n_tv];
41        read_Trd_des_file();
42        cTrd_des_ID=cTrd_des_IDs[0];
43        cTrd_des_ID_old=cTrd_des_ID;
44        cM_des_mat=Eigen::Matrix4d::Identity();
```

```
45        cM_Bdy_des_mat=Eigen::Matrix4d::Identity();
46        cM_Trd_des_mat=Eigen::Matrix4d::Identity();
47        getTriadM(cTrd_des_ID,  cM_Trd_des_mat);
48        getTriadParentBodyM(cTrd_des_ID, cM_Bdy_des_mat);
49        cM_des_mat=cM_Bdy_des_mat*cM_Trd_des_mat;
50        P_CLarrd=new double**[n_tv];
51        D_CLarrd=new double**[n_tv];
52        I_CLarrd=new double**[n_tv];
53        for (int i = 0; i < n_tv; i++){
54                P_CLarrd[i]=new double*[n_cs];
55                D_CLarrd[i]=new double*[n_cs];
56                I_CLarrd[i]=new double*[n_cs];
57                for(int j=0; j<n_cs;j++){
58                        P_CLarrd[i][j]=new double[n_cs];
59                        D_CLarrd[i][j]=new double[n_cs];
60                        I_CLarrd[i][j]=new double[n_cs];
61                }
62        }
63        read_PIDFiles();
64        P_CLs = new Eigen::MatrixXd[n_tv];
65        I_CLs = new Eigen::MatrixXd[n_tv];
66        D_CLs = new Eigen::MatrixXd[n_tv];
67        for (int i = 0; i < n_tv; ++i){
68                P_CLs[i]=Eigen::MatrixXd::Zero(n_cs,n_cs);
69                I_CLs[i]=Eigen::MatrixXd::Zero(n_cs,n_cs);
70                D_CLs[i]=Eigen::MatrixXd::Zero(n_cs,n_cs);
71                P_CLs[i]=myMotControl.ConvertToEigenMatrix(n_cs,n_cs,P_CLarrd[i]);
72                I_CLs[i]=myMotControl.ConvertToEigenMatrix(n_cs,n_cs,I_CLarrd[i]);
73                D_CLs[i]=myMotControl.ConvertToEigenMatrix(n_cs,n_cs,D_CLarrd[i]);
74        }
75
76        if(CALC_JACS){
77                Eigen::MatrixXd Adj_mat;
78                Adj_mat=myMotControl.get_Adj_mat();
79                int n_mdljnts= Adj_mat.maxCoeff()+1;
80                Eigen::VectorXd Path_r_c = Eigen::VectorXd::Zero(n_mdljnts);
81                Path_r_c=myMotControl.Dijkstra(Adj_mat, rBdy_ID, cBdy_ID);
82                DofToJoint = Eigen::VectorXd::Zero(n_dof);
83                  int count_jdof=0;
84            for (int ii = 0; ii < n_mdljnts; ++ii){
85                int n_jdof=model.joint(ii)->dof();
86                for (int iii = 0; iii < n_jdof; ++iii){
87                    DofToJoint(count_jdof)=ii;
88                    count_jdof+=1;
89                }
90            }
91                oVar_vec = Eigen::VectorXd::Zero(n_dof);
92                n_os=0;
93                n_jrc=0;
94                for (int i = 0; i < n_mdljnts; ++i){
95                        if(Path_r_c(i)){
96                                Path_r_c(i)=i;
97                                n_jrc++;
98                                for (int ii = 0; ii < n_dof; ++ii){
99                        if (DofToJoint(ii)==i){
100                               oVar_vec(n_os)=ii+1;
101                               n_os++;
102                       }
103                   }
104               }
105               }
106               oVar_vec.conservativeResize(n_os);
107               J_Pos_Ang_mat=Eigen::MatrixXd::Zero(6,n_os);
108               J_Ang_Pos_mat=Eigen::MatrixXd::Zero(n_os,6);
109               calc_J_Pos_Ang_mat(J_Pos_Ang_mat);
110               calc_J_Ang_Pos_mat(J_Ang_Pos_mat);
111               J_os_cs=J_Ang_Pos_mat;
112       }else{
113               n_os= read_cs_file(oVar_vec, CL_Folder + os_file);
114               J_os_csarrd=new double**[n_tv];
115               for (int i = 0; i < n_tv; i++){
116                       J_os_csarrd[i]=new double*[n_os];
117                       for(int j=0; j<n_os;j++) J_os_csarrd[i][j]=new double[n_cs];
118               }
119               read_JacFiles();
120               J_os_css = new Eigen::MatrixXd[n_tv];
121               for (int i = 0; i < n_tv; ++i){
122                       J_os_css[i]=Eigen::MatrixXd::Zero(n_cs,n_cs);
123                       J_os_css[i]=myMotControl.ConvertToEigenMatrix(n_cs,n_cs,J_os_csarrd[i]);
124               }
125       }
126
127       J_os_cs.resize(n_os,n_cs);
128       J_os_cs =Eigen::MatrixXd::Zero(n_os,n_cs);
129       P_CL.resize(n_cs,n_cs);
130       D_CL.resize(n_cs,n_cs);
131       I_CL.resize(n_cs,n_cs);
132       P_CL =Eigen::MatrixXd::Zero(n_cs,n_cs);
133       D_CL =Eigen::MatrixXd::Zero(n_cs,n_cs);
134       I_CL =Eigen::MatrixXd::Zero(n_cs,n_cs);
```

```
135         sensInFlagJC.resize(n_cs,1);
136         sensInFlagJC=Eigen::VectorXd::Zero(n_cs);
137         cVar_is.resize(n_cs,1);
138         cVar_des.resize(n_cs,1);
139         cVar_err.resize(n_cs,1);
140         cVar_d_err.resize(n_cs,1);
141         cVar_d_des.resize(n_cs,1);
142         cVar_pid.resize(n_cs,1);
143         cVar_i.resize(n_cs,1);
144         cVar_d.resize(n_cs,1);
145         cVar_old.resize(n_cs,1);
146         oVar_is_lcl.resize(n_os,1);
147         cVar_is=Eigen::VectorXd::Zero(n_cs);
148         cVar_des=Eigen::VectorXd::Zero(n_cs); //readFromFile!
149         cVar_err=Eigen::VectorXd::Zero(n_cs);
150         cVar_d_err=Eigen::VectorXd::Zero(n_cs);;
151         cVar_d_des=Eigen::VectorXd::Zero(n_cs);;
152         cVar_i=Eigen::VectorXd::Zero(n_cs);
153         cVar_d=Eigen::VectorXd::Zero(n_cs);
154         cVar_old = Eigen::VectorXd::Zero(n_cs);
155         oVar_cVar=Eigen::VectorXd(n_os);
156         oVar_is_lcl=Eigen::VectorXd(n_os);
157         Perm_os_Mat = Eigen::MatrixXd::Zero(n_os, n_dof);
158         Perm_os_Mat = calc_os_Permutation();
159
160         return;
161 };
```

Listing E.9: Function to update conceptional layer control variables

```
1  void ConceptionalLayer::Update_CtrlVars(){
2         if (ControlState!=ControlState_old)     {
3                 ControlState_old=ControlState;
4                 dbg_CL_runtime=true;
5                 if(ControlSpace==2 || ControlSpace==3){
6                         cVar_des=cVar_dess[ControlState];
7                         sensInFlagJC=Eigen::VectorXd::Zero(n_cs);
8                 }
9                 if(ControlSpace==3 || ControlSpace==4 || ControlSpace==5){
10                        P_CL=P_CLs[ControlState];
11                        I_CL=I_CLs[ControlState];
12                        D_CL=D_CLs[ControlState];
13                        if(!CALC_JACS) J_os_cs=J_os_css[ControlState];
14                }
15                if(ControlSpace==2 || ControlSpace==3){
16                        if(ControlSpace==2){//SenseIn
17                        for (int i = 0; i < n_cs; ++i)
18                                {
19                                        if(cVar_des(i)==888) sensInFlagJC(i)=1;
20                                if(sensInFlagJC(i)==1) {
21                                        cout << "Error(): SensIn detected, but not yet implemented!! setting desired
                                                value of [cDoF_ID= "<<i<<"] to 777, i.e. err=0" << endl;
22                                        cVar_des(i)=777;
23                                        }
24                                }
25                        }
26                }
27        }
28        dbg_CL_runtime=false;
29        return;
30 }
```

Listing E.10: Function to update current torques

```
1 void ConceptionalLayer::Update_Trq_is(){
2         cVar_is=-Ma_mat_lcl.transpose()*Fmtu_vec_lcl;
3         return;
4 };
```

Listing E.11: Function to update desired positions

```
1  void ConceptionalLayer::Update_Pos_des(){
2         getTriadM(rTrd_ID,  rM_Trd_mat);
3         getTriadParentBodyM(rTrd_ID, rM_Bdy_mat);
4         rM_mat=rM_Bdy_mat*rM_Trd_mat;
5         getTriadM(cTrd_des_ID,  cM_Trd_des_mat);
6         getTriadParentBodyM(cTrd_des_ID, cM_Bdy_des_mat);
7         cM_des_mat=(cM_Bdy_des_mat*cM_Trd_des_mat);
8         cVar_des.head(3)=Eigen::Vector3d(cM_des_mat(0,3), cM_des_mat(1,3),cM_des_mat(2,3));
9         cVar_des.tail(3)=Eigen::Vector3d(cM_des_mat(2,1), cM_des_mat(0,2),cM_des_mat(1,0));
10        return;
11 };
```

Listing E.12: Function to update current positions

```
void ConceptionalLayer::Update_Pos_is(){
        getTriadM(rTrd_ID,  rM_Trd_mat);
        getTriadParentBodyM(rTrd_ID, rM_Bdy_mat);
        rM_mat=rM_Bdy_mat*rM_Trd_mat;
        getTriadM(cTrd_ID,  cM_Trd_mat);
        getTriadParentBodyM(cTrd_ID, cM_Bdy_mat);
        cM_mat=(cM_Bdy_mat*cM_Trd_mat);
        cVar_is.head(3)=Eigen::Vector3d(cM_mat(0,3), cM_mat(1,3),cM_mat(2,3));
        cVar_is.tail(3)=Eigen::Vector3d(cM_mat(2,1), cM_mat(0,2),cM_mat(1,0));
        return;
};
```

Listing E.13: Function to update velocity of controlled coordinate

```
void ConceptionalLayer::Update_cVar_d(double dt){
        if(ControlSpace==3){
                bool calc_d =true;
                if (ControlSpace==4 && cTrd_des_ID!=cTrd_des_ID_old) calc_d =false;
            if(dt>=dtstep){
                if (calc_d)
                                        cVar_d=(cVar_err-cVar_old)/dt;
                    cVar_old=cVar_err;
                    if(ControlSpace==4) cTrd_des_ID_old=cTrd_des_ID;
                }
        }
        return;
};
```

Listing E.14: Function to update control error

```
void ConceptionalLayer::Update_cVar_err(){
        if(ControlSpace==3){
                cVar_err=cVar_is-cVar_des;
                for (int i = 0; i < n_cdof; ++i){
                        if (cVar_des(i)==777){
                                cVar_err(i)=0;
                        }
                }
        }
        if(ControlSpace==4){
                Eigen::Matrix4d cM_err_mat;
                cM_err_mat= cM_des_mat.inverse()*cM_mat;
                cVar_err.head(3)=Eigen::Vector3d(cM_err_mat(0,3), cM_err_mat(1,3), cM_err_mat(2,3));
                cVar_err.tail(3)=Eigen::Vector3d(0.5*(cM_err_mat(2, 1) - cM_err_mat(1, 2)),
                                                                                0.5*(cM_err_mat(0, 2) - cM_err_mat(2,
                                                                                    0)),
                                                                                0.5*(cM_err_mat(1, 0) - cM_err_mat(0,
                                                                                    1)));
                bool relrel=true;
                if(relrel){
                        cVar_err.head(3)=rM_mat.block(0,0,3,3).inverse()*cM_des_mat.block(0,0,3,3)*cVar_err.head(3);
                        cVar_err.tail(3)=rM_mat.block(0,0,3,3).inverse()*cM_des_mat.block(0,0,3,3)*cVar_err.tail(3);
                }
        }
        return;
};
```

Listing E.15: Function to update velocity of control error

```
void ConceptionalLayer::Update_cVar_d_err(){
        if(ControlSpace==2 || ControlSpace==3){
                cVar_d_err=cVar_d;
                if(ControlSpace==2) cVar_d_err-=cVar_d_des;
                for (int i = 0; i < n_cs; ++i){
                        if (cVar_des(i)==777){
                                cVar_d_err(i)=0;
                        }
                }
        }
        if(ControlSpace==4){//Position Control
                getTriadParentBodyW(cTrd_ID, cW_mat);
                getTriadParentBodyW(cTrd_des_ID, cW_des_mat);
                getTriadM(rTrd_ID,  rM_Trd_mat);
                getTriadParentBodyM(rTrd_ID, rM_Bdy_mat);
                rM_mat=rM_Bdy_mat*rM_Trd_mat;
                cM_Trd_d_mat=rM_mat.inverse()*(cW_mat-cW_des_mat)*rM_mat;
                cVar_d_err.head(3)=Eigen::Vector3d(cM_Trd_d_mat(0,3), cM_Trd_d_mat(1,3), cM_Trd_d_mat(2,3));
                cVar_d_err.tail(3)=Eigen::Vector3d(      0.5*(cM_Trd_d_mat(2, 1) - cM_Trd_d_mat(1, 2)),
                                                                                        0.5*(cM_Trd_d_mat(0, 2) -
                                                                                            cM_Trd_d_mat(2, 0)),
                                                                                        0.5*(cM_Trd_d_mat(1, 0) -
                                                                                            cM_Trd_d_mat(0, 1)));
```

```
22         }
23         return;
24 };
```

Listing E.16: Function to update inegration of control error

```
1 void ConceptionalLayer::Update_cVar_i(double dt){
2         if(dt>=dtstep) cVar_i+=cVar_err*dt;
3         if(ControlSpace==3) {
4                 for (int i = 0; i < n_cs; ++i)
5                         if (cVar_des(i)==777)
6                                 cVar_i(i)=0.0;
7         }
8         if(ControlSpace==4) {
9                 if (cTrd_des_ID==cTrd_ID)
10                                cVar_i=Eigen::VectorXd::Zero(6);
11        }
12        return;
13 };
```

Listing E.17: Function to update conceptional PID

```
1 void ConceptionalLayer::Update_cVar_pid(){
2         cVar_pid=P_CL*cVar_err+D_CL*cVar_d_err+I_CL*cVar_i;
3         return;
4 };
```

Listing E.18: Function to update conceptional layer output

```
1 void ConceptionalLayer::Update_oVar_cVar(){
2         oVar_cVar=oVar_is_lcl-(J_os_cs*cVar_pid)*180/PI;
3         oVar=oVar_cVar;
4         if(ControlSpace==3){
5                 for (int i = 0; i < n_cs; ++i){
6                         if(cVar_des(i)==777)oVar(i)=777;
7                 }
8         }
9         if(ControlSpace==4 || ControlSpace==5){
10                if(cTrd_des_ID==cTrd_ID){
11                        for (int i = 0; i < n_os; ++i){
12                                oVar(i)=777;
13                        }
14                }
15        }
16        return;
17 };
18
19 void ConceptionalLayer::Update_oVar(){
20        if(ControlSpace==2 )oVar=cVar_des;
21        if(ControlSpace==3 || ControlSpace==4 || ControlSpace==5) Update_oVar_cVar();
22        return;
23 }
```

Listing E.19: Function to update torque-angle Jacobian

```
1 void ConceptionalLayer::calc_J_Tht_Trq(){
2     Eigen::MatrixXd I_ncmtu_mat = Eigen::MatrixXd::Identity(n_cmtu,n_cmtu);
3     Eigen::MatrixXd J_Lbd_Tht=(I_ncmtu_mat+dLseedLce_mat_lcl).inverse()*Ma_mat_lcl;
4     J_os_cs=(Ma_mat_lcl.transpose()*dFcedLce_mat_lcl*J_Lbd_Tht).inverse();
5     return;
6 }
```

Listing E.20: Functions to update position-angle Jacobian

```
1 void ConceptionalLayer::calc_J_Pos_Ang_mat(Eigen::MatrixXd& jxqmat){
2         int current_DoF_ID=0;
3         jxqmat =  Eigen::MatrixXd::Zero(6,n_os);
4         for (int i = 0; i < n_os; ++i){
5                 current_DoF_ID=oVar_vec(i)-1;
6                 int ToID=model.getbodyidx(model.joint(DofToJoint(current_DoF_ID))->to()->parent());
7                 Eigen::Matrix4d M_to_jnt, M_Jnt_rl, M_X_rl;
8             Eigen::Vector3d Li_vec_to, Li_vec_wd, Li_vec_rl;
9                 Eigen::Vector3d PJnt_wd, PJnt_rl;
10                model.joint(DofToJoint(current_DoF_ID))->to()->Mmatrix(M_to_jnt);
11                PJnt_wd=M0o[ToID].block(0,0,3,3)*M_to_jnt.block(0,3,3,1);
12                M_Jnt_rl=rM_mat.inverse()*(M0o[ToID]*M_to_jnt);
13                PJnt_rl=M_Jnt_rl.block(0,3,3,1);
14                Eigen::Vector3d PX___wd=cM_mat.block(0,3,3,1);
15                Eigen::Vector3d PX____rl=cM_mat.block(0,3,3,1);
16                Eigen::Vector3d V_Jnt_X___wd, V_Jnt_X____rl;
```

```
17                  V_Jnt_X___wd=PX___wd*PJnt_wd;
18                  V_Jnt_X___rl=PX___rl*PJnt_rl;
19                  Li_vec_to(0)=Li[current_DoF_ID+1](2,1);
20              Li_vec_to(1)=Li[current_DoF_ID+1](0,2);
21              Li_vec_to(2)=Li[current_DoF_ID+1](1,0);
22              Li_vec_wd=M0o[ToID].block(0,0,3,3) * Li_vec_to;
23              Li_vec_rl=rM_mat.block(0,0,3,3).inverse()*Li_vec_wd;
24                  jxqmat.col(i).head(3)=Li_vec_wd.cross(V_Jnt_X___wd);
25                  jxqmat.col(i).tail(3)=Li_vec_wd.head(3);
26                  bool relrel=true;
27              if(relrel){
28                      jxqmat.col(i).head(3)=Li_vec_rl.cross(V_Jnt_X___rl);
29                          jxqmat.col(i).tail(3)=Li_vec_rl.head(3);
30                  }
31          }
32          J_Pos_Ang_mat=jxqmat;
33          return;
34  }
35
36  void ConceptionalLayer::calc_J_Ang_Pos_mat(Eigen::MatrixXd& jqxmat){
37          bool jqxpsinv=true;
38          if (jqxpsinv){
39                  jqxmat=(J_Pos_Ang_mat.transpose()*J_Pos_Ang_mat).inverse()*J_Pos_Ang_mat.transpose();
40          }
41          bool jxqttrans=false;
42          if (jxqttrans){
43                  jqxmat=J_Pos_Ang_mat.transpose();
44          }
45          J_Ang_Pos_mat=jqxmat;
46          J_os_cs=jqxmat;
47          return;
48  }
```

Listing E.21: Functions to calculate permutation matrices

```
 1  Eigen::MatrixXd ConceptionalLayer::calc_cDoFPermutation(Eigen::VectorXd &cdofvec){
 2          for (int i = 0; i < n_cdof; ++i){
 3                  for (int ii = 0; ii < n_dof; ++ii){
 4                          if(cdofvec(i)==ii+1) Perm_cDoF_Mat(i,ii)=1;
 5                  }
 6          }
 7          return Perm_cDoF_Mat;
 8  }
 9
10  Eigen::MatrixXd ConceptionalLayer::calc_cMtuPermutation(Eigen::VectorXd &cmtuvec){
11          Eigen::MatrixXd PermMat = Eigen::MatrixXd::Zero(n_cmtu, n_mtu);
12          for (int i = 0; i < n_cmtu; ++i)
13          {
14                  for (int ii = 0; ii < n_mtu; ++ii)
15                  {
16                          if(cmtuvec(i)==ii+1) PermMat(i,ii)=1;
17                  }
18          }
19          return PermMat;
20  }
21
22  Eigen::MatrixXd ConceptionalLayer::calc_os_Permutation(){
23          for (int i = 0; i < n_os; ++i){
24                  for (int ii = 0; ii < n_dof; ++ii){
25                          if(oVar_vec(i)==ii+1) Perm_os_Mat(i,ii)=1;
26                  }
27          }
28          return Perm_os_Mat;
29  }
```

E.5.4 Functions in `utrafolayer.cpp`

Listing E.22: Transformational layer initialisation function

```
1
2 void TransformationalLayer::Init_TL(){
3         dbg_TL_init=true;
4     dbg_TL_runtime=false;
5     ControlState_old=-1;
6     ControlState=0;
7         cout << "////////////////////////////////////////////////////" << endl;
8         cout << "////////INIT TRANSFORMATIONAL LAYER"<< endl;
9         cout << "////////////////////////////////////////////////////"<< endl;
10        n_dof=model.dof();
11        n_cdof= read_DofFile(cDoF_vec_TL, TL_Folder + cDoF_file);
12        ERR_CDOFVEC=false;
13        if (cDoF_vec_TL.size()==cDoF_vec_CL.size()){
14                Perm_cDoF_Mat_TL = Eigen::MatrixXd::Zero(n_cdof,n_dof);
15                Perm_cDoF_Mat_TL = calc_cDoFPermutation(cDoF_vec_TL);
16                Perm_cDoF_Mat_CL = Eigen::MatrixXd::Zero(n_cdof,n_dof);
17                Perm_cDoF_Mat_CL = calc_cDoFPermutation(cDoF_vec_CL);
18                for (int i = 0; i < n_cdof; ++i) if (cDoF_vec_CL(i)!=cDoF_vec_TL(i)) ERR_CDOFVEC=true;
19        }else ERR_CDOFVEC=true;
20
21        if(ERR_CDOFVEC){
22                cout << "ERROR(): MISMATCH IN CONCEPTIONAL AND TRANSFORMATIONAL JOINT VECTORS!!!" << endl;
23                cout << "        Perm_cDoF_Mat_TL.transpose()*cDoF_vec_TL=" << (Perm_cDoF_Mat_TL.transpose()*cDoF_vec_TL
                          ).transpose() << endl;
24                cout << "        Perm_cDoF_Mat_CL.transpose()*cDoF_vec_CL=" << (Perm_cDoF_Mat_CL.transpose()*cDoF_vec_CL
                          ).transpose() << endl;
25        }
26        else  cout << " Conceptional cDoFVecs are well defined!" << endl;
27
28        Tht_Coc_array=new double**[n_tv];
29        for (int i = 0; i < n_tv; i++){
30                Tht_Coc_array[i]=new double*[n_cdof];
31                for(int j=0; j<n_cdof;j++){
32                        Tht_Coc_array[i][j]=new double[1];
33                }
34        }
35        thtcocname = new std::string[n_tv];
36        read_Tht_Coc_Files();
37        Tht_Cocs = new Eigen::VectorXd[n_tv];
38        for (int i = 0; i < n_tv; ++i){
39                Tht_Cocs[i]=Eigen::VectorXd::Zero(n_cdof);
40                Tht_Cocs[i]=myMotControl.ConvertToEigenMatrix(n_cdof,1,Tht_Coc_array[i]);
41        }
42
43        P_Tht_arrd=new double**[n_tv];
44        D_Tht_arrd=new double**[n_tv];
45        I_Tht_arrd=new double**[n_tv];
46        P_Tht_Lbd_arrd=new double**[n_tv];
47        D_Tht_Lbd_arrd=new double**[n_tv];
48        I_Tht_Lbd_arrd=new double**[n_tv];
49        for (int i = 0; i < n_tv; i++){
50                P_Tht_arrd[i]=new double*[n_cdof];
51                D_Tht_arrd[i]=new double*[n_cdof];
52                I_Tht_arrd[i]=new double*[n_cdof];
53                P_Tht_Lbd_arrd[i]=new double*[n_cdof];
54                D_Tht_Lbd_arrd[i]=new double*[n_cdof];
55                I_Tht_Lbd_arrd[i]=new double*[n_cdof];
56                for(int j=0; j<n_cdof;j++){
57                        P_Tht_arrd[i][j]=new double[n_cdof];
58                        D_Tht_arrd[i][j]=new double[n_cdof];
59                        I_Tht_arrd[i][j]=new double[n_cdof];
60                        P_Tht_Lbd_arrd[i][j]=new double[n_cdof];
61                        D_Tht_Lbd_arrd[i][j]=new double[n_cdof];
62                        I_Tht_Lbd_arrd[i][j]=new double[n_cdof];
63                }
64        }
65        read_PID_Tht_Files();
66        read_PID_Tht_Lbd_Files();
67        P_Thts = new Eigen::MatrixXd[n_tv];
68        I_Thts = new Eigen::MatrixXd[n_tv];
69        D_Thts = new Eigen::MatrixXd[n_tv];
70        P_Tht_Lbds = new Eigen::MatrixXd[n_tv];
71        I_Tht_Lbds = new Eigen::MatrixXd[n_tv];
72        D_Tht_Lbds = new Eigen::MatrixXd[n_tv];
73        for (int i = 0; i < n_tv; ++i){
74                P_Thts[i]=Eigen::MatrixXd::Zero(n_cdof,n_cdof);
75                I_Thts[i]=Eigen::MatrixXd::Zero(n_cdof,n_cdof);
76                D_Thts[i]=Eigen::MatrixXd::Zero(n_cdof,n_cdof);
77                P_Tht_Lbds[i]=Eigen::MatrixXd::Zero(n_cdof,n_cdof);
78                I_Tht_Lbds[i]=Eigen::MatrixXd::Zero(n_cdof,n_cdof);
79                D_Tht_Lbds[i]=Eigen::MatrixXd::Zero(n_cdof,n_cdof);
80                P_Thts[i]=myMotControl.ConvertToEigenMatrix(n_cdof,n_cdof,P_Tht_arrd[i]);
81                I_Thts[i]=myMotControl.ConvertToEigenMatrix(n_cdof,n_cdof,I_Tht_arrd[i]);
82                D_Thts[i]=myMotControl.ConvertToEigenMatrix(n_cdof,n_cdof,D_Tht_arrd[i]);
```

```
83                  P_Tht_Lbds[i]=myMotControl.ConvertToEigenMatrix(n_cdof,n_cdof,P_Tht_Lbd_arrd[i]);
84                  I_Tht_Lbds[i]=myMotControl.ConvertToEigenMatrix(n_cdof,n_cdof,I_Tht_Lbd_arrd[i]);
85                  D_Tht_Lbds[i]=myMotControl.ConvertToEigenMatrix(n_cdof,n_cdof,D_Tht_Lbd_arrd[i]);
86          }
87          if(CALC_JACS){
88                  Eigen::MatrixXd Ma_mat_tmp;
89                  Ma_mat_tmp=Ma_mat_glb * Perm_cDoF_Mat_TL.transpose();
90              mVec_TL_global = Eigen::VectorXd::Zero(n_mtu);
91              mVec_TL = Eigen::VectorXd::Zero(n_mtu);
92              n_cmtu = 0;
93              for (int i = 0; i < n_mtu; ++i){
94                  if((Ma_mat_tmp.row(i)).norm()!=0){
95                   mVec_TL_global(i)=i+1;
96                   mVec_TL(n_cmtu)=i+1;
97                   n_cmtu++;
98                  }
99              }
100             mVec_TL.conservativeResize(n_cmtu);
101             Perm_cMtu_Mat_TL = Eigen::MatrixXd::Zero(n_cmtu,n_mtu);
102                 Perm_cMtu_Mat_TL = calc_cMtuPermutation(mVec_TL);
103                 Ma_mat_lcl=Perm_cMtu_Mat_TL*Ma_mat_tmp;
104             dLseedLce_mat_lcl=Perm_cMtu_Mat_TL*dLseedLce_mat_glb*Perm_cMtu_Mat_TL.transpose();
105             Lce_vec_lcl=Perm_cMtu_Mat_TL * Lce_vec_glb;
106             U_vec_lcl=Perm_cMtu_Mat_TL * U_vec_glb;
107             calc_J_Lbd_Tht();
108                 J_U_Lbd = Eigen::MatrixXd::Zero(n_cmtu, n_cmtu);
109             calc_J_U_Tht();
110         }else{
111                 n_cmtu = read_MuscleFile(mVec_TL, TL_Folder + mVec_file);
112                 J_Lbd_Tht_arrd=new double**[n_tv];
113                 J_U_Tht_arrd=new double**[n_tv];
114                 for (int i = 0; i < n_tv; i++){
115                         J_Lbd_Tht_arrd[i]=new double*[n_cmtu];
116                         J_U_Tht_arrd[i]=new double*[n_cmtu];
117                         for(int j=0; j<n_cmtu;j++){
118                          J_Lbd_Tht_arrd[i][j]=new double[n_cdof];
119                          J_U_Tht_arrd[i][j]=new double[n_cdof];
120                         }
121                 }
122                 read_JacFiles();
123                 J_Lbd_Thts = new Eigen::MatrixXd[n_tv];
124                 J_U_Thts = new Eigen::MatrixXd[n_tv];
125                 for (int i = 0; i < n_tv; ++i){
126                         J_Lbd_Thts[i]=Eigen::MatrixXd::Zero(n_cmtu,n_cdof);
127                         J_U_Thts[i]=Eigen::MatrixXd::Zero(n_cmtu,n_cdof);
128                         J_Lbd_Thts[i]=myMotControl.ConvertToEigenMatrix(n_cmtu,n_cdof,J_Lbd_Tht_arrd[i]);
129                         J_U_Thts[i]=myMotControl.ConvertToEigenMatrix(n_cmtu,n_cdof,J_U_Tht_arrd[i]);
130                 }
131         }
132         Perm_cMtu_Mat_TL = Eigen::MatrixXd::Zero(n_cmtu,n_mtu);
133         Perm_cMtu_Mat_TL = calc_cMtuPermutation(mVec_TL);
134         Tht_is = Eigen::VectorXd::Zero(n_cdof);
135         Tht_des = Eigen::VectorXd::Zero(n_cdof);
136         Tht_err = Eigen::VectorXd::Zero(n_cdof);
137         Tht_d = Eigen::VectorXd::Zero(n_cdof);
138         Tht_i = Eigen::VectorXd::Zero(n_cdof);
139         Tht_d_des = Eigen::VectorXd::Zero(n_cdof);
140         Tht_d_err = Eigen::VectorXd::Zero(n_cdof);
141         Tht_pid = Eigen::VectorXd::Zero(n_cdof);
142         Tht_Lbd_pid = Eigen::VectorXd::Zero(n_cdof);
143         Tht_Coc= Eigen::VectorXd::Zero(n_cdof);
144         Lbd_Tht = Eigen::VectorXd::Zero(n_cmtu);
145         U_Tht = Eigen::VectorXd::Zero(n_cmtu);
146         U_Tht_Coc = Eigen::VectorXd::Zero(n_cmtu);
147 return;
148 };
```

Listing E.23: Function to update transformational layer control variables

```
1 void TransformationalLayer::Update_CtrlVars(){
2         if (ControlState!=ControlState_old)     {
3                 ControlState_old=ControlState;
4                 P_Tht=P_Thts[ControlState];
5                 I_Tht=I_Thts[ControlState];
6                 D_Tht=D_Thts[ControlState];
7                 P_Tht_Lbd=P_Tht_Lbds[ControlState];
8                 I_Tht_Lbd=I_Tht_Lbds[ControlState];
9                 D_Tht_Lbd=D_Tht_Lbds[ControlState];
10                if(!CALC_JACS){//Read Jacs
11                        J_Lbd_Tht=J_Lbd_Thts[ControlState];
12                        J_U_Tht=J_U_Thts[ControlState];
13                }
14                Tht_Coc=Tht_Cocs[ControlState];
15                Tht_d_des = Eigen::VectorXd::Zero(n_cdof);      //
16        }
17        return;
18 }
```

Listing E.24: Function to update current angles

```
void TransformationalLayer::Update_Tht_is(){
        for (int i = 0; i < n_cdof; ++i){
                int current_DoF_ID=cDoF_vec_TL(i)-1;
                Tht_is(i)=q[current_DoF_ID]*180/PI;
        }
        return;
}
```

Listing E.25: Function to update velocity of controlled angles

```
void TransformationalLayer::Update_Tht_d(){
        for (int i = 0; i < n_cdof; ++i){
                int current_DoF_ID=cDoF_vec_TL(i)-1;
                Tht_d(i)=qd[current_DoF_ID]*180/PI;
        }
        return;
}
```

Listing E.26: Function to update angle control error

```
void TransformationalLayer::Update_Tht_err(){
        Tht_err=Tht_is-Tht_des;
        Tht_d_err=Tht_d-Tht_d_des;
        for (int i = 0; i < n_cdof; ++i){
                if (Tht_des(i)==777 || (Tht_des(i)==Tht_is(i))){
                        Tht_des(i)=Tht_is(i);
                        Tht_err(i)=0;
                        Tht_d_err(i)=0;
                }
        }
        return;
}
```

Listing E.27: Function to update inegration of angle control error

```
void TransformationalLayer::Update_Tht_i(double dt){
        if(dt>=dtstep) {
                Tht_i+=Tht_err*dt;
                for (int i = 0; i < n_cdof; ++i)
                        if (Tht_des(i)==777 || (Tht_des(i)==Tht_is(i)))
                                Tht_i(i)=0.0;
        }
        return;
}
```

Listing E.28: Function to update direct stimulation PID

```
void TransformationalLayer::Update_Tht_pid(){
        Tht_pid=P_Tht*Tht_err+D_Tht*Tht_d_err+I_Tht*Tht_i;
        return;
}
```

Listing E.29: Function to update hierarchical angle-length PID

```
void TransformationalLayer::Update_Tht_Lbd_pid(){
        Tht_Lbd_pid=P_Tht_Lbd*Tht_err+D_Tht_Lbd*Tht_d_err+I_Tht_Lbd*Tht_i;
        return;
}
```

Listing E.30: Function to update transformational layer output of desired MTU lengths

```
Eigen::VectorXd TransformationalLayer::Update_Lbd_Tht(){
        Lbd_Tht=Lce_vec_Icl - J_Lbd_Tht*Tht_Lbd_pid;
        return Perm_cMtu_Mat_TL.transpose()*Lbd_Tht;
}
```

Listing E.31: Function to update transformational layer output of direct stimulation

```
Eigen::VectorXd TransformationalLayer::Update_U_Tht(){
        U_Tht=-J_U_Tht*Tht_pid;
        Eigen::VectorXd utht;
        utht =Perm_cMtu_Mat_TL.transpose()*U_Tht;
        return utht;
}
```

Listing E.32: Function to update joint co-contraction

```
Eigen::VectorXd TransformationalLayer::Update_U_Tht_Coc(){
        bool THT_COC_ERR=false;
        THT_COC=false;
        for (int i = 0; i < n_cdof; ++i) if(Tht_Coc(i)!=0.0) THT_COC=true;
        if(THT_COC) {
                J_Tht_U=(J_U_Tht.transpose()*J_U_Tht).inverse()*J_U_Tht.transpose();
                Eigen::FullPivLU<Eigen::MatrixXd> lu(J_Tht_U);
                Eigen::MatrixXd A_null_space = lu.kernel();
                if(A_null_space.rows()!=n_cmtu || A_null_space.cols()!=n_cdof) THT_COC_ERR=true;
                if (!THT_COC_ERR){
                        Eigen::MatrixXd permNS =Eigen::MatrixXd::Zero(n_cdof, n_cdof);
                        static int ns_col=0;
                        static int ns_row=0;
                        static double rowval=0.0;
                        for (int i = 0; i < n_cdof; ++i){
                                ns_col=0;
                                ns_row=0;
                                for (int j = 0; j < n_cmtu; ++j){
                                        if (A_null_space(j,i)==1){
                                                ns_row=j;
                                                break;
                                        }
                                }
                                rowval=0.0;
                                for (int k = 0; k < n_cdof; ++k){
                                        if (abs(J_U_Tht(ns_row,k))>rowval){
                                                rowval=abs(J_U_Tht(ns_row,k));
                                                ns_col=k;
                                        }
                                }
                                permNS(i,ns_col)=1;
                        }
                        U_Tht_Coc=A_null_space*permNS*Tht_Coc; // #TODO: set eq NRs from Paper!
                }
        }
        if(THT_COC_ERR){
                U_Tht_Coc=Eigen::VectorXd::Zero(n_cmtu);
        }
        Eigen::VectorXd uthtcoc;
        uthtcoc =Perm_cMtu_Mat_TL.transpose()*U_Tht_Coc;
        return uthtcoc;
}
```

Listing E.33: Function to update angle-length Jacobian

```
void TransformationalLayer::calc_J_Lbd_Tht(){
        Eigen::MatrixXd I_ncmtu_mat = Eigen::MatrixXd::Identity(n_cmtu,n_cmtu);
        J_Lbd_Tht=(I_ncmtu_mat+dLseedLce_mat_lcl).inverse()*Ma_mat_lcl;
        return;
}
```

Listing E.34: Functions to update angle-stimulation Jacobian

```
void TransformationalLayer::calc_J_U_Tht(){
        for (int i = 0; i < n_cmtu; ++i) J_U_Lbd(i,i)=-U_vec_lcl(i)/Lce_vec_lcl(i);
        J_U_Tht=J_U_Lbd*J_Lbd_Tht;
        J_Tht_U=(J_U_Tht.transpose()*J_U_Tht).inverse()*J_U_Tht.transpose();
        return;
}
```

Listing E.35: Functions to calculate permutation matrices

```
Eigen::MatrixXd TransformationalLayer::calc_cDoFPermutation(Eigen::VectorXd &cdofvec){
        int ncdof=cdofvec.size();
        Eigen::MatrixXd PermMat = Eigen::MatrixXd::Zero(ncdof, n_dof);
        for (int i = 0; i < ncdof; ++i){
                for (int ii = 0; ii < n_dof; ++ii){
                        if(cdofvec(i)==ii+1) PermMat(i,ii)=1;
                }
        }
        return PermMat;
};

Eigen::MatrixXd TransformationalLayer::calc_cMtuPermutation(Eigen::VectorXd &cmtuvec){
        Eigen::MatrixXd PermMat = Eigen::MatrixXd::Zero(n_cmtu, n_mtu);
        for (int i = 0; i < n_cmtu; ++i){
                for (int ii = 0; ii < n_mtu; ++ii){
                        if(cmtuvec(i)==ii+1) PermMat(i,ii)=1;
                }
        }
        return PermMat;
};
```

E.5.5 Functions in `ustructlayer.cpp`

Listing E.36: Structural layer initialisation function

```
1 void StructuralLayer::Init_SL(){
2         dbg_SL_init=true;
3     dbg_SL_runtime=false;
4     ControlState_old=-1;
5     ControlState=0;
6         cout << "////////////////////////////////////////////////////" << endl;
7         cout << "////////INIT STRUCTURAL LAYER"<< endl;
8         cout << "////////////////////////////////////////////////////"<< endl;
9
10        n_dof=model.dof();
11        n_cmtu = read_MuscleFile(mVec_SL, SL_Folder + mVec_file);
12
13        Perm_cMtu_Mat_SL=Eigen::MatrixXd::Zero(n_cmtu,n_mtu);
14        Perm_cMtu_Mat_SL=calc_cMtuPermutation(mVec_SL);
15        if (ControlSpace!=1){
16                if(dbg_SL_init) cout << "        mVec_TL=" << mVec_TL.transpose() << endl;
17                ERR_MVEC=false;
18                if (mVec_SL.size()==mVec_TL.size()){
19                        Perm_cMtu_Mat_TL=Eigen::MatrixXd::Zero(n_cmtu,n_mtu);
20                        Perm_cMtu_Mat_TL=calc_cMtuPermutation(mVec_TL);
21                        for (int i = 0; i < n_cmtu; ++i) if (mVec_SL(i)!=mVec_TL(i)) ERR_MVEC=true;
22                }else ERR_MVEC=true;
23        if(ERR_MVEC){
24                cout << "ERROR(): MISMATCH IN STRUCTURAL AND TRANSFORMATIONAL MUSCLE VECTORS!!!" << endl;
25        }
26        else if(dbg_SL_init) cout << "  structural muscle Vector is well defined!" << endl;
27        }
28        if (ControlSpace==1){
29                lbdname = new std::string[n_tv];
30                lambda_arrd = new double*[n_tv];
31                for (int i = 0; i < n_tv; ++i) lambda_arrd[i] = new double[n_cmtu];
32                read_Lambda();
33                Lambda=Eigen::VectorXd::Zero(n_cmtu);
34                Lambdas = new Eigen::VectorXd[n_tv];
35                for (int i = 0; i < n_tv; ++i){
36                        Lambdas[i]=Eigen::VectorXd::Zero(n_cmtu);
37                        Lambdas[i]=myMotControl.ConvertToEigenVector(n_cmtu,lambda_arrd[i]);
38                }
39        }
40        ucocname = new std::string[n_tv];
41        kpaname = new std::string[n_tv];
42        ucoc_arrd = new double*[n_tv];
43        kappa_arrd = new double*[n_tv];
44        for (int i = 0; i < n_tv; i++){
45                ucoc_arrd[i] = new double[n_cmtu];
46                kappa_arrd[i] = new double[n_cmtu];
47        }
48        read_UcocRef();
49        read_Kappa();
50        U_Coc=Eigen::VectorXd::Zero(n_cmtu);
51        Kappa=Eigen::VectorXd::Zero(n_cmtu);
52        U_Cocs = new Eigen::VectorXd[n_tv];
53        Kappas = new Eigen::VectorXd[n_tv];
54        for (int i = 0; i < n_tv; ++i){
55                U_Cocs[i]=Eigen::VectorXd::Zero(n_cmtu);
56                Kappas[i]=Eigen::VectorXd::Zero(n_cmtu);
57                U_Cocs[i]=myMotControl.ConvertToEigenVector(n_cmtu,ucoc_arrd[i]);
58                Kappas[i]=myMotControl.ConvertToEigenVector(n_cmtu,kappa_arrd[i]);
59        }
60        if(ControlSpace==1) Lambda=Lambdas[0];
61        U_Coc=U_Cocs[0];
62        Kappa=Kappas[0];
63        L_ce_opt=Eigen::VectorXd::Zero(n_cmtu);
64        L_ce_opt=Perm_cMtu_Mat_SL*L_ce_opt_global;
65        L_ce=Eigen::VectorXd::Zero(n_cmtu);
66        U_Lbd=Eigen::VectorXd::Zero(n_cmtu);
67        U_Tht=Eigen::VectorXd::Zero(n_cmtu);
68        U_Tht_Coc=Eigen::VectorXd::Zero(n_cmtu);
69        return;
70 };
```

Listing E.37: Function to update structural layer layer control variables

```
void StructuralLayer::Update_CtrlVars(){
        if (ControlState!=ControlState_old)     {
                ControlState_old=ControlState;
                U_Coc=U_Cocs[ControlState];
                Kappa=Kappas[ControlState];
                Sigma=sigma0*Eigen::VectorXd::Ones(n_mtu);
                if(ControlSpace==1) Lambda=Lambdas[ControlState];
        }
        return;
}
```

Listing E.38: Function to update structural layer output of closed-loop stimulation

```
Eigen::VectorXd StructuralLayer::Update_U_Lbd(){
        U_Lbd=Kappa.array()*(L_ce-Lambda+sigma0*V_ce).array()/L_ce_opt.array();
        return Perm_cMtu_Mat_SL.transpose()*U_Lbd;
}
```

Listing E.39: Functions to calculate permutation matrices

```
Eigen::MatrixXd StructuralLayer::calc_cMtuPermutation(Eigen::VectorXd &cmtuvec){
        Eigen::MatrixXd PermMat = Eigen::MatrixXd::Zero(n_cmtu, n_mtu);
        for (int i = 0; i < n_cmtu; ++i){
                for (int ii = 0; ii < n_mtu; ++ii){
                        if(cmtuvec(i)==ii+1) PermMat(i,ii)=1;
                }
        }
        return PermMat;
};
```

Bibliography

E Acome, SK Mitchell, TG Morrissey, MB Emmett, C Benjamin, M King, M Radakovitz, and C Keplinger. Hydraulically amplified self-healing electrostatic actuators with muscle-like performance. *Science*, 359(6371):61–65, 2018.

Jack A Adams. A closed-loop theory of motor learning. *Journal of motor behavior*, 3(2): 111–150, 1971.

A. V. Alexandrov, A. A. Frolov, and J. Massion. Biomechanical analysis of movement strategies in human forward trunk bending. I. Modeling. *Biologcal Cybernetics*, 84(6): 425–434, 2001.

Frank Allgöwer and Alex Zheng. *Nonlinear model predictive control*, volume 26. Birkhäuser, 2012.

Michael A. Arbib and Shun-Ichi Amari. Sensori-motor transformations in the brain (with a critique of the tensor theory of cerebellum). *Journal of Theoretical Biology*, 112(1): 123–155, 1985.

Frances A Batchelor, Susan B Williams, Tissa Wijeratne, Catherine M Said, and Sandra Petty. Balance and gait impairment in transient ischemic attack and minor stroke. *Journal of stroke and cerebrovascular diseases*, 24(10):2291–2297, 2015.

A. Bayer, S. Schmitt, M. Günther, and D.F.B. F.B. Haeufle. The influence of biophysical muscle properties on simulating fast human arm movements. *Computer Methods in Biomechanics and Biomedical Engineering*, 20(8):803–821, 2017.

Julian Berberich, Johannes Köhler, Matthias A Muller, and Frank Allgower. Data-driven model predictive control with stability and robustness guarantees. *IEEE Transactions on Automatic Control*, 2020.

Nikolai A. Bernstein. *The co-ordination and regulation of movements*. Pergamon Press, 1967.

E. Bizzi, N. Hogan, F.A. Mussa-Ivaldo, and S. Giszter. Does the nervous system use equilibrium-point control to guide single and multiple joint movements? *The Behavioral and brain sciences*, 15(4):603–613, 1992.

Sarah-J. Blakemore, Chris D. Frith, and Daniel M. Wolpert. Spatio-temporal prediction modulates the perception of self-produced stimuli. *Journal of Cognitive Neuroscience*, 11 (5):551–559, 1999.

Ronen Blecher, Lia Heinemann-Yerushalmi, Eran Assaraf, Nitzan Konstantin, Jens R Chapman, Timothy C Cope, Guy S Bewick, Robert W Banks, and Elazar Zelzer. New functions for the proprioceptive system in skeletal biology. *Philosophical Transactions of the Royal Society B: Biological Sciences*, 373(1759):20170327, 2018.

Stefanie Brändle, Syn Schmitt, and Matthias A. Müller. A systems-theoretic analysis of low-level human motor control: application to a single-joint arm model. *Journal of Mathematical Biology*, 80(4):1139–1158, 2020.

David J Chalmers. What is a neural correlate of consciousness. *Neural correlates of consciousness: Empirical and conceptual questions*, pages 17–39, 2000.

Michel Chasles. Note sur les propriétés générales du système de deux corps semblables entr'eux et placés d'une manière quelconque dans l'espace; et sur le déplacement fini ou infiniment petit d'un corps solide libre. *Bulletin des Sciences Mathématiques, Férussac*, 14:321–26, 1830.

Shelagh B Coutts. Diagnosis and management of transient ischemic attack. *CONTINUUM: Lifelong Learning in Neurology*, 23(1):82, 2017.

Francis Crick and Christof Koch. Towards a neurobiological theory of consciousness. In *Seminars in the Neurosciences*, volume 2, page 203, 1990.

Carlo J. de Luca and Paola Contessa. Hierarchical control of motor units in voluntary contractions. *Journal of Neurophysiology*, 107(1):178–195, 2012.

S.J. De Serres and T.E. Milner. Wrist muscle activation patterns and stiffness associated with stable and unstable mechanical loads. *Experimental Brain Research*, 86(2):451–458, 1991.

Michel Desmurget, Karen T Reilly, Nathalie Richard, Alexandru Szathmari, Carmine Mottolese, and Angela Sirigu. Movement intention after parietal cortex stimulation in humans. *science*, 324(5928):811–813, 2009.

Travis DeWolf and Chris Eliasmith. The neural optimal control hierarchy for motor control. *Journal of Neural Engineering*, 8(6):065009, 2011.

Edsger W Dijkstra et al. A note on two problems in connexion with graphs. *Numerische mathematik*, 1(1):269–271, 1959.

Kenji Doya. Complementary roles of basal ganglia and cerebellum in learning and motor control. *Current Opinion in Neurobiology*, 10(6):732–739, 2000.

John C Eccles. *The cerebellum as a neuronal machine*. Springer Science & Business Media, 1967.

W. T. Edwards. Effect of joint stiffness on standing stability. *Gait & Posture*, 25(3):432–439, 2007.

Wilfried Elmenreich. An introduction to sensor fusion. *Vienna University of Technology, Austria*, 502:1–28, 2002.

AG Feldman. Control of the length of a muscle. *Biophysics*, 19(2):766–771, 1974.

R.J. Full and D.E. Koditschek. Templates and anchors: neuromechanical hypotheses of legged locomotion on land. *The Journal of Experimental Biology*, 202(Pt 23):3325–3332, 1999.

Gowrishankar Ganesh, Keigo Nakamura, Supat Saetia, Alejandra Mejia Tobar, Eiichi Yoshida, Hideyuki Ando, Natsue Yoshimura, and Yasuharu Koike. Utilizing sensory prediction errors for movement intention decoding: A new methodology. *Science Advances*, 4(5):eaaq0183, 2018.

Jia-Hong Gao, Lawrence M. Parsons, James M. Bower, Jinhu Xiong, Jinqi Li, and Peter T. Fox. Cerebellum implicated in sensory acquisition and discrimination rather than motor control. *Science*, 272(5261):545–547, 1996.

Peter Gawthrop, Ian Loram, Martin Lakie, and Henrik Gollee. Intermittent control: a computational theory of human control. *Biological cybernetics*, 104(1):31–51, 2011.

Pl Gawthrop. Self-tuning pid controllers: Algorithms and implementation. *IEEE Transactions on Automatic Control*, 31(3):201–209, 1986.

Rodolphe Gentili, Cheol E. Han, Nicolas Schweighofer, and Charalambos Papaxanthis. Motor learning without doing: Trial-by-trial improvement in motor performance during mental training. *Journal of Neurophysiology*, 104(2):774–783, 2010.

Keyan Ghazi-Zahedi. *Morphological Intelligence*. Springer International Publishing, 2019. ISBN 978-3-030-20621-5.

P.L. Gribble, D.J. Ostry, V. Sanguineti, and R. Laboissière. Are complex control signals required for human arm movement? *Journal of Neurophysiology*, 79(3):1409–1424, 1998.

P.L. Gribble, L.I. Mullin, N. Cothros, and A. Mattar. Role of cocontraction in arm movement accuracy. *Journal of Neurophysiology*, 89(5):2396–2405, 2003.

Gaël Guennebaud, Benoît Jacob, et al. Eigen v3. http://eigen.tuxfamily.org, 2010.

M Günther and H Ruder. Synthesis of two-dimensional human walking: a test of the λ-model. *Biological Cybernetics*, 89(2):89–106, 2003.

M Günther, O Müller, and R Blickhan. Watching quiet human stance to shake off its straitjacket. *Archive of Applied Mechanics*, 81(3):283–302, 2011.

M Günther, O Müller, and R Blickhan. What does head movement tell about the minimum number of mechanical degrees of freedom in quiet human stance? *Archive of Applied Mechanics*, 82(3):333–344, 2012.

M Günther, D F B Haeufle, and S Schmitt. The basic mechanical structure of the skeletal muscle machinery: One model for linking microscopic and macroscopic scales. *Journal of Theoretical Biology*, 456:137–167, 2018. [with Corrigendum].

Michael Günther. *Computersimulationen zur Synthetisierung des muskulär erzeugten menschlichen Gehens unter Verwendung eines biomechanischen Mehrkörpermodells*. PhD thesis, Eberhard-Karls-Universität, Tübingen, Germany, 1997.

Michael Günther and Heiko Wagner. Dynamics of quiet human stance: computer simulations of a triple inverted pendulum model. *Computer Methods in Biomechanics and Biomedical Engineering*, 19(8):819–834, 2016.

Michael Günther, Syn Schmitt, and Veit Wank. High-frequency oscillations as a consequence of neglected serial damping in hill-type muscle models. *Biological cybernetics*, 97(1):63–79, 2007.

Christophe Habas, Alain Bertholz, Tamar Flash, and Daniel Bennequin. Does the cerebellum implement or select geometries? A speculative note. *The Cerebellum*, 19(2):1–7, 2020.

D. F B Haeufle, M. Günther, A. Bayer, and S. Schmitt. Hill-type muscle model with serial damping and eccentric force-velocity relation. *Journal of Biomechanics*, 47(6):1531–1536, 2014a.

D. F. B. Haeufle, M. Günther, G. Wunner, and S. Schmitt. Quantifying control effort of biological and technical movements: An information-entropy-based approach. *Physical Review E*, 89(1):012716, 2014b.

DFB Haeufle, S Grimmer, K-T Kalveram, and A Seyfarth. Integration of intrinsic muscle properties, feed-forward and feedback signals for generating and stabilizing hopping. *Journal of The Royal Society Interface*, 9(72):1458–1469, 2012.

Maria Hammer, Michael Günther, D.F.B. Haeufle, and Syn Schmitt. Tailoring anatomical muscle paths: a sheath-like solution for muscle routing in musculo-skeletal computer models. *Mathematical Biosciences*, 311:68–81, 2019.

Richard S Hartenberg and Jacques Denavit. A kinematic notation for lower pair mechanisms based on matrices. *Journal of Applied Mechanics*, 77(2):215–221, 1955.

H Hatze. A myocybernetic control model of skeletal muscle. *Biological Cybernetics*, 25(2): 103–119, 1977.

Karl-Michael Haus. *Sensomotorik*, pages 87–119. Springer Berlin Heidelberg, Berlin, Heidelberg, 2014. ISBN 978-3-642-41929-4.

A. Henze. *Dreidimensionale biomechanische Modellierung und die Entwicklung eines Reglers zur Simulation zweibeinigen Gehens*. PhD thesis, Eberhard-Karls-Universität, Tübingen, Germany, 2002.

David J. Herzfeld and Reza Shadmehr. Cerebellum estimates the sensory state of the body. *Trends in Cognitive Sciences*, 18(2):66–67, 2014.

Archibald Vivian Hill. The heat of shortening and the dynamic constants of muscle. *Proceedings of the Royal Society of London. Series B-Biological Sciences*, 126(843):136–195, 1938.

Archibald Vivian Hill. The abrupt transition from rest to activity in muscle. *Proceedings of the Royal Society of London. Series B-Biological Sciences*, 136(884):399–420, 1949.

P.J. Holmes, R.J. Full, D. Koditschek, and J. Guckenheimer. The dynamics of legged locomotion: models, analyses, and challenges. *SIAM Review*, 48(2):207–304, 2006.

Wei-Li Hsu and John P. Scholz. Motor abundance supports multitasking while standing. *Human Movement Science*, 31(4):844–862, 2012.

M Hulliger, N Dürmüller, A Prochazka, and P Trend. Flexible fusimotor control of muscle spindle feedback during a variety of natural movements. *Progress in Brain Research*, 80: 87–101, 1989.

Hugh Esmor Huxley. The mechanism of muscular contraction. *Science*, 164(3886):1356–1366, 1969.

Tatsuay Ibuki, Johannes R Walter, Takeshi Hatanaka, and Masayuki Fujita. Frame rate-based discrete visual feedback pose regulation: A passivity approach. *IFAC Proceedings Volumes*, 47(3):11171–11176, 2014.

GC Joyce, PMH Rack, and DR Westbury. The mechanical properties of cat soleus muscle during controlled lengthening and shortening movements. *The Journal of physiology*, 204 (2):461–474, 1969.

N Karajan, O Röhrle, W Ehlers, and S Schmitt. Linking continuous and discrete intervertebral disc models through homogenisation. *Biomechanics and modeling in mechanobiology*, 12(3):453–466, 2013.

Amir Karniel. Open questions in computational motor control. *Journal of integrative neuroscience*, 10(03):385–411, 2011.

Bernhard Katz. The relation between force and speed in muscular contraction. *The Journal of Physiology*, 96(1):45–64, 1939.

Risa Kawai, Timothy Markman, Rajesh Poddar, Raymond Ko, Antoniu L Fantana, Ashesh K Dhawale, Adam R Kampff, and Bence P Ölveczky. Motor cortex is required for learning but not for executing a motor skill. *Neuron*, 86(3):800–812, 2015.

Ole Kiehn. Decoding the organization of spinal circuits that control locomotion. *Nature Reviews Neuroscience*, 17(4):224–238, 2016.

D. A. Kistemaker, Arthur J. van Soest, and Maarten F. Bobbert. Is equilibrium point control feasible for fast goal-directed single-joint movements? *Journal of Neurophysiology*, 95(5): 2898–2912, 2006.

D. A. Kistemaker, Arthur J. van Soest, and Maarten F. Bobbert. A model of open-loop control of equilibrium position and stiffness of the human elbow joint. *Biological Cybernetics*, 96(3):341–350, 2007.

S. Kitazaki and M.J. Griffin. A modal analysis of whole-body vertical vibration, using a finite element model of the human body. *Journal of Sound and Vibration*, 200(1):83 – 103, 1997.

Johannes Koehler, Matthias A Muller, and Frank Allgower. Constrained nonlinear output regulation using model predictive control. *IEEE Transactions on Automatic Control*, 2021.

Tamara G. Kolda and Brett W. Bader. Tensor decompositions and applications. *SIAM Review*, 51(3):455–500, 2009.

V Kumar. Meam 520 notes: The theorems of euler and chasles. *University of Pennsylvania*, 2014.

Mark L. Latash. The bliss (not the problem) of motor abundance (not redundancy). *Experimental Brain Research*, 217(1):1–5, 2012.

Mark L Latash, John P Scholz, and Gregor Schöner. Motor control strategies revealed in the structure of motor variability. *Exercise and Sport Sciences Reviews*, 30(1):26–31, 2002.

Giovanni Legnani, Federico Casalo, Paolo Righettini, and Bruno Zappa. A homogeneous matrix approach to 3d kinematics and dynamics – II. Applications to chains of rigid bodies and serial manipulators. *Mechanism and Machine Theory*, 31(5):589–605, 1996a.

Giovanni Legnani, Federico Casolo, Paolo Righettini, and Bruno Zappa. A homogeneous matrix approach to 3d kinematics and dynamics – I. Theory. *Mechanism and Machine Theory*, 31(5):573–587, 1996b.

Ian D Loram, Henrik Gollee, Martin Lakie, and Peter J Gawthrop. Human control of an inverted pendulum: is continuous control necessary? is intermittent control effective? is intermittent control physiological? *The Journal of physiology*, 589(2):307–324, 2011.

David Luenberger. An introduction to observers. *IEEE Transactions on automatic control*, 16(6):596–602, 1971.

Kevin M Lynch and Frank C Park. *Modern Robotics*. Cambridge University Press, 2017.

John H. Martin. The corticospinal system: From development to motor control. *The Neuroscientist*, 11(2):161–173, 2005.

Anthony N Martonosi. Animal electricity, ca2+ and muscle contraction. a brief history of muscle research. *Acta Biochimica Polonica*, 47(3):493–516, 2000.

PBC Matthews. A study of certain factors influencing the stretch reflex of the decerebrate cat. *The Journal of Physiology*, 147(3):547, 1959.

Joseph McIntyre and Emilio Bizzi. Servo hypotheses for the biological control of movement. *Journal of Motor Behavior*, 25(3):193–202, 1993.

Josh Merel, Matthew Botvinick, and Greg Wayne. Hierarchical motor control in mammals and machines. *Nature Communications*, 10(1):5489, 2019.

Mehran Mesbahi and Magnus Egerstedt. *Graph theoretic methods in multiagent networks*, volume 33. Princeton University Press, 2010.

Milana P Mileusnic, Ian E Brown, Ning Lan, and Gerald E Loeb. Mathematical models of proprioceptors. i. control and transduction in the muscle spindle. *Journal of neurophysiology*, 96(4):1772–1788, 2006.

T.E. Milner. Adaptation to destabilizing dynamics by means of muscle cocontraction. *Experimental Brain Research*, 143(4):406–416, 2002.

T.E. Milner, C. Cloutier, A.B. Leger, and D.W. Franklin. Inability to activate muscles maximally during cocontraction and the effect on joint stiffness. *Experimental Brain Research*, 107(2):293–305, 1995.

Manfred Morari and Jay H Lee. Model predictive control: past, present and future. *Computers & Chemical Engineering*, 23(4-5):667–682, 1999.

F Mörl, M Günther, J M Riede, M Hammer, and S Schmitt. Loads distributed *in vivo* among vertebrae, muscles, spinal ligaments, and intervertebral discs in a passively flexed lumbar spine. *Biomechanics and Modeling in Mechanobiology*, 19(6):2015–2045, 2020.

Falk Mörl, Tobias Siebert, Syn Schmitt, Reinhard Blickhan, and Michael Günther. Electro-Mechanical Delay in Hill-Type Muscle Models. *Journal of Mechanics in Medicine and Biology*, 12(5):85–102, 2012.

Giulio Mozzi. *Discorso matematico sopra il rotamento momentaneo dei corpi*. nella stamperia di Donato Campo, 1763.

Roy Müller, Daniel Florian Benedict Häufle, and Reinhard Blickhan. Preparing the leg for ground contact in running: the contribution of feed-forward and visual feedback. *Journal of Experimental Biology*, 218(3):451–457, 2015.

Richard M Murray, Zexiang Li, S Shankar Sastry, and S Shankara Sastry. *A mathematical introduction to robotic manipulation*. CRC press, 1994.

Ferdinando A Mussa-Ivaldi. Modular features of motor control and learning. *Current opinion in neurobiology*, 9(6):713–717, 1999.

Eiichi Naito, Tomoyo Morita, and Kaoru Amemiya. Body representations in the human brain revealed by kinesthetic illusions and their essential contributions to motor control and corporeal awareness. *Neuroscience Research*, 104:16–30, 2016.

NASA. *Anthropometric Source Book, Volume I-III*. NASA Anthropometry Project, 1978.

Jong Hyeon Park. Impedance control for biped robot locomotion. *IEEE Transactions on Robotics and Automation*, 17(6):870–882, 2001.

A. Pellionisz and R. R. Llinás. Tensor network theory of the metaorganization of functional geometries in the central nervous system. *Neuroscience*, 16(2):245–273, 1985.

Ilona J. Pinter, Arthur J. van Soest, Maarten F. Bobbert, and Jeroen B. J. Smeets. Conclusions on motor control depend on the type of model used to represent the periphery. *Biological Cybernetics*, 106(8):441–451, 2012.

Jay Pratt, Alison L Chasteen, and Richard A Abrams. Rapid aimed limb movements: age differences and practice effects in component submovements. *Psychology and aging*, 9(2): 325, 1994.

Jerry Pratt and Gill Pratt. Intuitive control of a planar bipedal walking robot. In *Proceedings of the International Conference on Robotics and Automation*, volume 3, pages 2014–2021. IEEE, 1998.

Tony J Prescott, Peter Redgrave, and Kevin Gurney. Layered control architectures in robots and vertebrates. *Adaptive Behavior*, 7(1):99–127, 1999.

Konrad Reif and Rolf Unbehauen. The extended kalman filter as an exponential observer for nonlinear systems. *IEEE Transactions on Signal processing*, 47(8):2324–2328, 1999.

Robert Rockenfeller and Michael Günther. Inter-filament spacing mediates calcium binding to troponin: a simple geometric-mechanistic model explains the shift of force-length maxima with muscle activation. *Journal of Theoretical Biology*, 454:240–252, 2018.

L. A. Rozendaal and A. J. van Soest. Joint stiffness requirements in a multi-segment stance model. In *Proceedings of the XXth Congress of the ISB*, page 622, Cleveland, Ohio, 2005.

TK Rupp, W Ehlers, N Karajan, M Günther, and S Schmitt. A forward dynamics simulation of human lumbar spine flexion predicting the load sharing of intervertebral discs, ligaments, and muscles. *Biomechanics and Modeling in Mechanobiology*, 14(5):1081–1105, 2015.

Soheil Sarabandi and Federico Thomas. A survey on the computation of quaternions from rotation matrices. *Journal of Mechanisms and Robotics*, 11(2), 2019.

Syn Schmitt. *Modellierung und simulation biomechanischer Vorgänge am Beispiel Skisprung*. PhD thesis, Diplomarbeit, Universität Stuttgart, 2003.

Syn Schmitt, Michael Günther, and Daniel F. B. Haeufle. The dynamics of the skeletal muscle: a systems biophysics perspective on muscle modeling with the focus on Hill-type muscle models. *GAMM-Mitteilungen*, 42(3):e201900013, 2019.

Rachael D Seidler, Douglas C Noll, and G Thiers. Feedforward and feedback processes in motor control. *Neuroimage*, 22(4):1775–1783, 2004.

R Shadmehr and FA Mussa-Ivaldi. Adaptive representation of dynamics during learning of a motor task. *Journal of Neuroscience*, 14(5):3208–3224, 1994.

Reza Shadmehr, Maurice A Smith, and John W Krakauer. Error correction, sensory prediction, and adaptation in motor control. *Annual review of neuroscience*, 33:89–108, 2010.

L.F. Shampine and M.K. Gordon. *Computer Solution of Ordinary Differential Equations: The Initial Value Problem*. W.H. Freeman & Co., San Francisco, 1975.

Michael A Sherman, Ajay Seth, and Scott L Delp. What is a moment arm? Calculating muscle effectiveness in biomechanical models using generalized coordinates. In *9th International Conference on Multibody Systems, Nonlinear Dynamics, and Control*, volume 7B, pages DETC2013–13633, 2013.

J Sinclair, Paul John Taylor, Christopher James Edmundson, Darrell Brooks, and Sarah Jane Hobbs. Influence of the helical and six available cardan sequences on 3d ankle joint kinematic parameters. *Sports Biomechanics*, 11(3):430–437, 2012.

J W Smith. The forces operating at the human ankle joint during standing. *Journal of anatomy*, 91(4):545–64, 1957.

Samuel J Sober and Philip N Sabes. Flexible strategies for sensory integration during motor planning. *Nature neuroscience*, 8(4):490–497, 2005.

Dimitar Stanev and Konstantinos Moustakas. Stiffness modulation of redundant musculoskeletal systems. *Journal of Biomechanics*, 85:101–107, 2019.

Katrin Stollenmaier. *Neuro-musculoskeletal Models: A Tool to Study the Contribution of Muscle Dynamics to Biological Motor Control*. PhD thesis, Universität Tübingen, 2021.

Katrin Stollenmaier, Winfried Ilg, and Daniel F. B. Haeufle. Predicting perturbed human arm movements in a neuro-musculoskeletal model to investigate the muscular force response. *Frontiers in Bioengineering and Biotechnology*, 8:308, 2020.

S. Tarbouriech and M. Turner. Anti-windup design: an overview of some recent advances and open problems. *IET Control Theory & Applications*, 3(1):1–19, 2009.

Kurt A Thoroughman and Reza Shadmehr. Learning of action through adaptive combination of motor primitives. *Nature*, 407(6805):742–747, 2000.

Olaf Till, Tobias Siebert, Christian Rode, and Reinhard Blickhan. Characterization of isovelocity extension of activated muscle: a hill-type model for eccentric contractions and a method for parameter determination. *Journal of theoretical biology*, 255(2):176–187, 2008.

Emanuel Todorov, Weiwei Li, and Xiuchuan Pan. From task parameters to motor synergies: A hierarchical framework for approximately optimal control of redundant manipulators. *Journal of Robotic Systems*, 22(11):691–710, 2005.

M. C. Tresch, P. Saltiel, and E. Bizzi. The construction of movement by the spinal cord. *Nature Neuroscience*, 2(2):162–167, 1999.

Ya-weng Tseng, Jörn Diedrichsen, John W. Krakauer, Reza Shadmehr, and Amy J. Bastian. Sensory prediction errors drive cerebellum-dependent adaptation of reaching. *Journal of Neurophysiology*, 98(1):54–62, 2007.

Arthur J. van Soest and Maarten F.. Bobbert. The contribution of muscle properties in the control of explosive movements. *Biological Cybernetics*, 69(3):195–204, 1993.

Johannes R Walter, Michael Günther, Daniel FB Haeufle, and Syn Schmitt. A geometry- and muscle-based control architecture for synthesising biological movement. *Biological cybernetics*, 115:7–31, 2021a.

Johannes R Walter, Patrick Lerge, and Syn Schmitt. Human-centred design: a comparison of ingress motion for two car concepts using a musculoskeletal, digital human body model. In *Stuttgarter Symposium für Produktentwicklung*, page accepted, 2021b.

Tim Wescott. Pid without a phd. *Embedded Systems Programming*, 13(11):1–7, 2000.

Norbert Wiener. *Cybernetics, or Control and Communication in the Animal and the Machine*. Technology Press, 1948.

U Windhorst. Muscle proprioceptive feedback and spinal networks. *Brain research bulletin*, 73(4-6):155–202, 2007.

Isabell Wochner, Danny Driess, Heiko Zimmermann, Daniel FB Haeufle, Marc Toussaint, and Syn Schmitt. Optimality principles in human point-to-manifold reaching accounting for muscle dynamics. *Frontiers in Computational Neuroscience*, 14:38, 2020.

Simon Wolfen, Johannes Walter, Michael Günther, Daniel FB Haeufle, and Syn Schmitt. Bioinspired pneumatic muscle spring units mimicking the human motion apparatus: benefits for passive motion range and joint stiffness variation in antagonistic setups. In *2018 25th International Conference on Mechatronics and Machine Vision in Practice (M2VIP)*, pages 1–6, 2018.

Daniel M. Wolpert. Computational approaches to motor control. *Trends in Cognitive Sciences*, 1(6):209–216, 1997.

Daniel M Wolpert and J Randall Flanagan. Motor prediction. *Current Biology*, 11(18): R729–R732, 2001.

Daniel M Wolpert and Zoubin Ghahramani. Computational principles of movement neuroscience. *Nature neuroscience*, 3(11):1212–1217, 2000.

Daniel M Wolpert and Michael S Landy. Motor control is decision-making. *Current opinion in neurobiology*, 22(6):996–1003, 2012.

Daniel M Wolpert, Kenji Doya, and Mitsuo Kawato. A unifying computational framework for motor control and social interaction. *Philosophical Transactions of the Royal Society of London. Series B: Biological Sciences*, 358(1431):593–602, 2003.

D.M. Wolpert and M. Kawato. Multiple paired forward and inverse models for motor control. *Neural Networks*, 11(7):1317–1329, 1998.

Ge Wu, Sorin Siegler, Paul Allard, Chris Kirtley, Alberto Leardini, Dieter Rosenbaum, Mike Whittle, Darryl D D'Lima, Luca Cristofolini, Hartmut Witte, et al. Isb recommendation on definitions of joint coordinate system of various joints for the reporting of human joint motion—part i: ankle, hip, and spine. *Journal of biomechanics*, 35(4):543–548, 2002.

Ge Wu, Frans CT Van der Helm, HEJ DirkJan Veeger, Mohsen Makhsous, Peter Van Roy, Carolyn Anglin, Jochem Nagels, Andrew R Karduna, Kevin McQuade, Xuguang Wang, et al. Isb recommendation on definitions of joint coordinate systems of various joints for the reporting of human joint motion—part ii: shoulder, elbow, wrist and hand. *Journal of biomechanics*, 38(5):981–992, 2005.

Michael Zeitz. The extended luenberger observer for nonlinear systems. *Systems & Control Letters*, 9(2):149–156, 1987.