

A unifying object-oriented methodology to
consolidate multibody dynamics computations
in robot control

Am Fachbereich Informatik der
Technischen Universität Darmstadt
eingereichte

Dissertation

zur Erlangung des akademischen Grades eines
Doktor-Ingenieurs (Dr.-Ing.)

von

Dipl.-Phys. Robert Höpler
(geboren in München)

Referenten der Arbeit: Prof. Dr. Oskar von Stryk
Prof. Dr.-Ing. habil. Dieter Bestle

Tag der Einreichung: 23. 07. 2004

Tag der mündlichen Prüfung: 06. 08. 2004

Acknowledgements

This work would not have been possible without the support of a series of people. First of all I would like to express my gratitude to Prof. Dr. Oskar von Stryk who continuously encouraged and supported me to finish this thesis. I am very grateful to Prof. Dr.-Ing. habil. Dieter Bestle for his kind interest in my work, invaluable comments, and for revising this thesis as a co-reviewer.

Many of the results presented were developed during my time as a junior research assistant at the Institute for Robotics and Mechatronics of Prof. Dr. Gerd Hirzinger at the German Aerospace Center (DLR) in Oberpfaffenhofen. The work in his institution in the team of Dr.-Ing. Johann Bals prepared the ground for this thesis. Grants from, and a strong cooperation with KUKA Roboter GmbH and Amatec Robotics GmbH, both Augsburg, Germany, opened my mind for practical concerns in robotics. Especially Dr. Martin Weiß and Dr. Dietmar Tscharnuter introduced me to the tough world of industrial robotics.

I was lucky having many teachers and friends in the last five years. It was a pleasure working with each of them, showing me different facets of modeling and control in the robotics domain. Michael Hardt, PhD, taught me the useful spatial operator description of multibody dynamics for legged robots. Pieter J. Mosterman, PhD, showed me that a model might be more than mathematical equations. I got in touch with the field of robot control engineering through my colleague Michael Thümmel. Martin Hörmann and Dr. Max Fischer opened my eyes for software engineering issues and the depths of C++. Maximilian Stelzer and Markus Glocker were optimal in trajectory optimizations. My five years at DLR and my time at the Technische Universität Darmstadt would not have been so enjoyable without my colleagues, especially my roommate Christian Ballauf, and my colleagues Gerhard Schillhuber and Astrid Jaschinski, sharing lots of discussions and coffee with me. Finally I would like to express my admiration for Dr. Richard Schwertassek, who filled me with enthusiasm for the area of multibody system dynamics.

Darmstadt, 23th July 2004

Contents

Notation	VI
Outline	IX
1 Introduction	1
1.1 Motivation	1
1.2 Contents and contributions	4
1.3 Literature survey	6
2 Principles of dynamics computations: modeling robots on mechanical, abstract, and algorithmic level	8
2.1 Dynamics preliminaries	9
2.1.1 Classical mechanics of a single gyrostat	9
2.1.2 Coordinate representations	11
2.2 Paradigm for abstract robot model specification	16
2.2.1 Component specification model	17
2.2.2 Robot specification model	26
2.3 Relating recursive dynamics algorithms to dataflow	28
3 Multibody dynamics algorithms for robots	34
3.1 Fundamental algorithms for rigid manipulators revisited	34
3.1.1 Recursive algorithms—general considerations	34
3.1.2 Inverse dynamics	39
3.1.3 Forward dynamics	42
3.2 Advanced and new dynamics algorithms	44
3.2.1 New elastic joint inverse dynamics	44
3.2.2 Obtaining sensitivity information	56
3.2.3 Inverse dynamics for closed chains	58
3.2.4 Operational space dynamics	61

4	Methodology for operational robot models	65
4.1	Classes for model specification	66
4.2	Mapping robot models	68
4.2.1	Mapping to a dataflow network	69
4.2.2	Model transformation and optimization	74
4.3	Reusable components of robot models in control	76
4.3.1	Basic solver interfaces	76
4.3.2	Interfaces for algorithm building blocks	77
4.4	Aspects relevant to realizing a dynamics framework	79
5	Selected applications in research and industry	83
5.1	DLR light-weight robot inverse dynamics	84
5.2	Modeling a palletizing robot	87
5.3	Calibration models of manufacturing robots	89
5.4	Trajectory optimization of legged robots	91
A	Glossary	99
B	Useful identities	105
	Bibliography	107

Used notation and symbols

The rules for the typesetting and the location of indices are inspired by the conventions used in the book of Roberson and Schwertassek [115] and explained below in Table 1. Used symbols and their meaning are listed alphabetically in Table 2 on the next page.

TABLE 1: Notational conventions adopted in this work.

entity	typeset/convention	scheme	remark
physical vectors and dyadics (1 st and 2 nd rank tensors) w.r.t. a certain reference frame/point y .	roman, bold, left subscript index optional	${}_y\mathbf{x}$	if y omitted then y is center of mass
scalar or matrix (n-dim. array of numbers)	italic	x	Dimension from context
one-dimensional matrix, components	round brackets around array	(\cdot)	per default column matrix
two-dimensional matrix, components	square brackets around array	$[\cdot]$	
matrix representation of vector/dyadic ${}_y\mathbf{x}$ resolved w. r. t. a vector base ${}_z\mathbf{e}$	italic, left superscript index	${}_y^z x$	
block matrix	roman, greek or latin, mostly capital	X	prominently blocks $\in \mathbb{R}^6$ and $\in \mathbb{R}^{6 \times 6}$
individual matrix entry	right subscript index set	x_{ij}	
block i, j of block matrix	right subscript index set in square brackets	$X_{[i,j]}$	dimension from context
matrices concerning MBS eq. in <i>state</i> or <i>joint space</i> form	calligraphic, capital	\mathcal{X}	
matrix in coordinate representation C	left upper index in $\langle \rangle$ brackets	$\langle^C x$	C is a letter abbreviating the representation, see 2.1.2 on page 11
matrices involved in inboard and outboard sweeps in spatial recursions	left upper index. \uparrow denotes outboard, \downarrow inboard matrices	x^\uparrow, x^\downarrow	
class identifier	sans serif	Class	
message symbol in a dataflow protocol of recursive algorithms	sans serif	\times_n	x denotes a matrix, n a port index
constructs in a programming language	typewriter	Code	
graph sets	Fraktur	\mathfrak{G}	

TABLE 2: List of important symbols.

symbol	entity	unit/dim.	explanation/definition
$\langle A \rangle$	absolute coordinate representation		denotes <i>absolute</i> coordinate representation for the following matrix
e	unit column vector	$\in \mathbb{R}^N$	
$\langle B \rangle$	body-fixed coordinate representation		denotes <i>body-fixed</i> coordinate representation for the following matrix
c	index denoting center of mass (frame) of a rigid body		
C	connectivity matrix		see (2.28)
\mathcal{C}	joint space matrix of gyroscopic forces	\mathbb{R}^{N_d}	
Δ	block matrix of stacked N_p relative spatial joint velocities	$\in \mathbb{R}^{6N_p}$	
E	energy	[J]	right lower index denotes type
${}_x e$	vector basis valid in frame F_x		Right handed orthonormal system.
\mathcal{E}_ϕ	manipulator rigid outboard shift transformation operator	$\in \mathbb{R}^{6N_p \times 6N_p}$	$N \times N$ blocks
\mathcal{E}_ψ	articulated shift transformation operator	$\in \mathbb{R}^{6N_p \times 6N_p}$	
f	spatial force matrix	$\in \mathbb{R}^6$	see Table 2.1
f	block matrix of stacked N_p spatial force matrices	$\in \mathbb{R}^{6N_p}$	
\mathbf{F}	force vector		
F_n	coordinate frame		= tuple of a reference point \mathbf{O}_n and a vector base $\{\mathbf{O}_n, {}_n e\}$
\mathcal{G}	joint space matrix of gravitational forces	\mathbb{R}^{N_d}	
H	joint Jacobian	$\in \mathbb{R}^{6 \times m}$	m is number of joint's motional d.o.f.
H	stacked joint Jacobian matrix	$\in \mathbb{R}^{6N_p \times m}$	m is total number of motional d.o.f.
I	unit matrix		
i	index denoting the inertial reference system		
$\langle I \rangle$	inertial coordinate representation		denotes <i>inertial</i> coordinate repr. for following matrix
${}_x \mathbf{J}$	tensor of moments of inertia w. r. t. a reference point \mathbf{O}_x	$\in \mathbb{R}^{3 \times 3}$	footnote on page 9
\mathcal{J}	manipulator Jacobian matrix	$\in \mathbb{R}^{6 \times N_d}$	see Section 3.2.2.1
\mathcal{J}_c	constraint Jacobian	$\in \mathbb{R}^{N_{cc} \times N_d}$	see Section 3.2.4
\mathbf{l}	vector of internal angular momentum		
\mathbf{L}	vector of angular momentum		
Λ	operational space inertia	$\in \mathbb{R}^{N_{cc} \times N_{cc}}$	see (3.62)
m	mass	[kg]	
M	spatial inertia matrix	$\in \mathbb{R}^{6 \times 6}$	see Table 2.1
M	stacked block-diagonal inertia matrix	$\in \mathbb{R}^{6N_p \times 6N_p}$	
\mathcal{M}	joint space manipulator mass matrix	$\in \mathbb{R}^{N_d \times N_d}$	

symbol	entity	unit/dim.	explanation/definition
N_x	number/dimension of s.t. denoted by x	$\in \mathbb{N}$	dimension of column vector
N_{cc}	total number of contact constraints	$\in \mathbb{N}$	
N_d	total number d.o.f. in MBS	$\in \mathbb{N}$	
N_p	total number of interaction ports in a multi-body component diagram	$\in \mathbb{N}$	
N^+	total number of bodies (or any other entities) in MBS	$\in \mathbb{N}$	
ω	vector of angular velocity		
Ω	spatial angular velocity matrix	$\in \mathbb{R}^6$	see Table 2.1
$\mathcal{O}(N^x)$	the complexity of a numerical multibody algorithm.		N is the number of bodies, x the order
P	articulated body inertia matrix	$\in \mathbb{R}^{6 \times 6}$	
$\phi_{y,x}$	rigid body transformation matrix from frame F_x to F_y	$\in \mathbb{R}^{6 \times 6}$	see (2.18)
Φ	stacked rigid manipulator velocity transformation	$\in \mathbb{R}^{6N_p \times 6N_p}$	see (2.35)
Π_x	spatial momentum matrix w. r. t. a frame F_x	$\in \mathbb{R}^6$	see Table 2.1
Π_i	internal spatial momentum matrix	$\in \mathbb{R}^6$	see Table 2.1
$\psi_{y,x}$	articulated shift matrix from frame F_x to F_y	$\in \mathbb{R}^{6 \times 6}$	see Section 3.1.3
$\psi_{i,i-1}$	articulated shift matrix from frame F_{i+1} to F_i	$\in \mathbb{R}^{6 \times 6}$	see Section 3.1.3
Ψ	stacked articulated manipulator force transformation	$\in \mathbb{R}^{6N_p \times 6N_p}$	see Section 3.1.3
q	column vector of joint position variables, in case of 1 dof joints	$\in \mathbb{R}^{N_d}$	
q_p, q_v	column vectors of positional and velocity state variables		
q_a	column vector of acceleration variables		
$\mathbf{r}_{x,y}$	position vector from location \mathbf{O}_x to \mathbf{O}_y		
2R_1	rotation matrix	$\in \mathbb{R}^{3 \times 3}$	Rotates from F_2 to F_1 , see (A)
\mathcal{R}	diagonal matrix of gear ratios		see Section 3.2.1
τ	transposed matrix		r.u. index
T_ϕ	stacked rigid body transfer matrix	$\in \mathbb{R}^{6N_p \times 6N_p}$	
$\boldsymbol{\tau}$	torque vector		
θ	column vector of drive positions		
u	column vector of generalized applied forces		
\mathbf{v}	vector of translational velocity		
V	spatial velocity matrix	$\in \mathbb{R}^6$	see Table 2.1
$1, 2, 3, n$	indices denoting interaction ports in multibody entities		
$+$	symbol denoting the total MBS		

Outline

The work aims on resolving some problems arising from the intermingling of mathematical modeling of robotic mechanisms, numerical schemes, and implementation and integration into a whole robot control software architecture in common practice. The solution proposed in this thesis is a carefully designed object-oriented class hierarchy supporting the robot control engineer in the processes of specifying, implementing, and performing multibody computations required in robot control systems.

The standard approach to multibody computations is to realize separate models for different applications of the same robot resulting in stand-alone numerical schemes—of course relying on individual idealizations and formalisms. This common procedure is nowadays no more sufficient as the demand for model-based robot control is growing and integration of an increasing number of schemes becomes a key enabler. The problem of integration is tackled here by means of an object-oriented methodology consolidating the several conceptual levels of robot modeling, involving abstract, mechanical, numerical, and software representations of the same system under consideration, the robot mechanism.

The common ground for the numerical schemes as well as their implementation is a new paradigm for the precise description of the robot's mechanical components and the desired computations. The numerical and mathematical description of the governing equations is captured by a new port-based extension of the mathematical framework of symbolic spatial operators. A series of multibody algorithms, standard ones as well as some new for special purpose are recasted in object-oriented form and categorized according to this paradigm. A new dataflow-driven model of computation is proposed for the efficient implementation of recursive algorithms, which directly applies to object-oriented software systems. This alleviates the coupling of several models, as shown by some selected example applications, and justifies the effort of applying this methodology for robot dynamics computations.

Chapter 1

Introduction

1.1 Motivation

The description of robotic motion, comprising fast movements of manufacturing robots as well as legged locomotion of a humanoid robot, requires models describing approximations of particular aspects of reality. Models are indispensable when robots move and when they interact with their environment. Processing and interpretation of perception data or strategies to autonomously execute predefined tasks, all must be specified and performed in terms of abstractions of the real robot and its world. When speaking of a model, however, its final representation will be segments of code running on one or more digital computers that control the energy applied to the actuators.

A very powerful abstraction amenable to further analytical treatment as well as to implementation on digital computers is a mathematical representation of general physical laws and processes governing the physical properties of the robot. We further refer to such a representation as a *mathematical model*.

The dominant features of reality to be modeled in robotics are the macroscopic mechanical properties of a robot's mechanism. This comprises its desired motion, its actuators, sensors and interaction with the physical environment a robot exists in. A mathematical model describing all this is referred to as a *mechanical model*.

A computer program or portions of code that can be executed are called a *computational model* if they represent certain aspects of a mathematical or mechanical model. The most prominent is the numerical calculation of kinematical and dynamical quantities determining robot motion.

If relevant properties of a model are caught by an abstract representation, this representation by itself is a model, referred to as *abstract model* or a *model specification*. The model specification is sufficiently detailed, if one can deduce all required properties of the model from the specification.

The objective of this work is twofold. First, a seamless and modular way is shown how computational models emerge from corresponding abstract mechanical models. This problem belongs to the domain of *computational mechanics*. The resulting computational models provide the physical state of the robot required by hard- and software controlling its motion, cf. Figure 1.1. It should be noted, that models can be involved in each part of a robot software down to the level of actuator control. Here, models are discussed which describe aspects of the complete mechanism. Second, design principles are proposed how the involved computational models can be defined and decomposed to integrate well in a robot control system architecture. This process of integration into a large software-based system requires methods from the domains of multibody system dynamics, systems and software design, and applied robotics, each imposing specific constraints on the proposed solution.

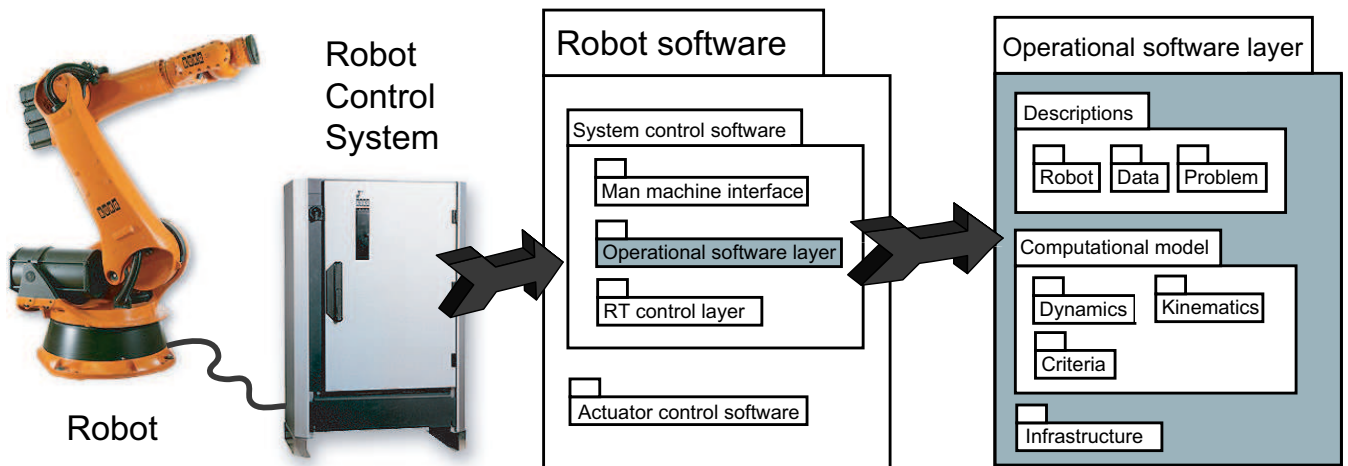


FIGURE 1.1: Categorization of a robot software system according to [73]. This work concentrates on those abstract and computational models living mainly in the operational software layer on the right side.

The science of how to create computational models of complex mechanical systems is called multibody system (MBS) dynamics. The main objective is to investigate the evolution of physical quantities with respect to time, briefly denoted by *the dynamics*, to establish equations of motion and to derive numerical schemes for their efficient evaluation. The astonishingly complex non-linear dynamics of this kind of systems stems predominantly from the coupling of small and simple systems to larger ones, usually bodies connected by joints. The research field of rigid MBS has reached a maturity in the last three decades, indicated by several landmark books by Wittenburg [149], Featherstone [40], Roberson and Schwertassek [115], Murray et al. [98], to name only a few. An immense variety of formalisms and algorithms to create computational robot models for a vast range of possible applications has been developed by scientists in the field of computational mechanics, see, e. g., Schiehlen [123].

At first sight it would be sufficient to establish the equations of motion of the robot just symbolically and, if available, apply a sufficiently powerful symbolic formalism to derive all desired further equations and to generate code. There are some approaches, e. g., [95], that follow this idea, but fail in many real-world applications due to the following practical constraints. In case of a robot control system, which is often an embedded system with hard real-time constraints, one is faced with robotics domain specific constraints:

- (i) *Efficiency*: There can be various possibilities to solve a system of equations numerically. Each variant may be efficient for a certain regime. The most prominent example in robot dynamics are so-called $\mathcal{O}(N)$ -formalisms and composite rigid body algorithms. The former outperforms the latter only for systems with more than five bodies [91]. This valuable domain-knowledge is hard to capture in a completely automatized procedure. Furthermore in a complex software system the required reuse of results imposes additional constraints on generated code not implicated in a symbolic approach.
- (ii) *Limited resources*: Resources might be limited, so either symbolic code generation requiring a compile-to-code step is not possible. Or in presence of switching or hybrid behaviour it is hard to store models for each system state due to combinatorial explosion.
- (iii) *Maintenance*: Totally automatically derived code is hermetic in several senses, because in most cases the numerical scheme itself is a neglectable amount of code in comparison to interfacing the results to the rest of the application, it is often impossible to trace errors and to debug because there is no transparent relationship between the equations and the code.

- (iv) *Safety*: Robotics is an extremely safety critical domain because man-machine interaction mandates avoidance of damage and injury to humans. Still it is hard to formally verify the correct function of completely automatically generated code and often manual coding is still preferred in practise.

Each dynamics algorithm has its strength and advantages but relies on certain modeling assumptions and axioms. In implementation, e. g., in a real-time control system for one certain type of robot, maximum performance is mandatory and the system designer is forced to integrate various algorithms and balance precision, generality, run-time characteristics, and implementation effort of a model or formalism. Efficient solutions often exist but, which is typical of robotics, often are problem specific, see e. g., [134, 150]. In research and industrial environments this balancing currently is repeated each time a new type of robot or control system is developed depending on the current state-of-the-art, and often software is developed by manual coding. One flexible and efficient solution to this problem is re-use and configuration instead of new implementation.

The central topic of this work are these numerical schemes for multibody computations with emphasis on the class of *recursive* methods, because they have shown to be most efficient. The idea is to go beyond 'pure' implementation of numerical algorithms, where function and efficiency are dominant aspects, and extend to a systems view. A dynamics algorithm inevitably is part of a larger problem architecture. It exists within a software and hardware architecture as well, where not necessarily mathematically described aspects are of concern to the user. The classical simulation environment used by mechanical engineers is just one concrete realization by means of time-integration. Obvious aspects are the interaction with a changing physical environment, communication, flexibility, time-to-market, especially in the field of real-time robot control. To put it in other words: A mathematical formalism and its realization is one way to model a complex technical system. Hatley and Pirbhai stress this important aspect that modeling may not be restricted to a numerical formalism and its implementation. A complex technical system requires to model the requirements and the design:

[...] components that make up a system—both hardware and software components—are highly interrelated, and, in order to successfully perform their intended function, they must integrate well. The system specification process, therefore, must define the system as a whole, as well as its partitioning into hardware and software components. It must define *what* problem the system is to solve (its requirements) and *how* that system is to be structured (its architecture or design structure). [51]

Here we follow a similar idea to focus on design principles to systematically solve the problem of performing multibody computations in robot control. Structured methods can help to approach the vision of a unified MBS model in a robot control system, which does not mean the naïve concentration on one outstanding formalism. The focus is not just the model itself, but also its *behaviour* and its *meaning* in various contexts. Though the procedure is the generation code solving MBS equations from a simple description of the system elements, the viewpoint is that of a general mapping from multibody-formalisms to a space of abstract (software) entities without sacrificing the power of specialized domain-specific MBS algorithms.

There are several possibilities to structure this kind of problem. This work follows object-oriented (oo) modeling and component-oriented design to simplify model and code generation and software reuse. One main motivation to employ object-oriented analysis is that the analogy between the physical model and the software model is very fruitful [1], because formulating a problem in terms of notions from the problem domain increases comprehensibility. An object-oriented model is a key enabler for another reason: it can

be combined with general principles of software engineering, such as separation of concerns, correctness, reliability, and robustness [37].

As the number of robots and other automation subsystems grows, integration becomes increasingly difficult. Software integration costs alone for the United States' robotics industry are estimated at \$1 billion annually [114]. Therefore dynamics computations must integrate in software and hardware architectures with, e. g., hard timing constraints and limited resources. Embedded code will increasingly consist of interacting software components [139, 82]. Chen and Yang [25] report the need to integrate several computational models of the robot in one application. Their scope is to use the same algorithms and codes for simulation as well as for control purpose. The problem of covering the whole of multibody algorithms in a robot control system (RCS) is analogous to MBS equations themselves where the high complexity stems from the interconnection of smaller blocks of low or medium complexity. Combining relatively simple blocks containing complicated parametrization and inner dynamics leads to very heterogeneous architectures being non-trivial to design, implement, debug, and maintain. Especially specification and parametrization of the computational dynamics models are often underrated, but are crucial points in code generation and formalism transfer and of real practical relevance.

One further goal of this work is to provide a *reusable context* for components performing multibody computations, i. e., to enable the reuse of algorithm design *and* code. A lack of reusability is only partially a problem of a lack of documentation. By virtue of so-called frameworks object-oriented systems reach a maximum of reusability [69]. One challenge is that frameworks are among the most complex of all software design approaches [44]. A framework should apply to all imaginable applications in the domain under consideration. This requires profound domain knowledge, i. e., mechanics of robots, multibody formalisms and software architectures, to create a comprehensive design of flexible and extendible characteristics.

1.2 Contents and contributions

Main thesis

The preceding motivation leads to the main thesis which is the attempt to reduce the gap between the heterogeneous worlds of

- (i) modeling (model specification) of robotic mechanisms, comprising abstraction and meaning of the abstract representations,
- (ii) existing and upcoming multibody formalisms and algorithms and their efficient implementation and coupling, and
- (iii) numerical requirements from, e. g., robot control schemes, simulation, and trajectory optimization methods,
- (iv) real-world robotic software applications, demanding (a) modular and extensible, (b) interoperable and (c) light-weight code running off-line or on real-time systems.

The vision drawn in this work is a *general purpose robot dynamics framework* to support a robot control design engineer in (i) robot model specification, (ii) automatic code generation *and* manual implementation from an optimally chosen mechanical model and multibody formalism, and (iii) integrating the computational models and components in an evolving software architecture for control, optimization, and

simulation. The focus is on basing on a sound mathematical formalism, using object-oriented design principles and reaping maximum performance. The desired result from this modeling of robot *models*, however, is not a *general purpose multibody program* demanding the least domain knowledge possible from the user and covering as many problem classes as possible.

Modeling robotic mechanisms and algorithms

Without multibody domain expertise it is hard to define practical requirements for software architectures such as frameworks. Ideally the developed approach has to cover all existing and upcoming cases of application. Chapter 2 formulates the relevant aspects of multibody computations in robot control ranging from mechanics, to formalisms and algorithms. The physics (classical mechanics) of one body and several bodies are discussed in Section 2.1 to show the importance of coordinate representations, *dynamics formalisms* [115] and their realizations.

Section 2.2 takes a closer look from an abstract, high-level viewpoint with emphasis on topological issues and structured methods [51]. The complexity arising in MBS model *specification* is investigated. A general paradigm for the specification based on entity-relationship-attribute (ERA) paradigm [26] and associated semantics are developed. Identifying common *implicit* assumptions in abstract models of robots improve formalism transfer and consistency. To grasp these assumptions this work introduces the new notion of a *MBS context*. Entities relevant for code generation and implementation are identified which form the basis of the methodology.

To deal with the vast number of numerical multibody algorithms a rough classification is proposed in Section 3.1.1 to embed the schemes in a 'component space' spanned by chosen formalism, coordinate representation and desired output values. A number of representative existing multibody algorithms covering a large range of applications are introduced to this classification scheme accompanied by some new specialized algorithms. Several new useful mathematical MBS expressions, e. g., for control applications are derived.

The crucial symbolical description of multibody equations is based on the well-known *spatial operator algebra* (SOA) by Rodriguez et al. [117]. In order to overcome the restrictions of this powerful mathematical formalism the Port-Based Spatial Operator Algebra is introduced in Section 2.3 which reaps the advantages of intuitive object-oriented modeling and implementation techniques and the power and expressiveness of the symbolic operator formulation. This allows for a uniform specification and presentation of SOA operator expressions of general multibody-systems including more general components and topologies. The ability to establish spatial operators from topological and component properties pays off in sections 3.1 and 3.2 either for symbolic manipulation or object-implementation while preserving the valuable algebraic properties of the SOA. A dataflow interpretation of recursive algorithms in Section 2.3, which is given for the first time, prepares the ground for efficient code-generation and implementation of this class of algorithms.

Operational architecture and applications

The representation of robot model specifications in an object-oriented system is investigated in Section 4.2. This section introduces an ontology of interrelated classes describing the mechanism, the desired numerical algorithm and modeling assumptions. To reach a maximum in generality, this work proposes a clear separation between model specification and transformation issues such as code generation. This enables the software designer including characteristics that are concern of the user and avoids placing

constraints on the design level, e. g., choice of coordinate representation or the time and place of code generation. An abstraction level being too high for this purpose [51].

The specification model is subjected to several kinds of mapping in Section 4.2. The discussed mappings are code generation for dataflow networks, model optimization and textual representation. A new object-oriented architecture is proposed in Section 4.2.1 to create executable algorithm objects from a given specification model to decouple the various components. This alleviates extending applications but still allowing interoperability between new and old components. The user of the architecture *may* at every level of abstraction interact with the model specification if required by the concrete application, e. g., optimize topological properties, without the need to go down on lowest level, the equation level. In Chapter 4.3 some fundamental algorithmic robot dynamics building blocks required by control applications, multibody scenarios and boundary conditions are formulated in terms of tools developed in Section 2.2. A framework becomes concrete by choosing an object-oriented programming language [70, 69]. Requirements for a C++ realization are discussed in Chapter 4.4. The architecture presented is applied to several problems arising from industrial and scientific problems in Chapter 5 substantiating the applicability of this work especially in heterogeneous robot control applications.

Finally several appendices containing lists of used symbols, a small glossary, an index and tables of useful mathematical identities hope to prevent the reader from getting lost in notation and connotations.

1.3 Literature survey

Software for multibody computations in robotics

The main focus is on symbolic recursive formalisms to compute the highly non-linear dynamics of robots for they are known to be numerically efficient, which is indispensable in real-time control of robots. The first recursive technique reported was developed by Vereshchagin [144] in 1974. For a review and classification of recursive schemes see Jain [60].

The existing packages can be subdivided according to the offered computations and the type of application, either an executable generating code, or a programming library usually realized as collections of source code. The main objectives of nearly all commercial and non-commercial tools are the generation of equations of motion from simple input model description and time integration (simulation) of the differential equations.

For a survey of *dynamics formalisms* developed until 1988 refer to the book of Roberson and Schwertassek [115], a number of packages prominently for simulation purpose are reviewed in [123]. The disadvantages of the large commercial general purpose tools are lack of computational efficiency and flexibility, consumption of resources and high effort to integrate own optimized and specialized components, all paramount when migrating to embedded systems like robot controls. This work does not intend to provide yet another simulation package, but to support the robot domain specialist in implementing various methods and formalisms.

This basic idea of providing components for multibody computations is taken up in the package AUTOLEV [122]. This collection of functions helps the multibody dynamics domain specialist in formulating the equations of motion, but is restricted to Kane's equations and generation of a complete simulator code. The main advantage is full control over the equation formulation process, inevitable when exploiting maximum performance.

The package MOBILE [74] for simulation of various types of mechanical systems is based on object-oriented principles and implemented in C++. A component oriented design helps the domain engineer in

describing the multibody system properties and code generation by means of source code. This approach focuses on time integration, specialized for medium to heavily constrained mechanical systems. These aspects are not prevailing in robot control.

DYNAMECHS is a multi-purpose collection of robot dynamics algorithms implemented in C++ by McMillan et al. [92]. It is designed for simulation of under-water vehicles and legged robots and comprises some popular dynamics algorithms for certain types of multibody systems. It is merely driven by implementation of multibody formalisms but does not emphasize a high-level design.

A C++ package intended for use in robotics is ROBOOP [46]. It comprises several classes for kinematics and dynamics computations, but is restricted to certain types of robots and choices of coordinates. A similar library is by Corke [28], implemented in Matlab scripting language. DARTS [61] is a collection of functions written in C, forming an engine used for robot simulations especially for space applications [16]. The idea to use the same algorithms and codes for simulation as well as for control purposes has been reported by Chen and Yang [25]. Their approach is restricted to the class of tree-structured robots and one special dynamics formalism.

There is a great number of commercial tools mostly dedicated to simulation, i. e., time integration, of general multibody systems, often providing export of symbolic code, too. To name only a few prominent examples: SIMPACK [120] is based on an $\mathcal{O}(N)$ -formalism, ADAMS [103] based on Kahn's equations. SIMMECHANICS [89] by The Mathworks are multibody blocksets which enables control system design for mechanical systems within the SIMULINK environment.

Software architectures for robot systems

From the perspective of a high-level software design for robot control software the project *open source robot control software* (OROCOS) [22, 88] is closest to the ideas of modeling and software design developed in this work. The long-term objective is to provide generic and public-domain C++ software components for all concerns of robot control. The block of kinematics and dynamics computations was not developed during the period this work was performed.

SMART [11] is a component-oriented control architecture for tasks on a higher level than model computations such as collision avoidance, trajectory generation, integration of sensor and haptic devices, especially for the field of teleoperated systems. It is possible to adapt the software system by reconfiguring various modules representing operational modes. Code is generated, compiled, downloaded and initialized including a re-synchronization of the robotic system even on a system with several CPUs.

RIPE by Miller and Lennox [93] is a set of C++ base classes intended to represent a robotic system, by base classes 'WorkPiece', 'Station' and 'Device'. Those generic classes can be derived by a user to implement specific features of a real system.

Chapter 2

Principles of dynamics computations: modeling robots on mechanical, abstract, and algorithmic level

Abstraction is not just simplifying or eliminating detail, but focusing on specific details. The introduction indicated that a robot *model* has different meanings depending on the considered level of abstraction and perspective. In the context of robot control certainly the model closest to physical reality, i. e., the representation containing most relevant detail, is a mathematical representation of robot kinematics and dynamics. Most control schemes require information about the actual physical state of the controlled machine. On the one hand, measurements reflect true aspects of the current state, on the other hand physical-based models of the technical system allow prediction. Measurement and prediction are two complementary approaches to provide this kind of information:

[...] chances of establishing a good model depend strongly on a deep understanding of the physical-technical processes of the object to be modeled. A good model means a representation of mechanical properties and therefore a good correspondance to practice and its measurements. [111]

The meaning of model here is restricted to the notion of a mechanical model, i. e., denoting a mathematical representation of physical laws and processes governing the motion of the robot, crucial in model-based control schemes often requiring some information about the dynamics.

This chapter forms the fundamental analysis of requirements as to *what* the methodology has to provide, by summarizing the basic ideas of physical modeling and dynamics formalisms for multibody computations used in the field of robotics. Wittenburg [149] stresses that a concise method to state a formalism and the system equations is inevitable when comparing and classifying algebraical and numerical properties in multibody system dynamics. Among the many choices reported in literature this work borrows heavily from the spatial operator algebra (SOA) by Rodriguez et al. [117]. This (i) provides a uniform description and analysis of existing algorithms, (ii) reveals structural properties of the involved mathematical quantities—invaluable in the object-oriented modeling process in Section 2.2.2—and (iii) presents a powerful means for derivation of new algorithms, e. g., [60].

2.1 Dynamics preliminaries

A multibody system is the idealization of a mechanical system by a collection of interacting material bodies. Interaction is modelled by mechanisms constraining the relative motion (*joints*) between contiguous bodies and external forces applied through springs, etc. [149]. The most successful mechanical model of robots has been proved to be a collection of rigid joint-connected bodies with prominently holonomic, scleronomic constraints between the bodies [98].

The major ingredients of multibody computations depend strongly on the desired application. For robot control purpose so-called inverse models and the topics from the following list are of major concern according to Roberson and Schwertassek [115], Wittenburg [149] and will be discussed in the following:

- (i) physical properties of the robotic mechanism
 - (a) mechanical properties of the components (bodies, springs, ...)
 - (b) system topology = interconnection and interactions between the components
- (ii) applied dynamical formalism, e. g., those presented in [115, 72]
- (iii) set of dependent variables and coordinate representations
- (iv) type and motion of reference frames (kinematic and dynamic, base frame) [115]

In some cases, especially in time integration, additionally the initial configuration and state of the mechanism is required. A crucial fact in robotics is the physical environment of the robot. Is it mounted on the ground or does it operate in free-floating mode? Is it subject to external forces such as gravity? Is it cooperating with other mechanisms or robots? Or is it just favourable to describe the motion w. r. t. a certain reference frame? Which physical values can be measured?

The basis of the description of the dynamics of coupled mechanical systems is a description of the equations of motion of a single rigid body. The following section describes the equations of motion of an unconstrained rigid body, including internal angular momentum, and introduces the notion of a *coordinate representation*, which will turn out to play a crucial role in design and efficient and reliable implementation of multibody algorithms. The presentation follows in parts that in [115] and takes a Eulerian viewpoint.

2.1.1 Classical mechanics of a single gyrostat

A rigid body containing internal angular momentum, e. g., stemming from an embedded rotating mass, is called a *gyrostat* [149]. In this section it will be sufficient to rely on the setup in Figure 2.1 showing a gyrostat with several Cartesian frames¹⁾: one inertial reference frame F_I , one arbitrary reference frame F_0 and three frames fixed to the body, a center of mass frame F_c and two body-fixed frames F_A and F_B . For the moment the motion is considered unconstrained and free of external forces such as gravitation. The body has constant mass m , a constant inertia dyadic²⁾ w. r. t. F_c ${}_c\mathbf{J}$, and the vector of internal relative

¹⁾ A Cartesian frame in Euclidean space is characterized by the *location* of its origin \mathbf{O} and the *orientation* of three orthogonal(-normal) axes $\mathbf{e} = \{\mathbf{e}_1, \mathbf{e}_2, \mathbf{e}_3\}$ forming a dextral system, a vector base. The symbols used throughout this work are F_x or $\{\mathbf{O}_x, \mathbf{e}_x\}$ to denote location and orientation explicitly. All frames used in this work are Cartesian.

²⁾ A continuum distribution of mass results in a tensor of moments of inertia ${}_c\mathbf{J}$ with matrix components ${}_cJ_{\alpha\beta} = \int_{body} \rho(x) \cdot (x_\alpha^2 \delta_{\alpha\beta} - x_\alpha x_\beta) \cdot dV$, $\alpha = \{1, 2, 3\}$ when resolved w. r. t. a body-fixed basis. $\rho(\mathbf{x})$ is the mass density. The formed matrix is symmetric positive-definite.

angular momentum w. r. t. F_c is ${}_c\mathbf{l}$. Please note, that in this work Cartesian tensors of first and second rank are consistently referred to as *vector* and *dyadic*, in contrast to m -dimensional *matrices*, except for matrices $\in \mathbb{R}^{m \times 1}$ which are referred to as *column vectors*. Bold typeface ${}_1\mathbf{x}$ indicates a tensor with a reference point O_1 , italic typeface ${}_1^2x$ denotes a matrix representation of ${}_1\mathbf{x}$ w. r. t. vector basis ${}_2\mathbf{e}$.

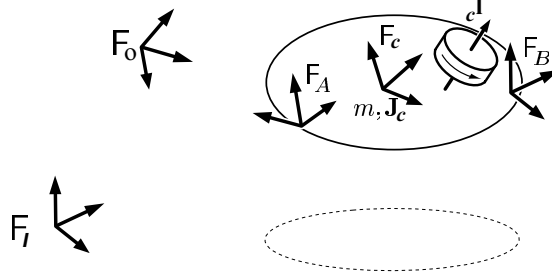


FIGURE 2.1: Schematic setup of one inertial frame F_I , one reference frame F_0 and two arbitrary frames F_A and F_B attached to a gyrostat with center of mass frame F_c . Important to note that only F_I must be inertial. The tilted cylinder symbolizes the internal angular momentum ${}_c\mathbf{l}$.

The fundamental laws of rigid body motion by Newton and Euler state that the net force \mathbf{F} and net torque ${}_I\boldsymbol{\tau}$ on the rigid body³⁾ equal the absolute total time derivatives of its linear and angular momentum ${}_I\mathbf{p}$ and ${}_I\mathbf{L}$:

$$\mathbf{F} = \frac{d{}_I\mathbf{p}}{dt}, \quad (2.1)$$

$${}_I\boldsymbol{\tau} = \frac{d{}_I\mathbf{L}}{dt}. \quad (2.2)$$

Please note for the vectors ${}_I\boldsymbol{\tau}$, ${}_I\mathbf{p}$, ${}_I\mathbf{L}$ the frame F_I is used as a reference, which *must* be an inertial frame. Any tensorial quantity tied to an inertial frame is called *absolute*. A tensor whose reference is non-inertial is called *relative*. Analogously an *absolute derivative* is the derivative of a vector calculated w. r. t. to an inertial frame (usually symbolized F_I). An absolute derivative is denoted by $\dot{\mathbf{x}}$ or $\frac{d\mathbf{x}}{dt}$. A derivative calculated w. r. t. an arbitrary noninertial frame F_1 (or vector basis ${}_1\mathbf{e}$) is called *relative* [115], *apparent* [41], or *local* [49] *time derivative*. It is further referred to as relative derivative and is denoted by $\frac{{}^1d\mathbf{x}}{dt}$. If the vector basis is obvious from the context the circle derivative $\overset{\circ}{\mathbf{x}}$ is used. A vector's absolute and relative derivative w. r. t. a frame F_1 are related by

$$\frac{d\mathbf{x}}{dt} = \frac{{}^1d\mathbf{x}}{dt} + \boldsymbol{\omega}_{I,1} \times \mathbf{x}, \quad (2.3)$$

where $\boldsymbol{\omega}_{I,1}$ is the absolute angular velocity of frame F_1 . The relative time derivative can be viewed as derivation w. r. t. a moving but non-rotating frame. When deriving dynamics equations it is more convenient to choose a reference point other than O_I . For a general reference point O_0 and using $\mathbf{r} := \mathbf{r}_{I,c}$ and $\mathbf{s} := \mathbf{r}_{I,0}$ the Equations 2.1 and 2.2 become

$$\mathbf{F} = m\ddot{\mathbf{r}}, \quad (2.4)$$

$$\frac{d{}_0\mathbf{L}}{dt} = {}_0\boldsymbol{\tau} - m(\mathbf{r} - \mathbf{s}) \times \ddot{\mathbf{s}}. \quad (2.5)$$

The most important and compact choice is to use the center of mass as reference point $O_0 \equiv O_c$ which implies $(\mathbf{r} - \mathbf{s}) = 0$ and using the well-known relationship between translational momentum and mass

³⁾ In this work *action* [115] is used to denote the concepts of force and torque.

and angular momentum and inertia tensor [149] to obtain the linear and angular momentum equations

$${}_I\mathbf{P} = m\mathbf{v}, \quad (2.6)$$

$${}_c\mathbf{L} = {}_c\mathbf{J}\boldsymbol{\omega} + {}_c\mathbf{l}, \quad (2.7)$$

where $\boldsymbol{\omega} := \boldsymbol{\omega}_{I,c}$ is the absolute angular velocity of the body and $\mathbf{v} = \dot{\mathbf{r}}$ the absolute velocity of the center of mass. After differentiating (2.6) and (2.7) w. r. t. time and using that $\frac{{}_c d\boldsymbol{\omega}}{dt} = \frac{d\boldsymbol{\omega}}{dt}$ the dynamical equations in terms of velocities are

$$\mathbf{F} = m \frac{d\mathbf{v}}{dt}, \quad (2.8)$$

$${}_c\boldsymbol{\tau} = {}_c\mathbf{J} \frac{d\boldsymbol{\omega}}{dt} + \boldsymbol{\omega} \times ({}_c\mathbf{J}\boldsymbol{\omega} + {}_c\mathbf{l}) + \frac{{}_c d{}_c\mathbf{l}}{dt}. \quad (2.9)$$

So far all expressions shown were based on pure coordinate free tensorial notation. A matrix representation is obtained if tensors (arbitrary vector \mathbf{v} and dyadic \mathbf{D}) are resolved w. r. t. an orthonormal vector basis ${}_x\mathbf{e} := ({}_x\mathbf{e}_1, {}_x\mathbf{e}_2, {}_x\mathbf{e}_3)^T$

$$\begin{aligned} \mathbf{v} &= {}_x\mathbf{e}^T {}_x v = {}_x v^T {}_x\mathbf{e} \\ \mathbf{D} &= {}_x\mathbf{e}^T D {}_x\mathbf{e} \end{aligned}$$

where ${}_x\mathbf{e}_{\alpha x} \mathbf{e}_{\beta} = \delta_{\alpha\beta}$ are the simple Cartesian metric coefficients and hence ${}_x\mathbf{e}^T {}_x\mathbf{e} = I_{3 \times 3}$ is just the unit matrix. The orientation of the center of mass frame is given by the direction cosine matrix ${}_c\mathbf{e} = {}_c R_I(t) {}_I\mathbf{e}$.

2.1.2 Coordinate representations

Symbolic manipulation and reformulation of the physical laws (2.1) and (2.2) is possible in tensorial form. A matrix representation of the dynamics equations is optional for symbolic treatment, but mandatory for numerical evaluation and implementation on a digital computer. Three key requisites determine the ways of calculating the dynamics:

- reference point \mathbf{O}_0 used for calculating the dynamics in (2.4) and (2.5)
- vector bases required for resolving vectors and dyadics
- absolute or relative derivatives

The notion of a *coordinate representation* denotes one certain choice of each of the three categories. Some prominent used in robotics are *inertial*, *body*, and *absolute* representation [49]. In order to indicate that a matrix belongs to a certain representation R a left superscript ${}^{(R)}X$ is used. The symbol R denotes one particular representation, in this work three are used, namely $R \in \{I, B, A\}$.

The goal of this section is to emphasize the existence and importance of coordinate representations used to formulate dynamics algorithms. The design of a dynamics software architecture must consider coordinate representations to enable consistent code generation, code reuse, and interpretation and exchange of computational results during run-time. The prominent representations mentioned are reviewed shortly below, to introduce expressions which form the basis of a family of multibody formalisms. Though describing the same underlying physics, different representations result in different spatial operators, time derivatives, dynamical equations, and, finally, in different matrices, since the operators will describe quantities at different points in space and w. r. t. to different vector bases. Each approach has distinct advantages and

properties concerning symbolic und numerical complexity, computational efficiency, and numerical stability as discussed later in Section 3.1 and 3.2. Studies comparing the influence of coordinate representation on computational efficiency of certain classes of recursive methods can be found in [87, 40, 136, 115]. Unfortunately there exists no uniform notation and choice of coordinate representation in robotics literature to describe mechanical quantities, so this section has to (re-)state clear definitions here.

2.1.2.1 Inertial coordinate representation of dynamics

In *inertial representation* the dynamics is calculated (i) w. r. t. a location \mathbf{O}_x fixed in the body, (ii) with all tensors resolved w. r. t. inertial system ${}_I e^T$, and (iii) using absolute time derivatives. The advantage of this traditional formulation of the dynamics is that all quantities have a direct physical meaning. For compact symbolic expressions we adopt the *spatial notation* (not to be confused with *spatial representation* of Featherstone [40]) similar to that introduced by Rodriguez et al. [117] where matrices $\in \mathbb{R}^6$ (two matrices $\in \mathbb{R}^3$ stacked, called *spatial vector*) and $\in \mathbb{R}^{6 \times 6}$ are used. The most prominent are defined in Table 2.1 on the current page. All tensors are resolved w. r. t. the inertial frame F_I , the absolute translational velocity is defined by ${}^I v_{I,c} := \frac{d}{dt} {}^I r_{I,c}$.

TABLE 2.1: Prominent spatial matrix operators in inertial coordinate representation. Here the center of mass \mathbf{O}_c is used as reference point.

spatial velocity	$\langle^I V_c := \begin{pmatrix} {}^I \omega_{I,c} \\ {}^I v_{I,c} \end{pmatrix}$
spatial angular velocity	$\langle^I \Omega_c := \begin{pmatrix} {}^I \omega_{I,c} \\ \mathbf{0}_3 \end{pmatrix}$
spatial force	$\langle^I f_c := \begin{pmatrix} {}^I \tau \\ {}^I F \end{pmatrix}$
spatial inertia	$\langle^I M_c := \begin{bmatrix} {}^I J & \mathbf{0}_{3 \times 3} \\ \mathbf{0}_{3 \times 3} & m I_{3 \times 3} \end{bmatrix}$
spatial momentum	$\langle^I \Pi_c := \begin{pmatrix} {}^I L \\ {}^I p \end{pmatrix}$
internal spatial momentum	$\langle^I \Pi_{ic} := \begin{pmatrix} {}^I l \\ 0 \end{pmatrix}$

The momentum equations (2.6) and (2.7) can be written compactly

$$\langle^I \Pi_c = \langle^I M_c \langle^I V_c + \langle^I \Pi_{ic}. \quad (2.10)$$

The dynamical equations (2.8) and (2.9) w. r. t. F_c are as follows

$$\langle^I f_c = \langle^I M_c \langle^I \dot{V}_c + \left(\langle^I \tilde{\Omega}_c \langle^I M_c - \langle^I M_c \langle^I \tilde{\Omega}_c \right) \langle^I V_c + \langle^I \dot{\Pi}_{ic}. \quad (2.11)$$

Proof: Taking the time derivative of (2.10) leads to

$$\langle^I \dot{\Pi}_c = \langle^I M_c \langle^I \dot{V}_c + \langle^I \dot{M}_c \langle^I V_c + \langle^I \dot{\Pi}_{ic}.$$

Comparing this expression to (2.9) shows that

$$\langle^I \dot{M}_c \langle^I V_c = \begin{pmatrix} {}^I \omega \times {}^I J^I \omega \\ \mathbf{0}_3 \end{pmatrix} = \begin{pmatrix} {}^I \tilde{\omega} {}^I J^I \omega - {}^I J^I \tilde{\omega} {}^I \omega \\ {}^I \tilde{\omega} m I_{3 \times 3} - m I_{3 \times 3} {}^I \tilde{\omega} \end{pmatrix}.$$

Here the *tilde operator*⁴⁾ has been introduced to express the vector product by pre-multiplication of an anti-symmetric matrix $\tilde{p} \in \mathbb{R}^{3 \times 3}$ such that $p \times q = \tilde{p} \cdot q$, $p, q \in \mathbb{R}^3$

$$\tilde{p} := \begin{bmatrix} 0 & -p_z & p_y \\ p_z & 0 & -p_x \\ -p_y & p_x & 0 \end{bmatrix}. \quad (2.12)$$

The six-dimensional complement of (2.12) which operates on spatial vectors is called *spatial cross product* [66] and defined by

$$X \times Y = \tilde{X}Y = \begin{pmatrix} \tilde{a}c \\ \tilde{b}c + \tilde{a}d \end{pmatrix} \text{ where} \quad (2.13)$$

$$\tilde{X} := \begin{bmatrix} \tilde{a} & 0_{3 \times 3} \\ \tilde{b} & \tilde{a} \end{bmatrix} \text{ with } X := \begin{pmatrix} a \\ b \end{pmatrix}, \text{ and } Y := \begin{pmatrix} c \\ d \end{pmatrix}. \quad (2.14)$$

Using ${}^{(I)}\tilde{\Omega}_c = \text{diag}({}^{(I)}\tilde{\omega}, {}^{(I)}\tilde{\omega})$ and observing ${}^{(I)}\dot{M}_c {}^{(I)}V_c = ({}^{(I)}\tilde{\Omega}_c {}^{(I)}M_c - {}^{(I)}M_c {}^{(I)}\tilde{\Omega}_c) {}^{(I)}V_c$ leads to the result. \square

The two kinds of tilde operators satisfy a number of useful identities which are listed in Appendix B. The dynamics of an arbitrary body-fixed location \mathbf{O}_x is as follows:

$${}^{(I)}f_x = {}^{(I)}M_x {}^{(I)}\dot{V}_x + \left({}^{(I)}M_x {}^{(I)}\tilde{\Omega}_x + {}^{(I)}\tilde{V}_x {}^{(I)}M_x \right) {}^{(I)}V_x + {}^{(I)}\phi_{c,x}^\top {}^{(I)}\dot{\Pi}_{ic}. \quad (2.15)$$

Proof: The transformations for velocity and torque when changing from body-fixed reference point \mathbf{O}_y to \mathbf{O}_x , where $\mathbf{l} := \mathbf{r}_{y,x}$ is the vector from \mathbf{O}_y to \mathbf{O}_x , are [115]

$${}^I\boldsymbol{\omega}_x = {}^I\boldsymbol{\omega}_y \quad \mathbf{v}_{I,x} = \mathbf{v}_{I,y} - \mathbf{l} \times {}^I\boldsymbol{\omega}_y \quad (2.16)$$

$${}_x\boldsymbol{\tau} = {}_y\boldsymbol{\tau} - \mathbf{l} \times {}_x\mathbf{F} \quad {}_x\mathbf{F} = {}_y\mathbf{F} \quad (2.17)$$

These equations reflect that angular velocity is the same in the whole rigid body and that the body accelerates identically as a particle of equivalent mass. These fundamental transformation properties can be expressed by introducing the *rigid body transformation operator*

$${}^{(I)}\phi_{x,y} := \begin{bmatrix} I_{3 \times 3} & 0_{3 \times 3} \\ -{}^I\tilde{r}_{y,x} & I_{3 \times 3} \end{bmatrix} \quad (2.18)$$

which transforms spatial velocities from F_y to F_x and, when transposed, spatial forces from F_x to F_y , thus writing

$${}^{(I)}V_x = {}^{(I)}\phi_{x,y} {}^{(I)}V_y \quad {}^{(I)}f_y = {}^{(I)}\phi_{x,y}^\top {}^{(I)}f_x. \quad (2.19)$$

Because \mathbf{l} is constant w. r. t. the body it follows from (2.3) $\frac{d\mathbf{l}}{dt} = {}^I\boldsymbol{\omega} \times \mathbf{l}$. Using (B.8) leads to the important operator derivative

$${}^{(I)}\dot{\phi}_{x,y} = {}^{(I)}\tilde{\Omega}_y {}^{(I)}\phi_{x,y} - {}^{(I)}\phi_{x,y} {}^{(I)}\tilde{\Omega}_y. \quad (2.20)$$

The spatial inertia transforms similarly to the spatial force

$${}^{(I)}M_x := {}^{(I)}\phi_{c,x}^\top {}^{(I)}M_c {}^{(I)}\phi_{c,x} = \begin{bmatrix} {}^I J - m^I \tilde{l} \tilde{l} & m^I \tilde{l} \\ -m^I \tilde{l} & m I_{3 \times 3} \end{bmatrix}$$

⁴⁾ The vector product $\mathbf{r} = \mathbf{p} \times \mathbf{q}$ can be viewed as the double contraction of the total antisymmetric pseudo-tensor ε_{ijk} (Levi-Civita symbol) by $r_i = \sum_{jk} \varepsilon_{ijk} p_j q_k$ so the operator \tilde{p} becomes a single contraction of $\varepsilon_{ijk} p_j$ w.r.t. to second index j .

where its total time derivative, using (2.20), writes

$$\langle I \rangle \dot{M}_x := \langle I \rangle \tilde{\Omega}_x \langle I \rangle M_x - \langle I \rangle M_x \langle I \rangle \tilde{\Omega}_x.$$

Omitting $\langle I \rangle$ superscripts for brevity, the last required operator identity is

$$-\tilde{V}_x M_x V_x = \phi_{c,x}^\top \tilde{\Omega}_x \phi_{c,x}^{-\top} M_x V_x \quad (2.21)$$

which can be shown by multiplication of the matrices and using (B.8) and (B.4). Using $\Omega_c = \Omega_x$ and $\langle I \rangle \phi_{c,x} = \langle I \rangle \phi_{x,c}^{-1}$ one writes

$$\begin{aligned} f_x &= \phi_{c,x}^\top f_c \\ &= \phi_{c,x}^\top \left(M_c \dot{V}_c + \dot{M}_c V_c + \dot{\Pi}_{ic} \right) \\ &\stackrel{(2.20)}{=} \phi_{c,x}^\top \left(M_c \left[\phi_{c,x} \dot{V}_x + \left\{ \tilde{\Omega}_x \phi_{c,x} - \phi_{c,x} \tilde{\Omega}_x \right\} V_x \right] + \left[\tilde{\Omega}_x M_c - M_c \tilde{\Omega}_x \right] V_c + \dot{\Pi}_{ic} \right) \\ &= M_x \dot{V}_x + \phi_{c,x}^\top M_c \tilde{\Omega}_x V_c - M_x \tilde{\Omega}_x V_x + \phi_{c,x}^\top \tilde{\Omega}_x M_c V_c - \phi_{c,x}^\top M_c \tilde{\Omega}_x V_c + \phi_{c,x}^\top \dot{\Pi}_{ic} \\ &= M_x \dot{V}_x - M_x \tilde{\Omega}_x V_x + \phi_{c,x}^\top \tilde{\Omega}_x \phi_{c,x}^{-\top} M_x V_x + \phi_{c,x}^\top \dot{\Pi}_{ic} \\ &\stackrel{(2.21)}{=} M_x \dot{V}_x - \left(M_x \tilde{\Omega}_x + \tilde{V}_x M_x \right) V_x + \phi_{c,x}^\top \dot{\Pi}_{ic} \quad \square \end{aligned}$$

(2.15) is one of the key results of this section. It is the starting point when deriving other representations, some discussed below, and serves as a basis when treating multiple rigid bodies.

2.1.2.2 Body representation

A more concise notation of the dynamical equations is the *body representation* [149, 109, 49, 113]. Dynamics is calculated (i) in a body-fixed frame F_x , (ii) where tensors are represented w. r. t. to a body-fixed frame, usually the same frame F_x , (iii) calculating time derivatives in terms of body-local (relative) derivatives. Matrices belonging to this representation are denoted by a left superscript $\langle B \rangle$. In case of a single gyrostator the transition from inertial to body representation is merely done by considering possibly different vector bases when transforming between two body-fixed locations F_x and F_y and expressing absolute time derivatives by (body) relative derivatives [2]. The rigid body transformation operator (2.18) now writes

$$\langle B \rangle \phi_{x,y} := \begin{bmatrix} {}^x R_y & 0_{3 \times 3} \\ -{}^x R_y \tilde{l} & {}^x R_y \end{bmatrix}, \quad (2.22)$$

where again $\mathbf{l} := \mathbf{r}_{y,x}$. With help of the obvious spatial generalization of (2.3), i. e.,

$$\frac{dV}{dt} = \overset{\circ}{V} + \tilde{\Omega}_x V \quad (2.23)$$

the dynamics w. r. t. F_x in body coordinates is

$$\langle B \rangle f_x = \langle B \rangle M_x \langle B \rangle \overset{\circ}{V}_x - \langle B \rangle \tilde{V}_x^\top \langle B \rangle M_x \langle B \rangle V_x + \langle B \rangle \phi_{c,x}^\top \langle I \rangle \dot{\Pi}_{ic}. \quad (2.24)$$

In further symbolic manipulation it might be useful to decompose the rigid body transformation operator in parts for *pure* rotation and pure displacement:

$$\phi_R(R) := \begin{bmatrix} R & 0_{3 \times 3} \\ 0_{3 \times 3} & R \end{bmatrix} \quad \text{and} \quad \phi_D(l) := \begin{bmatrix} I_{3 \times 3} & 0_{3 \times 3} \\ -\tilde{l} & I_{3 \times 3} \end{bmatrix}$$

so (2.22) goes over into $\langle^B\rangle\phi_{x,y} = \phi_R(xR_y)\phi_D(yI)$ and (2.18) into $\langle^I\rangle\phi_{x,y} = \phi_D(II)$.

The strong relation between Lie groups and this body representation of dynamics has been pointed out by Park et al. [109]. Following geometrical arguments in the book of Murray et al. [98], one can show that the rigid body transformation operator is a matrix representation of the adjoint operator Ad on $SE(3)$ and the spatial cross product operator definition in (2.13) is a representation of the ad operator on the corresponding Lie algebra $se(3)$.

2.1.2.3 Absolute representation of dynamics

The main idea of this representation is to calculate the dynamics at a point fixed in the inertial frame, so that \ddot{s} becomes zero in (2.5). This is in contrast to inertial and body representation where a point attached to the rigid body was used as reference point. Screw-theory based multibody dynamics was elaborated in the book of Featherstone [40] and related to body representation and Lie group interpretations by Murray et al. [98].

The absolute representation is characterized by (i) using \mathbf{O}_I , the origin of the inertial frame F_I as reference point for dynamics, (ii) applying the same frame to resolve all tensors, and (iii) employing total time derivatives for spatial operators. A spatial operator in absolute representation is denoted by a superscript $\langle^A\rangle$. It can be obtained from the inertial one using the rigid body transformation operator (2.18). The spatial velocity then is

$$\langle^A\rangle V = \langle^I\rangle\phi_{I,x}\langle^I\rangle V_x. \quad (2.25)$$

The time derivative of a spatial vector in absolute representation is

$$\langle^A\rangle\dot{X} = \langle^A\rangle\overset{\circ}{X} + \langle^A\rangle\tilde{V}\langle^A\rangle X \quad (2.26)$$

which implicates $\langle^A\rangle\dot{V} = \langle^A\rangle\overset{\circ}{V}$. The spatial inertia in absolute representation is the spatial inertia w. r. t. the origin of the inertial frame \mathbf{O}_I

$$\langle^A\rangle M := \langle^I\rangle\phi_{x,I}^T \langle^I\rangle M_x \langle^I\rangle\phi_{x,I}$$

and exploiting $\langle^I\rangle\dot{\phi}_{x,I} = (\langle^I\rangle\tilde{\Omega}_x - \langle^I\rangle\tilde{V}_x + \langle^I\rangle\tilde{\Omega}_x \langle^I\rangle\dot{\phi}_{x,I} - \langle^I\rangle\dot{\phi}_{x,I} \langle^I\rangle\tilde{\Omega}_x)$ gives the time derivative of the spatial inertia

$$\langle^A\rangle\dot{M} = -\langle^A\rangle\tilde{V}^T \langle^A\rangle M - \langle^A\rangle M \langle^A\rangle\tilde{V}.$$

The net force acting on the body is the time derivative of the spatial momentum about the origin of F_I which leads to the dynamics in absolute representation

$$\langle^A\rangle f = \langle^A\rangle M \langle^A\rangle\dot{V} - \langle^A\rangle\tilde{V}^T \langle^A\rangle M \langle^A\rangle V + \phi_{c,I}^T \langle^I\rangle\dot{\Pi}_{ic}. \quad (2.27)$$

One appealing property of the absolute representation is that quantities belonging to different bodies and locations can be related immediately without applying transformations such as (2.22). As shown later this leads to an enormous simplification of symbolic expressions describing multiple bodies. On the other hand the physical interpretation of quantities in absolute representation is not as intuitive as in the inertial one.

The only mechanical 'device' discussed so far was the single gyrostat. Its dynamics in three-dimensional Euclidean space is influenced by the gyrostat's internal characteristic properties (i) mass, (ii) moments of inertia, and (iii) internal angular momentum. A robot consists of various interacting components with different behaviour and properties. The following section attempts to capture the description and behaviour of such multibody systems in a conceptual and a mathematical way – both prerequisites for a computational robot model.

2.2 Paradigm for abstract robot model specification

This section shows a generalized way of *specifying* the robot system's mechanical *properties*. This is motivated by the idea that specifying the properties has a counterpart in the process of *establishing* the system's *equations*, to guide generation and organization of code on a later stage. As a consequence this section does not follow the classical introduction of multibody formalisms and starts from more general considerations.

In robotics the starting point for formulating the equations of motion often are mechanical systems composed of *links*. The notion of a link is very useful when dealing with kinematic chains where a number of material bodies are connected sequentially by holonomic joints. In robotics literature a link represents a body and two locations fixed in the body where adjacent bodies are supposed to be connected by a joint. There are two reasons why often coordinate frames are associated with joint locations. On the one hand the convenient and popular convention according to Denavit and Hartenberg [34] describes how frames can be located in the joints. On the other hand many multibody algorithms can be efficiently formulated in joint reference frames [40]. When taking a closer look there remain some open questions even for the *specification* of this simple case of chain-structured mechanisms: Is the robot's base itself a link? Does an endeffector or a payload need two joint locations? How should the bodies be indexed?

For simplicity most researchers treat robotic mechanisms uniformly as joint-connected bodies, more precisely pairs of one adjacent joint and one body. They assign special meaning to, e. g., link 0 which is treated as the reference system, or introduce specific and context-depending rules for model interpretation, or zero-mass or zero-length links. This is very convenient and is an appropriate representation if one stays within one single formalism. But relying on different paradigms and implicit assumptions is awkward for formalism transfer, efficient code generation, and coupling between several algorithms. One example are the algorithms presented in [67], where the crucial information which coordinate representation is used is implicitly contained in the mathematical formalism. Some situations limiting flexibility and applicability of the existing formalisms, such as SOA formulation, to more general multibody systems are:

- there are more or less than two frames associated with bodies
- the need for MBS topologies such as tree-structures and loops
- presence of general entities and physical effects in the system such as internal and external forces, internal angular momentum, or artificial muscles in biomechanics
- environmental conditions, such as cooperating robots, inhomogeneous gravitational fields, or buoyancy

In order to obtain a sufficiently general system model this section tries to answer the questions:

- What are the major components of the mechanical robot model?
- What is its static and dynamic behaviour?

A proven method to achieve the required flexibility for a modeling language is to model the language itself by an abstract higher-level formalism. Tools that support that kind of meta-formalism are DOME [55] or ATOM³ by Vangheluwe and de Lara [143]. Having an object-oriented implementation at a later stage in mind the following section starts from an abstract level. We generalize from a purely data-driven to an

abstract model specification by *formally* capturing the structure of a robot model to handle more complex systems and maintain extensibility. The shown approach is similar to the PSPEC approach proposed by Hatley and Pirbhai [51] for real-time system specification which contains all the information necessary for the designer to know *what* to do without saying *how* to do it.

The entity-relationship (ER) model introduced by Chen [26] is applied to formulate the parts of a mechanical robot model in a component-oriented and port-based way. This concept of a component is denoted by *MBS entity*. Components and contexts known from robot and multibody system dynamics are analyzed w. r. t. to the MBS entity paradigm to capture and classify the behaviour and all concerned relevant information and relations. The advantage of this approach is an abstract high-level view on the properties of the components and relations between the parts revealing *explicitly* as much information as possible without doing, e. g., a simulation run. By this the following ameliorations are attained:

- Efficient code generation for various algorithms, e. g., according to a dataflow model of computation, may be based on different assumptions about the model. These can be expressed explicitly and are not entangled within code or a mathematical formalism.
- Enhancing the potential for optimization of the given model description according to different criteria.
- Dynamics computations might not be the only software task to be driven by a robot specification in a robot control system, so the reduction to just links limits the applicability of the model. Integration of components, implemented by persons with different objective and background, is enabled because created objects are based on the same reference model.
- Checking for model correctness by formal methods.
- Clear textual or graphical representations enabling persistence of model description and system state.
- When formulating mathematical equations by means of spatial operators introduced in Section 2.3 and defining topology and causality of the multibody model graph the ER paradigm is beneficial. The goal is to preserve the advantageous algebraic properties and explicitness of spatial operators expressing physical properties directly.

2.2.1 Component specification model

Multibody systems such as robots naturally offer a component-oriented view for two reasons. On the one hand real mechanisms consist of distinct rigid bodies, joints, force-element, drives and other devices. Problems from biomechanics also allow such a subdivision into bones, muscles, tendons. On the other hand the mathematical equations describing these systems show a similar structure which is exploited by dynamics formalisms, which will be shown in Section 3.1. As discussed above researchers often reduce this set of components to one member, the link, e. g., see [60], or restrict their properties for specification of the robot in order to formulate, e. g., dynamics algorithms for that class of components. This procedure is completely legit, but one might run into problems to drive several algorithms by one single specification, because assumptions about the behaviour of components were made implicitly. Basing the components' description on an abstract representation which captures formally all properties and behaviour and their relations is the solution proposed in this section.⁵⁾

⁵⁾ This section is an advanced version of parts of the paper [56].

The *entity-relationship* (ER) model [26] offers in a natural way an abstract view on systems primarily formed by components. The concept is based on the general idea to represent the data in two logical parts, entities and relations. An entity e is defined as a 'thing' which can be distinctly identified, a relationship r is an association between two entities. Both can be augmented by *attributes*, which may represent arbitrary data. Entities are classified into different entity sets E_i . There exists a predicate associated with each entity set to test whether an entity e_i belongs to it. A relationship set R_i is a mathematical relation among n entities, each taken from an entity set $\{[e_1, \dots, e_n] | e_i \in E_i\}$ and each tuple $[e_1, \dots, e_n]$ is a relationship. A graphical representation for ER models is offered through *ER diagrams*. In this work the relations between concrete realizations of MBS entities, objects, are graphically represented by Unified Modeling Language (UML) [100, 101, 127] class diagrams, where an entity is represented by a UML class symbol and a relation by a UML association symbol.

The UML is a mainly graphical notation or syntax that object-oriented methods use to express software designs. The portions of the UML applied in this work relies on the presentation in [43]. Figure 2.2 shows the graphical constructs used in this section. ER diagrams are expressed here using *UML class diagrams* which describe the types of entities in a model and the various static relationships that exist between them. The two principal kinds of static relationships between *classes* are *association* and *subtype*.

A *class* is depicted by a rectangular box. The association represents a conceptual relationship classes. It has two ends, which can be explicitly named by a label called *role name*. It has a *multiplicity*, which is an indication of how many objects may participate on each end in the given relationship. Multiplicity can be a whole number or a range denoted by $x..y$, $x, y \in \mathbb{N}$, and $*$ denotes an infinite number. On a less conceptual level a class may have a number of *attributes* and *operations* listed in separate blocks inside the box. From a conceptual perspective attributes are like associations, the difference shows up in the implementation perspective not considered here. Operations are the processes that a class knows to carry out. The *composition* expresses that a part belongs one whole, the class where the line ends in a black diamond. The *dependency* among components show how changes to one component may cause the depending components to change. The *generalization* of Supertype to Subtype states that Subtype1 is a *subtype* of Supertype, i. e., an extension where attributes of the supertype are replicated and new ones can be added and operations of the supertype may be either replicated or overridden by new operations.

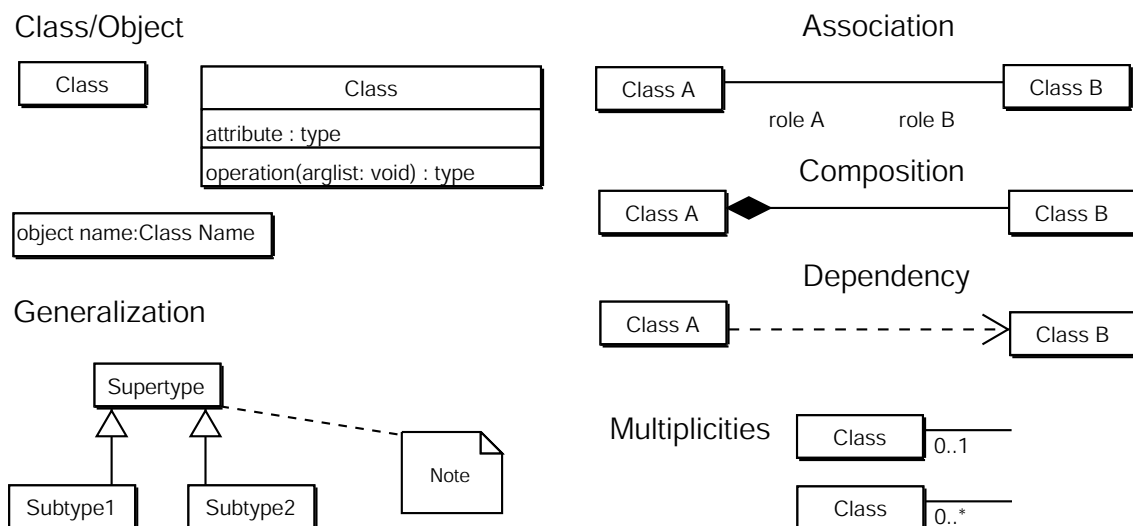


FIGURE 2.2: Groups of graphical constructs used in UML class diagrams, required to express Entity-Relationship (ER) diagrams. For the interpretation of the constructs please see text. Right side: A is associated to B, B is composed of A, B depends on A.

The first step in the design of an ER model of robotic mechanisms is to identify the entity sets and rela-

tionship sets which are necessary. The analysis performed in Section 2.1.1 suggests to base the component model, visualized in Figure 2.3, on the following entity sets which graphically are expressed by the UML symbol for a Class:

- MBS entity (ME)
- Interaction port (IP)
- Physical state (PS)
- Physical property (PP)
- Coordinate Frame (CF)
- Constraint Relations (CR)

The components from the multibody and robotics domain are subsumed under the entity set *MBS entity* (ME). The robot and parts of its environment are restricted to be constructed from a finite set of primitive domain components that is sufficient to cover a large class of robots, either representing technical devices or physical phenomena:

- joints (revolute, prismatic, etc.)
- rigid links and bodies
- spring, dampers, and external forces
- drivetrains
- mechanical unilateral contact
- custom-specific or more abstract parts

An analysis based on the ER model for a selected number of fundamental components including their substructure is given in the following sections. The MBS entities and attributes are indicated by a different typesetting, using sans serif.

MBS entities are supposed to interact via distinct interaction units called *interaction ports* (IP). The kind of interaction is not specified on this abstract conceptual level. Concrete examples are mechanical connections, e. g., fixed, unilateral contact, or interaction that may takes place on a pure logical level. Context dependent port semantics will be applied to determine the interpretation of a port.

In mechanics a *body* denotes an entity carrying *mass* and mostly having a geometrical extension. So each ME is related to some physical property, such as mass, angular momentum, etc., which are captured by the entity set *physical property* (PP). PPs are time-independent in a physical sense which distinguishes them from *physical state* (PS) attributes. These represent state information, e. g., joint angles, forces, etc.

As shown in Section 2.1.1 in multibody dynamics most physical phenomena such as force and torque, are of tensorial nature. Tensors often are inherently related to reference frames and require coordinate frames to be resolved in matrix form. Though coordinate frames are an idea of geometric nature, these are of such fundamental interest in MBS dynamics to motivate the entity set of *coordinate frames* (CF).

The substructure of an ME represented by the attributive entity sets IP, PP, PS, and CF, is related by *composition* relationships. This idea from object-oriented modeling can express well that, e. g., the idea of

a mass value without any related body is of limited physical sense in a robot model. Optionally, the PP can depend on each other. The *constraint* relationship set models this dependence which might be (i) a mutual dependency, e. g., mass and moments of inertia can be calculated from geometrical shape and mass density of a body (see footnote on page 9), (ii) parameter values may restrict the range of other parameters in order to get a physically meaningful model, e. g., requiring the matrix of moments of inertia being positive definite. Both relations may be expressed for instance by symbolic expressions. Some PP such as inertia matrix require the existence of a reference point or frame to be meaningful. This *reference* relationship set expresses that a PP has a discrete number of reference frames. In a numerical computation the components of vectors and tensors must be resolved w. r. t. a vector basis (see 2.1.2) to obtain numerical values. The relationship set *representation* states which vector bases are used to represent the matrix entries. More generally speaking, this allows for *interpretation* of the model specification but not determines physical reality. IP entities are supposed to represent the points where ME entities can interact. When dealing with mechanical interaction, inevitably a location in space is required to define where the interaction takes place. In numerical schemes a complete coordinate frame CF is required to resolve vectors. The *association* between a CF and IP entity manifests this. Note that not each CF has to be associated to an IP because reference frames might not be 'exposed' to the outside of a component and may remain a conceptual idea required on a logical level. The relationship set defined on the single entity set ME is the *component* relation. This relation models the hierarchical decomposition of entities into parts. If cardinality is greater than zero such an ME is referred to as *MBS assembly*. A component relation might induce some additional constraints on the structure of an ME which are discussed below.

In this *conceptual* view, it is not mandatory to apply another feature of the ER model, *attributes* and associated *value sets* representing any kind of data belonging to entities and relationships. Concrete examples forming the basic classes for a robotics domain component library are given in the next section where entities of the multibody domain are modelled using Figure 2.3. In the first step properties are emphasized, resulting in an abstract data model, and in the second step emphasis is on behaviour (transformation model).

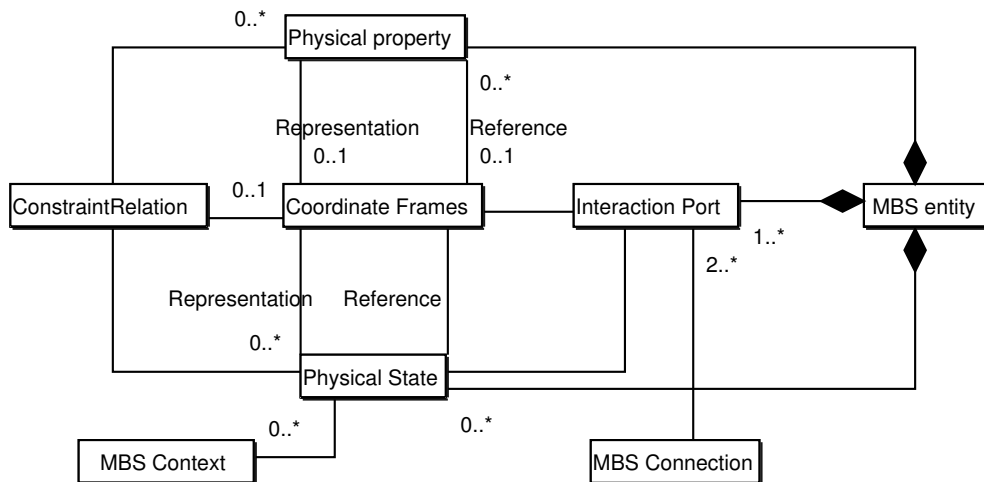


FIGURE 2.3: Conceptual Entity-Relationship model of an MBS entity and its constituents in graphical UML notation.

2.2.1.1 RigidBody entity

In mechanics a *body* denotes an entity carrying *mass*, having a geometrical extension, and locations where some actions can be applied. In the domain of multibody system dynamics one can take the attributes mass

m and inertia \mathbf{J} for granted. The nature of a body is the ability to couple to a gravitational field and to react to external action by inertial forces as discussed in 2.1.1. The interesting thing is, that even this well-accepted view is implicitly based on modeling assumptions. Considering the gravitational (volume) forces these properties are only sufficient to some extent to calculate the dynamics correctly. Only for homogeneous fields the center of mass, which is the point upon the net inertial forces act, is identical to the center of gravity, onto which the net gravitational volume force acts. So without further semantical information stemming from some kind of *context* and some additional physical information about the mass distribution of the body this model is valid for homogeneous fields.⁶⁾

Modeling these concepts within the MBS entity paradigm yields Figure 2.4. The most general model `RigidBody` contains physical property attributes denoted by `PP:mass` and `PP:inertia`. The latter requires a reference point and a vector base represented by `CF:Rep` and `CF:Rep`. There are a number of N_F frames `CF` for designating locations attached to the body which are exposed to the same number of interaction ports `IP:x`. The constraint relation between the frames `CF:Ref` and `CF:x` might be implicit or explicit. If one identifies `CF:Ref` with the center of mass system this is a MBS entity model of the body depicted in Figure 2.1 for $N_F = 2$.

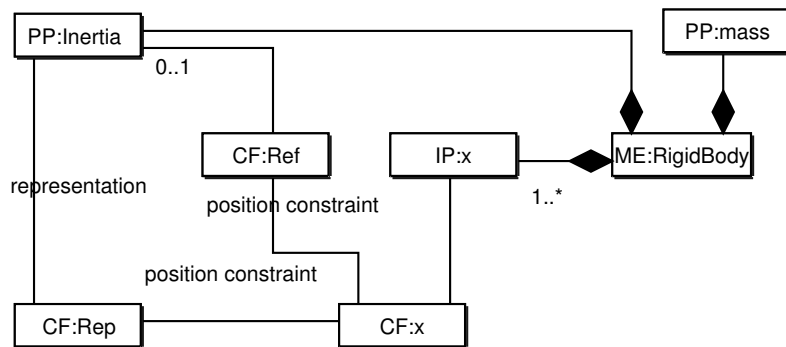


FIGURE 2.4: ER model of the MBS entity `RigidBody`.

The most concise specialization of this model is MBS entity `RigidBody<1,Type1>`, which is shown in Figure 2.5. It represents a body with one interaction port called `IP:1` and one frame $F_{CF:1}$ represented by `CF:1` associated with it. This component can only act as a leaf in a multibody graph structure. The reference system of the inertia `PP:l` is the center of mass frame `CF:CM` located in the center of mass and equivalent to $F_{CF:1}$. The vector basis used to represent the entries of \mathbf{J} is also $\mathbf{1}_e$. The specialization tag `<1,Type1>` denotes the number of frames $N_F = 1$ and `Type1` the equivalence of the center of mass frame and $F_{CF:1}$.

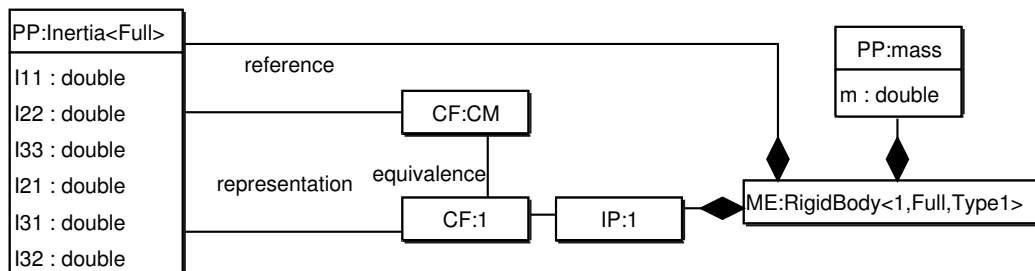


FIGURE 2.5: ER model of the specialized MBS entity `RigidBody<1,Full,Type1>`.

⁶⁾ Hence, this assumption holds for all ground-based robots and technical processes, but not for geostationary satellites of large extent.

From Figure 2.4 the mathematical operators as discussed in Section 2.1.1 can be extracted. From the viewpoint of a mathematical formalism using spatial notation the proposed rigid body provides a spatial inertia matrix M_{CF} w. r. t. a frame F_{CF} and rigid body transformations $\phi_{CF:x,CF:y}$ between various frames where $x \neq y, x, y \in \{1, \dots, N_F\}$, where the set $\{F_{CF:1}, \dots\}$ must be exposed by the interaction ports. This is one kind of behavioural description of a rigid body.

To provide more modeling flexibility it should be possible to derive models which provide a different view [38] on the body data, e. g., to supply information about the mass density and geometrical data. The number of PP will increase, and due to the redundancy some constraints for deriving the body data has to be provided. The only difference is in the data model. It must be emphasized, however, that such a component must have the same *behaviour* as a rigid body otherwise it would represent a different entity. To capture this fact by the modeling paradigm is exactly the reason to take this abstract point of view. It should be noted that the model chosen above is motivated by separating the characteristics of the rigid body. A more user-friendly specialization of the RigidBody entity should provide attachment points different from the center of mass.

2.2.1.2 Displacement entity

Describing one reference location O_1 on a rigid body by means of a second one O_2 on this body, often required in robotics, implies a time-invariant shift between two locations by a vector $r_{1,2}$. In mechanics terms this behaviour shifts an attachment or reference point or the point of application of actions to a different location. Introducing two coordinate frames F_1 and F_2 there also is the possibility to change the local vector basis ${}_2e = {}^2R_1e$ which does not affect the physical behaviour of shifting the reference point. This behaviour offers several interpretations depending on the modeling concepts used ranging from massless rigid rod or rigid body to homogeneous transformation or a link with zero degree of freedom. Mathematically spoken, a displacement entity puts a positional constraint between two coordinate frames which depends on time-invariant local positional parameters. Figure 2.6 shows an ME diagram of that dependencies.

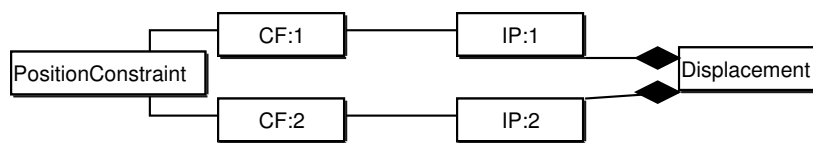


FIGURE 2.6: ER model of the MBS entity Displacement.

The positional parameters denote a set of time-independent parameters describing how the relative translation and orientation between the two frames CF:1 and CF:2 is parametrized. The most common used in robotics are Denavit-Hartenberg (DH) parameters [34] and related sets by Craig [29] and Khalil and Kleinfinger [76] and separate parameterizations of translation and rotation, e. g., through Euler-angles or quaternions [98]. A model of a simple displacement is shown in Figure 2.7, which represents a simple translation parametrized by matrix ${}^{CF:1}T_{CF:1,CF:2}$. This is referred to as Displacement<Type1> and equivalent to the component called 'FrameTranslation' in [94].

Extracting mathematical operators from the model described in Figure 2.6 shows a behaviour which is very simple in terms of spatial operators. The operator concerned by this kind of ME is the rigid body transformation operator $\phi_{CF:1,CF:2}$ through positional constraint.

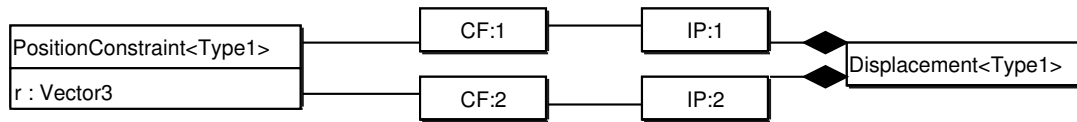


FIGURE 2.7: ER model of the specialized MBS entity Displacement<Type1>.

2.2.1.3 Joint entity

Constrained motion between two bodies results (i) from direct physical contact of two bodies or (ii) from an interconnection of two bodies by means of kinematically constraining mechanisms, whose mass and inertial properties have been lumped with the bodies connected. Important examples often applied in mechanical engineering are revolute, prismatic, universal, and ball joints. It is possible to model the effects of rigid body contact and a constraining mechanism as two coordinate frames F_1 and F_2 whose relative motion is kinematically constrained and the contiguous bodies are thought to be attached w. r. t. to the two frames. In both cases (i) and (ii) the idea of the kinematic equality constraint as a separate entity is called a *joint*. Situations where more than two points or frames have to be constrained at once or the constraint must be expressed in terms of an inequality relation is rarely encountered in robotics and are not investigated in this work. The ideas developed in this section are based on the general mathematical treatment of representations of joints in multibody systems by Schwertassek and Senger [124].

The fundamental characteristic to distinguish the *behaviour* of joints is the way they relate the relative motion of frames F_1 and F_2 . Unconstrained motion between two frames has 6 degrees of freedom (dof), i. e., one requires 6 independent variables to describe the relative position⁷⁾. The set of independent variables parametrizing the *relative* position and velocity are generally called *position and velocity state variables* denoted by symbols q_p and q_v . The number of holonomic and nonholonomic constraint equations be N_{fp} and N_{fg} . The number of motional dof N_f across the joint then is $N_{fp} + N_{fg} = 6 - N_f$.

It has been pointed out in [124] that using relative position and velocity variables allows for all constraint equations to be established without any knowledge about the multibody aspects of the whole system under consideration. This enables a completely component oriented view on every type of joint with a minimum coupling to the complete model. The various joint types can be compiled in an independent joint library which allows for changes in the applied multibody formalism without requiring changes in the library. That perfectly fits into an object-oriented modeling paradigm with demand for encapsulation and minimum but well-defined coupling between all parts of the system. The main characteristic of joint behaviour is captured by a classification of the type of its positional constraint. From a data perspective the requirements for the description of a joint in a multibody dynamics simulation are

- joint positional constraint (desirable in mathematical explicit form),
- sets of joint position state variables q_p and joint velocity state variables q_v ,
- kinematical equations of motion, i. e., the relations between the chosen velocity variables and the time derivatives of the position variables,
- mode vectors and dual vectors, describing the mappings between state space in cartesian space.

There are two more categories given in the joint description [124] not considered in this context for clarity. The first are so-called kinematical excitation functions which describe modes of motion which not excited

⁷⁾ Position encompasses the terms displacement and orientation.

by state variables. This feature of special mechanisms is not required in robotic applications. The second are laws for the generalized applied forces, i. e., functions representing applied interaction forces in terms of the motion across joint most prominently friction, which is later captured by drivetrains including friction.

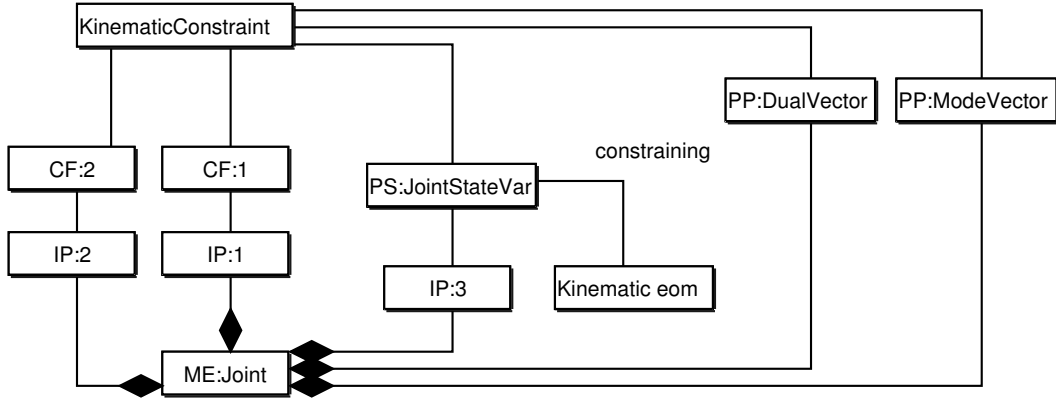


FIGURE 2.8: ER model of the MBS entity Joint.

In the domain of robotics the most important joints are revolute and prismatic which are single dof joints. They constrain the motion between frame F_1 and F_2 along one axis n . One obvious choice for *relative* position and velocity state variables are the joint angle θ and its time derivative $\dot{\theta}$. The kinematical equations of motion take the most simple form $q_v = \frac{d}{dt}q_p$. One concrete specialization is shown in Figure 2.9 and called Joint<Revolute,Type1>. The classifier Type1 denotes the parameterization of the positional constraint equation, in this case using an arbitrary axis of rotation resolved w. r. t. frame $F_{CF:1}$ hence $^{CF:1}n$.

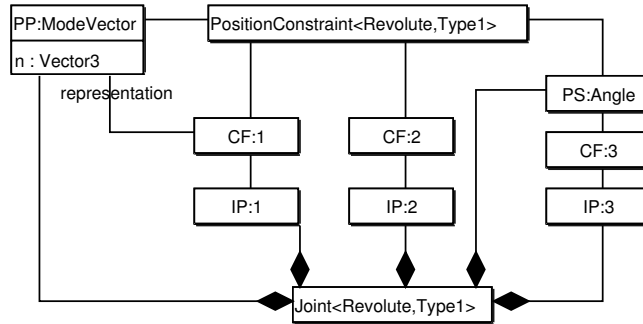


FIGURE 2.9: ER model of the MBS entity Joint<Revolute,Type1>.

The spatial operator induced by the positional constraint is the rigid body transformation operator $\phi_{CF:1,CF:2}$. A crucial point for all algorithms based on a joint space or state space representation – actually the most efficient ones – is a mapping between the spaces of velocity variables and cartesian velocities and its dual mapping between cartesian actions and variables of generalized force variables. In the simplest case of single dof revolute joints this turns out to be the mappings between joint angular velocities and torques and cartesian velocities and actions. In case of holonomic constraints the mode vectors and its dual are equivalent to the *joint projection operator* H [66]. This operator will be applied in the further analysis in the following sections. If $^{CF:1}n$ is a unit column vector describing the axis of rotation of a revolute joint the $H = (^{CF:1}n^\top, 0_3^\top)^\top$, if $^{CF:1}n$ is the joint axis of a prismatic joint $H = (0_3^\top, ^{CF:1}n^\top)^\top$.

2.2.1.4 ReferenceFrame entity

Motion of mechanical systems finally are described w. r. t. inertial space. The idea of a reference frame component is a coordinate frame whose translation and orientation w. r. t. an inertial frame are known functions of time, which may, in particular, be zero [115]. This frame could be reference to one prominent ('reference' or 'main') body attached to the robot mechanism, and is strictly speaking not a part of the mechanism itself. Or one might choose a reference frame corresponding to no material body and having no physical relation with any body of the system. This case can be covered by the idea of a 'conceptual' joint relating the reference frame and the 'main' body. Because this conceptual joint models, depending on context context, different qualities from a 'real' joint entity [115] it is part of the separate ReferenceFrame model shown in Figure 2.10(a). For instance a conceptual joint with six degrees of freedom [40] does not constrain motion in any sense. ReferenceFrame is a purely kinematical concept, though similar components, as reported in literature [94], carry semantical information about gravitational acceleration. As discussed in Section 2.1.1 this implicitly assumes a homogeneous gravitational field.

More important is the meaning of ReferenceFrame as part of the description of the robot's environment. Information is required on which parts of the robot model are interacting with, e. g., the ground if it is mounted on the floor. The specialized model ReferenceFrame<Kinematic,Type1> shown in Figure 2.10(b) is designed to express these conditions. The specialization tag <Kinematic,Type1> tells that the frame is a kinematic reference where the inertial frame coincides with the frame of reference.

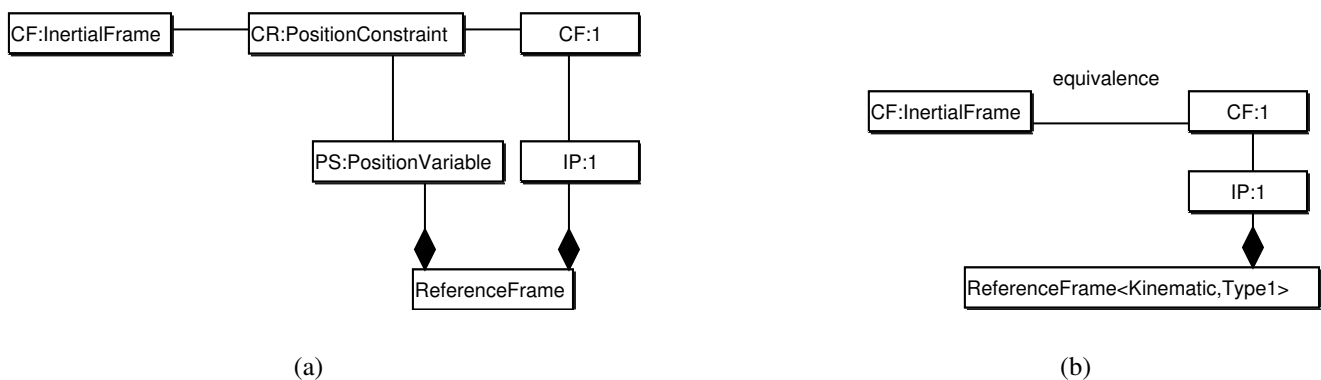


FIGURE 2.10: Left: ER model of the MBS entity ReferenceFrame and its specialization ReferenceFrame<Kinematic,Type1>.





2.2.1.5 Default component set and additional components

Robotics is a rapidly developing domain with a vast range of applications. New types of components and machines are continuously emerging and there is undoubtedly an increasing demand for precision, safety, and efficiency. This is reflected by the modeling requirements where domain specific solutions and new algorithms require new types of components catching more detailed physical effects. In Section 3.2 where algorithms for concrete applications are presented some more types of specialized MBS entities will be introduced, e. g., representing elasticity, physical contact, or angular momentum.

When considering the Multibody entities in the following chapters, prominently in the context of multibody system algorithms we will rely on a default set of components which are listed in Table 2.2. These suffice to describe tree-structured robot mechanisms system with fixed base on a detailed level without further assumptions. The table also introduces an iconic representation für each default entity useful

in drawing multibody component diagrams and shows which spatial operators appears in the equations through the presence of this type of entity in the model description.

TABLE 2.2: Default set of MBS entities.

MBS entity declarator	iconic representation	spatial operator
ReferenceFrame<Kinematic,Type1>		
Joint<Revolute,Type1>		H, Φ
Displacement<Type1>		Φ
RigidBody<1,Type1>		M

2.2.2 Robot specification model

Known interconnection topologies of the robots are:

- kinematic chain
- star topology
- tree structure
- partial loops
- fully closed kinematic mechanisms

Figure 2.11 shows a schematic view of each class of these mechanisms. When mechanisms are depicted schematically throughout this work, small circles symbolize any kind of holonomic joint and ellipses material bodies.

The reason to do this subdivision is that each class has distinct mechanical properties and hence algebraical properties in a mathematical description. For instance the presence of kinematic loops make the numerical treatment of kinematics and dynamics much more intricate due to additional algebraic constraint equations. One main issue of the various existing dynamics formalisms in general and one main topic of this work is to exploit this structural properties to arrive at an efficient numerical representation of the governing equations.

The most common type of mechanism in robotics is the kinematic chain, a series of bodies, where adjacent ones are connected by joints. There is one body with a special status lying on one end of the chain which is called *reference body*, or *robot base*. When several branches emerge from one common reference body the system has star topology. When branches are allowed to emerge from every body in the system, the mechanism is called to have tree structure. In a tree the total number of joints N_j^+ is one less than the total

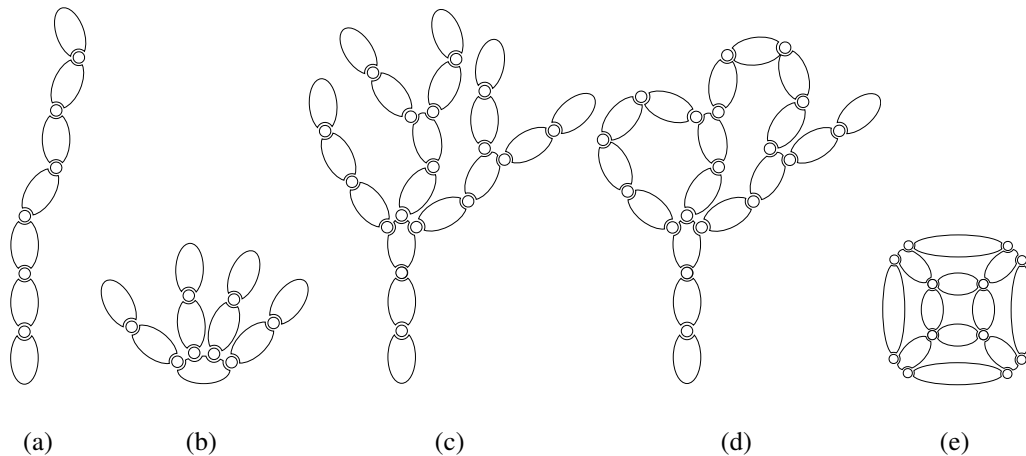


FIGURE 2.11: The main topologies of robotic mechanisms depicted schematically. From left to right: chain structure, star topology, tree structure, partial loops, and fully closed mechanism.

number of bodies N_B^+ . Index $^+$ denotes numbers which are related to the multibody system as a whole. Introducing some additional joints which connect some of the branches of the tree leads to kinematic loops, each extra joint forming one loop. When each body is at least connected to two contiguous bodies by joints the robotic mechanism is called fully closed. Most robots belong to the class of moderately constrained mechanical systems, i. e., the number of mechanical degrees of freedom of the robot equals roughly the total number of bodies and the total number of mechanical dof and the number of kinematic loops is low in comparison to the number of bodies.

Model component interaction is based on a port concept in this thesis, a versatile technique from object-oriented modeling that is sufficiently powerful to handle the MBS domain. Relations between MBS entity objects are established by an *MBS connection* (MC) between two or more interaction ports. The interpretation of an MBS connection is manifold and depends on the level of abstraction one is considering:

- From a modeling perspective it is an undirected relation between two or more interaction ports. It can be represented by pairwise binary relations or by relations between multiple ports. To reduce modeling errors it is possible to enforce or avoid the formation of topological constructs by restricting the possible connections. This has been used, e. g., in [104, 56] to enforce the formation of a spanning tree of the model graph by providing two complementary types of ports, A- and B-types, allowing only for connections between one A- and one B-port. This strategy is adequate if the application relies on tree-structure only. The limited scope of the application, for instance a dynamics algorithm for tree-structure, is mirrored in the modeling paradigm. In this work assuring, if the topology is adequate for a certain application, is postponed to a later logical stage where information is available about the requirements from the algorithm to apply.
- From an MBS perspective it defines a unique location or frame where two or more entities of the MBS model interact mainly mechanically. Depending on port semantics and the type of coordinate frame associated to the interaction port, this corresponds to different levels of detail in physical modeling. One encounters two types in this work. The first is a fixed mechanical connection in a spatial frame living in three dimensional Cartesian space. This can be illustrated by matching all coordinate frames associated to the ports taking part in the MBS connection in one frame. The second is used for drivetrains where the associated coordinate frames are aligned along a conceptual symmetry axis.

- From an implementation perspective kinematic and dynamic data which are exchanged between the components, e. g., force and torque vectors, are represented and available in coordinate frames associated to the 'implemented ports'. This concept is borrowed from the concept of 'cut' actions in mechanics, two examples for this perspective are the works of Leister [85], Kecskeméthy [74]. In the dataflow model presented in the next section a re-interpretation of a multiple relation between N ports to a (tree-structured) set of directed connections is mandatory to result in streams of well-defined input-output causality of the software components representing the MBS entities and their ports.

2.3 Relating recursive dynamics algorithms to dataflow

This section relates the important class of efficient and versatile recursive multibody algorithms to the model of dataflow. This has two advantages: First, the algorithm for a complete robot can be established symbolically from properties and equations describing the components and their interconnection structure. Second, the dataflow model of computation allows for an immediate conversion of the algorithm into code or an executable.

In multibody view a robot is an ensemble of material bodies connected by joints, including some interactions through actuators, elastic elements and external forces such as gravitation. Such systems can be treated mathematically and numerically by a number of formalisms. Modeling a technical system by means of a network of operators or blocks transformings flows of data is well known from the domains of electronic circuits and automatic control. A higher-level description is achieved by using transfer functions with block-diagram structures and the dynamical equations capturing the behaviour of the systems. These formalisms are rather similar to what a *dataflow system* is in the domain of computer science [71, 84]. The PTOLEMY project [23], for instance, studies modeling, simulation, and design of concurrent, real-time, embedded systems. The focus is on assembly of concurrent components. The key underlying principle in the project is the use of well-defined models of computation that govern the interaction between components. A major problem area being addressed is the use of heterogeneous mixtures of models of computation. Model-Integrated Computing [138] has been developed for building embedded software systems. The key element of this approach is the extension of the scope and usage of models such that they form the "backbone" of a model-integrated system development process. The main application domain is signal processing.

The family of recursive multibody formalisms stresses transformation properties of each component and topological properties of the system as a whole. Starting from this observation this section develops a symbolic mathematical representation of multibody systems such as robots which combines information about interconnection structure and equations describing the components. For reasons that become clear in the following this mathematical representation is called *Port-Based Spatial Operator Algebra* (PSOA).

There are distinct advantages of using the dataflow model as a basis for a high-level description which are discussed in [47]. It is a *functional* model with its subsequent mathematical representation, with no complex side effects, amenable to formal verification and safe program transformation, because functional relations over dataflows might be time invariant properties. Moreover it is an inherent parallel model, where any sequencing and synchronization constraints arise from *data* dependencies. The following section develops mathematical operators for the description of multibody system dynamics within the high-level dataflow paradigm.

Port-Based Spatial Operator Algebra

The formalism developed in this section can be viewed as a natural extension of the Spatial Operator Algebra (SOA) introduced by Rodriguez et al. [117]. PSOA extensions (a) offer a more general way to specify and describe mechanisms and (b) alleviate development and object-oriented realization of recursive symbolic multibody algorithms which are applicable to a broad range of robots [63, 68, 65, 66, 67]. As revealed in Section 2.1.1 even the mathematical description of one single body requires considerable algebraical effort. The spatial operators and operator identities derived through SOA help in reducing the symbolic complexity and arriving at more concise expressions. Spatial operators present powerful means to establish *spatial recursions*, i. e., to state the kinematics and dynamics equations of complex rigid multibody systems symbolically and recursively.

The following sections clarify the relation between MBS model *structure* and spatial recursions, i. e., the entities that are to be modeled and the algebraical relationships between them. The idea is that a generalized way of *specifying* the mechanical system's *properties* has a counterpart in the process of *establishing* the system's *equations*, to guide generation and organization of code on a later stage. As a consequence this section does not follow the classical introduction of multibody formalisms and starts from more abstract arguments.

Multibody component diagrams

Consider an N -joint kinematic chain with fixed base, the case $N = 2$ being depicted schematically in Figure 2.12(a). Following robotics literature, bodies are numbered ascending from 0, where index 0 denotes the robot base. Starting from body 0, the joint labelled i is followed by body labelled i . Moving from base body to bodies/joints of increasing index is called *outboard* direction, the opposite direction is called *inboard*.

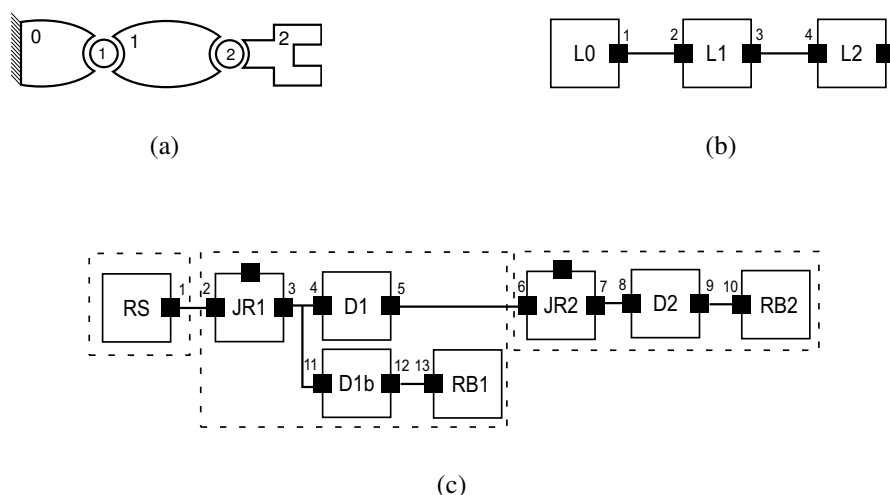


FIGURE 2.12: (a) Schematic view of a tooling 2-joint kinematic chain manipulator and (b) a more abstract graphical representation, (c) a more detailed representation using the default MBS entities from Section 2.2.1.5. Outlined boxes are MBS entities, black squares MBS ports indexed starting from 1, straight lines between ports are MBS connections.

A graphical representation of Figure 2.12(a) where the interconnection structure is more emphasized are shown in Figure 2.12(b) and (c). Joint i and body i and their mechanical connection are merged into a

single component represented by an outlined box. The mechanical connection between joint i and body $i-1$ is symbolized by a straight line terminated by black squares placed on the boxes' border. The compact iconic representation is taken from [127] where it is used to draw Unified Modeling Language (UML) class collaboration diagrams [100] in order to explicitly represent interconnections between architectural entities. At this stage it is sufficient to mention that in [127] outlined boxes represent *capsules*, black squares *ports*, two ports connected by a line a *connector*. We will further refer to such a diagram as *MBS component diagram* (MCD). The outlined and filled boxes in a MCD are iconic representations of *MBS entities* and *MBS ports*. The straight line between two connectors represents a *MBS connection*. The rules for drawing a MCD are defined as follows:

- (i) An MBS entity owns one or more ports which lie on the boundary.
- (ii) An MBS entity may contain an arbitrary number of other MBS entities.
- (iii) Ports must belong to any MBS entity and lie on its border.
- (iv) Connections terminate in ports, nowhere else.
- (v) Connections are irreflexive, they relate two *different* ports. Self-connections are not allowed.
- (vi) Connections are transitive, if port 1 is connected to ports 2 and 3, then there is a connection between 2 and 3. This rule might not be explicit in the MCD.
- (vii) Two different ports may be related by exactly one connection. No parallel connections are possible.

These rules induce a transitive binary symmetric relation on the ordered set of ports. The interconnection structure of an ensemble of MBS entities is captured by the *connectivity matrix* which is introduced here. This matrix determines which pairs of ports are coupled by a connection. Each of all N_p ports, which are present in a MCD, are assigned a unique integer value $\{1, \dots, N_p\}$, as shown in Figure 2.12(b). Identifying this value with a component index the connectivity matrix $C \in \mathbb{F}^{N_p \times N_p}$ is defined by

$$C := \begin{cases} C_{i,j} = 1, & \text{if there is a connection between port } i \text{ to } j, \\ C_{i,j} = 0, & \text{otherwise,} \end{cases} \quad (2.28)$$

where \mathbb{F} is the set of numbers $\{0, 1\}$ with addition and multiplication modulo 2. C is inherently symmetric as the relation 'connection' is symmetric and all diagonal elements are zero because of rule (v). Ports which are not connected are *open* and lead to zero columns/rows in C .

Roberson and Wittenburg introduced a graph theoretical description of multibody systems [116, 149]. A comprehensive overview of applications of graph theory in MBS dynamics can be found in [115]. Some required definitions and properties are reproduced here. A *graph* $\mathcal{G}(\mathfrak{N}, \mathcal{E})$ is defined as the nonempty set of *nodes* (or *vertices*) \mathfrak{N} and a set \mathcal{E} of pairs of these nodes, called *edges*. If the set \mathcal{E} is ordered one can assign a direction to each edge and the edges and the graph are called *directed*. Otherwise the graph is called *undirected*. We are concerned with graphs that are *simple*, i. e., that contain no self-loops and multiple edges and which are connected.

If ports are omitted from the MCD and each entity and its ports are represented by one node one obtains a connected graph \mathcal{G} . Undirected graph edges represent connections. The connectivity C^+ of the graph is obtained from the connectivity of the MCD using the port-entity projection operator $PE \in \mathbb{F}^{N_p \times N_e}$ defined by

$$PE := \begin{cases} PE_{i,j} = 1, & \text{if port } i \text{ belongs to entity } j, \\ PE_{i,j} = 0, & \text{otherwise} \end{cases}$$

which requires a sequential numbering of the nodes starting from 1. PE can be used to collapse C to get $C^+ := PE^T \cdot C \cdot PE$. C^+ determines connectivity between MBS entities and not between ports.

Causality and spatial recursions

Starting from the visual syntax of the MCD we are now able to specify a graphical description resulting in a directed graph where nodes represent entities from the multibody domain and edges represent a mechanical interrelation. Turning to recursive multibody computations, either symbolic or numerical, the notion of a *dataflow process network* [84] becomes valuable, where nodes represent transformations and edges *directed* streams of data, in MBS domain: usual kinematical and dynamical values.

The first step is to refine the connectivity by assigning a direction to all connections. Thus each port is represented by a unit vector $u_i \in \mathbb{F}^{N_p}$, $i \in 1, \dots, N_p$ and either defined to be *outport* or *inport*. A directed connection points from one outport to one inport. The unit vectors of all outports form the columns of the matrix O and span the *outport space* and those of all inports form I and span the *inport space*. $\{O|I\}$ forms a complete basis of the linear vector space \mathbb{F}^{N_p} , if each port is either defined in- or outport. For example, the MCD from Figure 2.12, if ports 0, 2, 4 are defined outports, results in

$$C = \begin{bmatrix} 0 & 1 & & & \\ & 1 & 0 & & \\ & & & 0 & 1 \\ & & & 1 & 0 \\ & & & & & 0 \end{bmatrix}, \quad O = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \quad \text{and} \quad I = \begin{bmatrix} 0 & 0 \\ 1 & 0 \\ 0 & 0 \\ 0 & 1 \\ 0 & 0 \end{bmatrix}.$$

Proceeding from this pure topological considerations the next step adds new connection semantics and defines which dataflow quantity⁸⁾ is used and transferred between MBS ports. As a first example we choose the spatial velocity $V \in \mathbb{R}^6$ and associate one vector to each of the N_p ports. The N_p velocities are stacked to form the *stacked* spatial velocity $V \in \mathbb{R}^{6N_p}$. The i^{th} spatial component vector of any stacked vector is denoted by right lower index $V_{[i]} \equiv V_i$ in square brackets.

In order to state transformation properties of the MBS entities we add new port semantics and associate a frame F_i in Euclidean space with each port i . In our example obviously $F_1 \equiv F_I$ and $F_5 \equiv F_{N_p}$ is the tool frame. The semantics of an MBS entity now is to transform data entering from all the inports to all outports. For the moment it suffices to think of a linear transformation similar to (2.19).

A good example to explain the transformation property between ports are 2-port MBS entities representing the coupled joint-body pair from Figure 2.12, where i is the port at the joint side and $i + 1$ the port on the body side. The spatial velocities of frames F_i and F_{i+1} associated to the ports, $V_{[i]}$ and $V_{[i+1]}$ are related by

$$V_{[i+1]} = \phi_{i+1,i} V_{[i]} + \Delta_{i+1}. \quad (2.29)$$

The relative rate of velocity change across the joint Δ_{i+1} depends on the inner properties of the MBS entity, e. g., the joint angle q . The spatial velocity $V_{[i]}$ transforms under the rigid body transformation operator $\phi_{i+1,i}$ analogously to (2.19). Please note that the expressions derived are independent of coordinate representation R , thus index $^{(R)}$ is omitted. The rigid body transformation (2.29) of all MBS entities in a MCD is captured by the *stacked linear rigid body transformation matrix* T_ϕ . The transformations from $i \rightarrow i + 1$ as well as $i + 1 \rightarrow i$ can be covered uniformly because $\phi_{i+1,i}$ is invertible. For the example this

⁸⁾ later we will see that this is related to the notion of a *protocol* [127].

results in

$$V = \begin{pmatrix} V_1 \\ V_2 \\ V_3 \\ V_4 \\ V_5 \end{pmatrix}, T_\phi = \begin{bmatrix} 0 & & & & \\ & 0 & \phi_{2,3} & & \\ & \phi_{3,2} & 0 & & \\ & & & 0 & \phi_{4,5} \\ & & & \phi_{5,4} & 0 \end{bmatrix}, \text{ and } \Delta = \begin{pmatrix} 0 \\ \Delta_2 \\ \Delta_3 \\ \Delta_4 \\ \Delta_5 \end{pmatrix}.$$

The final goal of a recursive algorithm is to derive an iterative method by successive transformation of a dataflow quantity in a particular direction and between particular ports. To put it in other words: according to a given causality. Please note that this form of causality does not invoke *time*. Inspired by Wittenburg [149] assigning a direction to each connection results in a directed MCD. He distinguishes between an undirected MBS *system graph* to define topology and components and a *directed graph* to enable unique assignment of relative motion and direction of forces.

Inboard or outboard causality is determined by projecting the involved stacked operators C , T_ϕ , V , and Δ into the associated inport and outport spaces using the projection operators

$$\hat{O} := (OO^\top) \cdot I_{6 \times 6} \quad \text{and} \quad \hat{I} := (II^\top) \cdot I_{6 \times 6}. \quad (2.30)$$

Multiplication by unit matrix $I_{6 \times 6}$ denotes identification of \mathbb{F} by $\{0_{6 \times 6}, I_{6 \times 6}\}$, i. e., scalar numbers are replaced by matrices $\in \mathbb{R}^{6 \times 6}$. The *outboard transfer matrix*

$$C^\uparrow := \hat{I}C\hat{O} \quad (2.31)$$

guides the dataflow from outports to inports via connections. The arrow \uparrow is used to denote outboard quantities. Its complement is the outboard rigid body matrix

$$T_\phi^\uparrow := \hat{O}T_\phi\hat{I}, \quad (2.32)$$

which transforms inport data to outport data. This leads to the definition of the *outboard rigid shift operator*

$$\mathcal{E}_\phi^\uparrow := T_\phi^\uparrow + C^\uparrow, \quad (2.33)$$

which transforms the dataflow one stage outboard, from each inport to the adjacent outport and from each outport the adjacent inports. In case of our example this is

$$\mathcal{E}_\phi^\uparrow = \begin{bmatrix} 0 & & & & \\ 1 & 0 & & & \\ & \phi_{3,2} & 0 & & \\ & & 1 & 0 & \\ & & & \phi_{5,4} & 0 \end{bmatrix}.$$

The outboard recursion for the spatial velocities in terms of stacked operators gives

$$V = \mathcal{E}_\phi^\uparrow V + \hat{O}\Delta. \quad (2.34)$$

For tree-structured systems it has been shown that $\mathcal{E}_\phi^\uparrow$ is nilpotent [117]. With the definition of the *rigid manipulator velocity transformation*

$$\Phi := \sum_{i=0}^N (\mathcal{E}_\phi^\uparrow)^i \quad (2.35)$$

and following the ideas from [117] one can use the nilpotency to get the fundamental inverse

$$\Phi = (I_{6N_p \times 6N_p} - \mathcal{E}_\phi^\uparrow)^{-1}, \quad (2.36)$$

and (2.34) writes

$$V = \Phi \hat{O} \Delta. \quad (2.37)$$

This identity is an *explicit* expression for the spatial velocity of frames associated to *all* ports present in the MCD. Often one desires a more coarse spatial recursion, leading to evaluation of (2.37) on a sub-set of ports. Evaluation is taken to be symbolic. The sub-space is spanned by columns of matrix E with orthogonal space spanned by E^\perp . Restriction to this subset means that dataflow in ports $\in E^\perp$ does not explicitly appear in the final symbolic recursion. There is not necessarily a graphical representation of this recursion as MCD. The restricted recursion denoted by subscript $|_E$ can be shown to have the same structure as (2.37):

$$V|_E = \Phi|_E \Delta|_E, \quad (2.38)$$

where

$$\Phi|_E := E^\top \Phi E \quad (2.39)$$

$$\Delta|_E := (E^\top + \Phi|_E^{-1} E^\top \Phi E^\perp E^{\perp\top}) \hat{O} \Delta. \quad (2.40)$$

Proof:

$$\begin{aligned} E^\top V &= E^\top \Phi \hat{O} \Delta \\ &= E^\top \Phi (EE^\top + E^\perp E^{\perp\top}) \hat{O} \Delta \\ &= \Phi|_E E^\top \hat{O} \Delta + (\Phi|_E \Phi|_E^{-1}) E^\top \Phi E^\perp E^{\perp\top} \hat{O} \Delta \\ &= \Phi|_E (E^\top + \Phi|_E^{-1} E^\top \Phi E^\perp E^{\perp\top}) \hat{O} \Delta \quad \square \end{aligned}$$

The special case $E = O$ will be discussed in more detail: Using $V|_O = O^\top V$ can be shown to lead to the stacked symbolic recursion

$$V|_O = O^\top \mathcal{E}_\phi \uparrow^2 O V|_O + \Delta|_O. \quad (2.41)$$

The squared shift operator stems from the property that an outport quantity first is transformed into an inport quantity. As we would like to restrict to outboard quantities, i. e., $E = O$ one has to perform two outboard shifts in sequence between two successive ports. They are referred to as *sub-recursions*. Equation (2.41) is an important result as it recasts the spatial recursion for the spatial velocity derived by Rodriguez et al. [117] but including the topology of the multibody system. The iterative algorithm can be established easily from the reduced PSOA expression (2.41). Defining a *sweep* as a single recursion along a kinematic chain or tree structure in either direction, the single outboard sweep for the case of the example 2-dof manipulator is

```

N = 2, V[0] = 06
for i=1 to N do {Outboard sweep}
  V[i] = φi,i-1 V[i-1] + Δ[i]
end for

```

A fundamental difference must be noted between the definition of an MBS graph introduced here and that used commonly in literature, e. g., Wittenburg [149], Roberson and Schwertassek [115]. The usual MBS graph interpretation represents bodies by nodes and joints and springs by edges. It is called *primary graph* in [149]. Other ways of interaction such as forces are sometimes represented by edges, too. A complementary definition would be the *port connection graph* [110], where all elements of the mechanical network are associated with an *edge*. The presented approach treats all entities of interest, bodies, joints, etc. uniformly as MBS entities to emphasize its active role while acting on dataflows. Edges have pure dataflow semantics and carry no dynamic transformation semantics, neither physically nor in symbolic expressions, making them purely passive. The next section discusses the advantages of this point of view on the graph and components of a multibody system.

Chapter 3

Multibody dynamics algorithms for robots

3.1 Fundamental algorithms for rigid manipulators revisited

Enabled by the PSOA formulation even complex multibody algorithms can be described in a purely mathematical but object-oriented way without losing any rigor while being at the same time very close to object-oriented implementation. This chapter recasts fundamental algorithms emerging from standard problems in robot control in a new component oriented form using the new Port-Based Spatial Operator Algebra formulation presented in Section 2.3. This forms a powerful basis for immediate object-oriented implementation at a later stage. The algorithms presented will mainly rely on the following set of modeling assumptions, concerning physical aspects of the robot and its environment:

- (i) rigid links and joints
- (ii) fixed base
- (iii) tree-structure
- (iv) only joints with one mechanical dof
- (v) fully actuated, i. e., each joint is driven
- (vi) homogeneous gravitational field

These assumptions hold for a large range of robotic systems and are used most commonly in literature. Algorithms based on these assumptions are among the most efficient ones known [136, 90] and so qualify naturally for real-time control applications. Please note, that the equations in this section describe exclusively three-dimensional robot models. The mathematical notation of the presented algorithms relies on the spatial notation introduced in Section 2.1.2. When stacked notation is applied to describe a complete system, then we presume a causality definition step, as described in Section 2.3, has been performed already.

3.1.1 Recursive algorithms—general considerations

It has been shown in Section 2.2.2 that most robotic systems have a moderately connected topology. A recursive algorithm exploits this fact by calculating *iteratively* kinematical and dynamical quantities by

starting from a certain location or coordinate frame and applying local transformations based on local states to calculate, e. g., the position of each coordinate frame present in the mechanical model. One iteration is called a *sweep*. In terms of matrix computations the system to solve is very sparse and this sparsity is directly exploited by the algorithm induced by MBS connection topology. Recursive algorithms are known to be (i) very efficient, (ii) easily modularizable, (iii) hence require low computing resources, and are therefore well-suited for application in embedded systems [87].

Exploiting sparsity in each algorithm presented in this chapter is enabled by operators acting on a dataflow which is established between the interaction ports of MBS entities. These operators are interpreted as algorithm specific *mathematical* instantiations of the ME classes introduced by the specification paradigm in Section 2.2.1. The algorithm specific dataflow between IPs is an instantiation of the abstract MBS connections. To emphasize the property of receiving and transforming dataflow these kind of component is called in some works *transmission object* [74]. The requisites required to completely describe a given dynamics algorithm are:

- coordinate representation
- choice of a protocol [126], mainly the dataflow depending on algorithm and coordinate representation, in other words static IP semantics.
- MBS entity semantics: Transformations/equations between the interaction ports for given causality – the expected behaviour for each ME class in a certain algorithm.
- requirements and local constraints, to model algorithm specific properties and restrictions. For instance if only certain types of joints are supported by the mathematical formalism. Famous examples are the dynamics algorithms by Walker and Orin [147] formulated for prismatic and revolute joints and Armstrong [12] for spherical joints.

This work suggests a categorization of dynamics algorithms which is not restricted to numerical evaluation of mathematical quantities. We propose a dynamics algorithms 'space' which is spanned by the categories of

- (i) desired mathematical result, i. e., joint torques,
- (ii) used internal coordinate representation,
- (iii) port semantics, the internal dataflow coupling the components, and
- (iv) set of additional requirements.

This will allow for a more flexible design of algorithm components in Chapter 4.3 to alleviate exchange of code. This additional degree of freedom in design can be applied to create flexibly the most efficient robot control system architecture.

3.1.1.1 Data-Transfer Protocol and Ports

The complete set of data or *messages* exchanged between two parties in the MBS component diagram, i. e., two different interaction ports, is called data flow or *protocol* [126]. It comprises (i) the type of data exchanged, and (ii) a relative order in which the data is exchanged, and (iii) a direction in which the data is sent.

The first two, (i) and (ii), solely depend on the kind of dynamics algorithm to be performed. The last also depends on how the topology of the system has been specified, it is determined by the causality of the MBS components. It is essential for proper function of a software to define clearly the meaning of the data flow variables used in a protocol for three reasons: to allow for code reuse, maintenance and for optimizations. One certain type of protocol is symbolized by `<algorithmtag,porttag>` where the two tags denote the kind of algorithm and port semantics.

As an introduction the first example presented is the calculation of the forward kinematics. That is determination of the position and orientation and their time derivatives in Cartesian space for each location of interest in the mechanical model from given joint position and velocity variables. One prominent example is to calculate the tool-frame of a manufacturing robot. Because this section is restricted to tree-structured systems the algorithm comprises one sweep iterating from the base location outboard to the leafs of the multibody system by successively applying transformations prescribed by the components' behaviour.

Forward kinematics to some extent is a prerequisite for nearly any type of dynamics computation. In presence of closed kinematic loops the equations to solve might be rather complex and have no or multiple solutions, e. g., see [150, 115].

The protocol data in a *spatial* kinematics recursion is listed in Table 3.1 on the facing page. The messages, of course, are spatial position, orientation, and velocity. We use a unique symbol in SansSerif shown in the left column to emphasize the equal quality of the messages explained in the second column. The data structure in implementations of various coordinate representations would also be identical. The various mathematical interpretations of the message symbol depends on the coordinate representation chosen shown in the remaining columns. Orientation can be expressed in various ways, for instance in terms of angle parametrizations, quaternions, or direction cosine matrices. To avoid the singularity problem of angle representation direction cosine matrices are chosen. These also can be used to resolve tensorial quantities w. r. t. local frames. The representation of velocity is depending on the coordinate representation employed.

Requirements for a well-posed recursive forward kinematics are:

- each interaction point is associated to coordinate frame F_n , which are called *spatial ports*
- existence of a unique reference frame F_0
- tree-structured topology
- input values are joint position and velocity states q_p and q_v

Joint variables are provided via *state ports*, that are interaction ports where different port semantics applies. The messages required are joint position and joint velocity as shown in Table 3.2. This work concentrates on relative joint variables, so the symbol to denote this coordinate representation for joint state variables is suppressed for the sake of brevity.

3.1.1.2 Component Space for MBS entities

The representation of an MBS entity within a certain algorithm or task is only valid for a context which is determined by:

- (i) given numerical algorithm (see above)

TABLE 3.1: Protocol $\langle \text{RKIN}, 6\text{D} \rangle$ required in spatial forward kinematics recursions. Please note a frame F_n is associated to this protocol.

message symbol	explanation	semantics depending on coordinate representation		
		$\langle I \rangle$	$\langle B \rangle$	$\langle A \rangle$
p	distance from O_0	$\langle I \rangle r_{0,n} = {}^I r_{0,n}$	$\langle B \rangle r_{0,n} = {}^n r_{0,n}$	$\langle A \rangle r_{0,n} = {}^I r_{0,n}$
R	orientation w. r. t. ${}_0\mathbf{e}$	${}^0 R_n$	${}^0 R_n$	${}^0 R_n$
V	spatial velocity	$\langle I \rangle V_n = \begin{pmatrix} {}^0 \dot{\omega}_{0,n} \\ {}^0 \dot{r}_{0,n} \end{pmatrix}$	$\langle B \rangle V_n = \begin{pmatrix} {}^n \dot{\omega}_{0,n} \\ {}^n \dot{r}_{0,n} \end{pmatrix}$	$\langle A \rangle V_n = \langle I \rangle \phi_{I,n} \langle I \rangle V_n$

TABLE 3.2: Protocol $\langle \text{RKIN}, 1\text{D} \rangle$ required in scalar forward kinematics recursions.

message symbol	explanation	semantics
p	joint position variable	q_p
V	joint velocity variable	q_v

- (ii) given interaction port semantics, i. e., the *protocol* associated to each port
- (iii) given causality, i. e., direction of the transformation, takes messages from which interaction port to which one

In order to categorize MBS components, it is useful to introduce a component space spanned by the type of described MBS entity, and the three criteria above. In Table 3.3 the transformation properties of the default set of MBS entities are described within the given categorization.

The default MBS entities with two ports can have a reverse causality, required, e. g., if the displacement is 'mounted' in a reverse manner in the model specification. In this case the reverse causalities are easily obtained, because the rigid body transformation is generally invertible. The PSOA formulation does not presume a defined causality, but subsumes several ones in one symbolic expression, if possible.

Recall Section 2.3 to obtain the complete closed form and *symbolic* algorithm, amenable to further treatment. The algorithm for a certain robot can now be expressed by

- (i) the connectivity C of the model, which is a structural hence algorithm invariant property,
- (ii) determining the causality, which is an algorithm specific strategy depending on C , and the reference nodes, for example depth-first search, and
- (iii) assembling the spatial operators describing all N_p system ports by stacking the relevant local properties and messages of the present MBS entities in larger matrices. The typesetting is, e. g., $V \in \mathbb{R}^6$ for the spatial velocity and $V \in \mathbb{R}^{6 \cdot N_p}$ to denote the stacked pendant. When an MBS entity does not contribute to a certain operator, e. g., the stacked joint projection operator H , the missing entries are filled by zero-blocks.

In forward kinematics the second issue is quite simple, one chooses a reference port which then is connected to the reference system port. Starting from there, one can determine causality by performing a

TABLE 3.3: Spatial forward kinematics transformations for the default MBS entities.

component: Displacement<Type1>			
port semantics: $\{1, 2\} \leftarrow \{3D, 3D\}$			
msg.	$\langle I \rangle$	$\langle B \rangle$	$\langle A \rangle$
	${}^0r_{1,2} = {}^0R_1 r$ $\phi_{2,1} = \phi_D({}^0r_{1,2})$ $T_\phi := \begin{bmatrix} & & \\ & & \phi_{2,1}^\top \\ \phi_{2,1} & & \end{bmatrix}$	$\phi_{2,1} = \phi_D(r)$ $\phi_{2,1}^\top$	see $\langle I \rangle$ $\phi_{2,1} = I_{6 \times 6}$ $V := \begin{pmatrix} V_1 \\ V_2 \end{pmatrix}$
p	$p_2 = p_1 + {}^0r_{1,2}$	$p_2 = p_1 + r$	see $\langle I \rangle$
R	$R_2 = R_1$		
V	$V = T_\phi V$		

component: Joint<Revolute,Type1>			
port semantics: $\{1, 2, 3\} \leftarrow \{3D, 3D, 1D\}$			
msg.	$\langle I \rangle$	$\langle B \rangle$	$\langle A \rangle$
	$u := n/ n , \theta := q_p^3, \dot{\theta} := \dot{q}_v^3, {}^1R_2$ from (B.1)		
	$H := \langle I \rangle H = \phi_R(R_1) \langle B \rangle H$ $\phi_{2,1} = I_{6 \times 6}$	$H := \langle B \rangle H = \begin{pmatrix} u \\ 0_3 \end{pmatrix}$ $\phi_{2,1} = \phi_R({}^2R_1)$	$H := \langle A \rangle H = \phi_{0,1} \langle B \rangle H$ $\phi_{2,1} = I_{6 \times 6}$
	$H := \begin{pmatrix} -H \\ H \end{pmatrix}, T_\phi =$	$\begin{matrix} \phi_{2,1}^\top \\ \phi_{2,1} \end{matrix}$	$V = \begin{pmatrix} V_1 \\ V_2 \end{pmatrix}$
p	$p_2 = p_1$	$p_2 = {}^2R_1 p_1$	see $\langle I \rangle$
R	$R_2 = R_1 \cdot {}^1R_2$		
V	$V = T_\phi V + Hq_v^3$		

component: ReferenceFrame<FixedBase>	
port semantics: $\{1\} \leftarrow \{3D\}$	
msg.	symbolic transformation
p	$\langle I \rangle, \langle B \rangle, \langle A \rangle$ $p_1 = 0_3$
R	$\langle I \rangle, \langle B \rangle, \langle A \rangle$ $R_1 = I_{3 \times 3}$
V	$\langle I \rangle, \langle B \rangle, \langle A \rangle$ $V_1 = 0_6$

depth-first-search [140] on all spatial ports. All state ports are defined as inports because they provide the mathematical input data, the joint positions. The forward kinematics on velocity level for any robot model formulated in terms of the components listed in Table 3.3 can be compactly written as

$$V = \mathcal{E}_\phi V + Hq_v.$$

In this expression causality is already defined and the operators are restricted to spatial ports. Applying identity (2.36) one obtains the *rigid velocity shift operator* Φ [66]. This gives the closed form forward velocity kinematics valid for tree structured systems

$$V = \Phi Hq_v.$$

Note this expression is independent of the three coordinate representations regarded.

3.1.2 Inverse dynamics

This section presents the standard recursive inverse dynamics algorithm for rigid manipulators, but recasted in new component oriented style as required by the Port-Based Spatial Operator Algebra presented in Section 2.3. This allows for further symbolical manipulation vital for deriving new advanced algorithms, as will be shown for instance in Section 3.2.1, as well as direct object-oriented implementation.

In the context of robotics *inverse dynamics* denotes the computation of all driving forces from predefined trajectories, e. g., joint positions. The ability to calculate the inverse dynamics is a basic prerequisite for path planning algorithms and feed forward controllers in robotic systems, so there is plenty of literature on its numerical calculation, e. g., [87, 109, 60]. The computation of the inverse dynamics for rigid robots in tree structure is described by the well-known joint-space equations of motion for tree-structured rigid MBS

$$\mathcal{M}(q)\ddot{q} + \mathcal{C}(q, \dot{q}) + \mathcal{G}(q) = u, \quad (3.1)$$

and given the desired joint positions $q(t)$ and their derivatives w. r. t. time up to a order of two, the force exerted by the drives $u(t)$ can be calculated algebraically. \mathcal{M} , \mathcal{C} , and \mathcal{G} , are joint space mass matrix, matrices of gyroscopic and gravitational effects. There exist several formalisms to derive these equations, the most prominent are Lagrangian and recursive Newton-Euler (RNE) formalisms leading to equivalent results [129]. Without any precautions taken Lagrangian formalisms lead to algorithms of complexity $\mathcal{O}(N^4)$. The complexity of the most efficient inverse dynamics algorithms based on recursive arguments is $\mathcal{O}(N)$, either starting from Lagrangian [53] or Newtonian arguments, the latter reported firstly by Luh et al. [87] and with minimum operations by Khalil and Kleinfinger [77]. An extensive survey of methods and computational complexity can be found in the paper by Li [86]. The classical RNE comprises a sequence of two sweeps [133]:

- sweep 1 (outboard): calculate position, spatial velocities, accelerations for each IP.
- sweep 2 (inboard): calculate spatial force in each spatial IP and joint torques in each state IP.

The RNE inverse dynamics has been formulated in various coordinate representations. Inertial representation is used by Jain [60], Featherstone [40] uses a kind of absolute representation, and Park and Bobrow [108] an equivalent formulation of the body fixed representation, which has been shown by Hardt [49]. The protocol required for the calculations extends the kinematics protocol described in Table 3.1.

TABLE 3.4: Protocol <RNE,6D> required in spatial inverse dynamics recursions.

message symbol	explanation	sw.	semantics depending on coord. repr.		
			$\langle I \rangle$	$\langle B \rangle$	$\langle A \rangle$
\dot{V}	spatial acceleration	1	$\frac{d}{dt} \langle I \rangle V_n = \langle I \rangle \dot{V}_n$	$\frac{d}{dt} \langle B \rangle V_n$	$\frac{d}{dt} \langle A \rangle V_n$
$\dot{V}g$	spatial acceleration due to gravitation	1	$\langle I \rangle \dot{V}_{g_n} = \begin{pmatrix} 0_3 \\ I g \end{pmatrix}$	$\langle B \rangle \dot{V}_{g_n}$	$\langle A \rangle \dot{V}_{g_n}$
f	spatial force	2	$\langle I \rangle f_n$	$\langle B \rangle f_n$	$\langle A \rangle f_n$

A prerequisite for sweep 1 is the calculation of position and orientation of each MBS entity. Due to the structure of the underlying Hamiltonian system the fundamental equations of motion of a MBS consisting of joints and bodies do *not* depend on absolute position if there are no potentials dependent on absolute or

TABLE 3.5: Protocol <RNE,1D> required in 1D scalar inverse dynamics recursion.

message symbol	explanation	sweep	semantics
\dot{V}	joint acceleration variable	1	q_a
f	joint generalized force variable	2	u

relative position. Position calculation can therefore be omitted for 'pure' dynamics. Gravitation couples to material bodies carrying mass. The coupling is provided in this realization by the protocol which 'supplies' each material body with a gravitational acceleration. In case of a position independent gravitational field one can apply a pseudo-acceleration of the reference system for two reasons: (i) the point of application of gravitation resultant is identical to the point of application of inertial forces and (ii) gravitational acceleration is position independent. An MBS context could provide this information to select a mode for efficient code generation at a later stage.

The transformation properties for the inverse dynamics contribution of each MBS entity are shown in Table 3.6 and Table 3.7. They extend the kinematics transformations from Table 3.3. An object-oriented implementation can exploit that relation through, e. g., an inheritance relation. When compared to spatial kinematics the inverse dynamics recursions require some additional spatial operator matrices related to properties of certain MBS entities. Stacked diagonal spatial inertia M , stacked antisymmetric spatial velocity \tilde{V} , stacked antisymmetric spatial angular velocity $\tilde{\Omega}$, and stacked antisymmetric joint velocity contribution $\tilde{\Delta}$.

TABLE 3.6: Recursive Newton Euler inverse dynamics transformations for default MBS entities ReferenceFrame<FixedBase> and Displacement<Type1> and three coordinate representations. The columns msg. and sw. denote the involved message type and sweep number.

component: ReferenceFrame<FixedBase>			
port semantics: $\{1\} \leftarrow \{3D\}$			
msg.	sw.	symbolic transformation	
\dot{V}	1	$\langle I \rangle, \langle B \rangle, \langle A \rangle$	$\dot{V}_1 = 0_6$
\dot{V}_g	1	$\langle I \rangle, \langle B \rangle, \langle A \rangle$	$\dot{V}_{g_1} = \begin{pmatrix} 0_3 \\ g \end{pmatrix}$

component: Displacement<Type1>			
port semantics: $\{1, 2\} \leftarrow \{3D, 3D\}$			
msg.	sw.	symbolic transformation	
		$\dot{V} = \begin{pmatrix} \dot{V}_1 \\ \dot{V}_2 \end{pmatrix}, f = \begin{pmatrix} f_1 \\ f_2 \end{pmatrix}$	
\dot{V}	1	$\langle I \rangle$	$\tilde{\Omega} := \text{diag}(\tilde{\Omega}_1, \tilde{\Omega}_2)$ $\dot{V} = T_\phi \dot{V} + \tilde{\Omega} T_\phi V - T_\phi \tilde{\Omega} V$ see 2.20
		$\langle B \rangle, \langle A \rangle$	$\dot{V} = T_\phi \dot{V}$
f	2	$\langle I \rangle, \langle B \rangle, \langle A \rangle$	$f = T_\phi^T f$
\dot{V}_g	1	$\langle I \rangle, \langle B \rangle, \langle A \rangle$	$\dot{V}_g = T_\phi \dot{V}_g$

An interesting observation is that each kind of displacement in absolute representation is completely pas-

TABLE 3.7: Recursive Newton Euler inverse dynamics transformations for default MBS entities Displacement<Type1> and RigidBody<1,Full,Type1> and three coordinate representations. The columns msg. and sw. denote the involved message type and sweep number.

component: Displacement<Type1>			
port semantics: $\{_{1,2}\} \leftarrow \{3D, 3D\}$			
msg.	sw.	symbolic transformation	
		$\dot{V} = \begin{pmatrix} \dot{V}_1 \\ \dot{V}_2 \end{pmatrix}, f = \begin{pmatrix} f_1 \\ f_2 \end{pmatrix}$	
\dot{V}	1	$\langle I \rangle$	$\tilde{\Omega} := \text{diag}(\tilde{\Omega}_1, \tilde{\Omega}_2)$ $\dot{V} = T_\phi \dot{V} + \tilde{\Omega} T_\phi V - T_\phi \tilde{\Omega} V$ see 2.20
		$\langle B \rangle, \langle A \rangle$	$\dot{V} = T_\phi \dot{V}$
f	2	$\langle I \rangle, \langle B \rangle, \langle A \rangle$	$f = T_\phi^T f$
\dot{V}_g	1	$\langle I \rangle, \langle B \rangle, \langle A \rangle$	$\dot{V}_g = T_\phi \dot{V}_g$

component: RigidBody<1,Full,Type1>			
port semantics: $\{_1\} \leftarrow \{3D\}$			
msg.	sw.	symbolic transformation	
	2	$\langle B \rangle$	$M_{[1]} = \langle B \rangle M_1 = \begin{bmatrix} I - m\widetilde{rCMrCM} & m\widetilde{rCM} \\ -m\widetilde{rCM} & mI_{3 \times 3} \end{bmatrix}, \tilde{\Omega}_{[1]} := \text{diag}(I_{3 \times 3}, 0_3)V_1$
		$\langle I \rangle$	$M_{[1]} = \phi_R(R_1)^T \langle B \rangle M_1 \phi_R(R_1)$
		$\langle A \rangle$	$M_{[1]} = \phi_{0,1}^T \langle B \rangle M_1 \phi_{0,1}$
f	2	$\langle I \rangle$	$f_1 = M_{[1]}(V_1 + \dot{V}_{g_1}) - (M_{[1]} \tilde{\Omega}_{[1]} + \dot{V}_1^T M_{[1]})V_1$
		$\langle B \rangle, \langle A \rangle$	$f_1 = M_{[1]}(V_1 + \dot{V}_{g_1}) - \dot{V}_1^T M_{[1]} V_1$

sive w. r. t. dynamics, reflecting its purpose of just moving the reference point. The complete closed form inverse dynamics algorithm in PSOA is shown in Table 3.8 on the next page for pre-defined causality. Expanding Φ using (2.36) will again lead to the well-known two sweep calculation, one outboard sweep from base-to-tip to calculate spatial velocity and acceleration and one tip-to-base inboard sweep to calculate the spatial and joint forces.

Now turning again to the joint space representation of the dynamics reveals explicit factorizations of the joint space matrices by means of spatial operators. Expanding the symbolic expression for u leads to

$$u = H^T \Phi^T \left(M(\Phi(Hq_a - \tilde{\Delta}V) + \dot{V}_g) - \tilde{V}^T M V \right) \quad (3.2)$$

which can be shown to be independent from the chosen coordinate representation $\langle I \rangle$, $\langle B \rangle$, or $\langle A \rangle$. Comparison to (3.1) gives the *Newton-Euler factorization* of the joint space mass-inertia matrix \mathcal{M} [119]:

$$\mathcal{M} = H^T \Phi^T M \Phi H \quad (3.3)$$

Extending the standard set of ME will lead to an augmented set of stacked spatial operators and hence extensions in the closed form PSOA expressions, if additional physical effects are included. Please note that efficiency in terms of floating point operations for the three inverse dynamics formulations is different. An assessment of the most efficient method is depending on the topology of the system and the set of required of MBS entities, but for the settings described in Section 3.1 representation $\langle B \rangle$ will be most efficient [136].

TABLE 3.8: PSOA expressions for RNE inverse dynamics in inertial, body, and absolute coordinate representation for the standard set of MBS entities.

$\langle I \rangle$	$\dot{V} = \Phi \left(Hq_a - \tilde{\Delta}V \right) + \tilde{\Omega}V$ $f = \Phi^T \left(M(\dot{V} + \dot{V}_g) - (M\tilde{\Omega} + \tilde{V}^T M)V \right)$ $u = H^T f$
$\langle B \rangle$	$\overset{\circ}{V} = \Phi \left(Hq_a - \tilde{\Delta}V \right)$ $f = \Phi^T \left(M(\overset{\circ}{V} + \dot{V}_g) - \tilde{V}^T M V \right)$ $u = H^T f$
$\langle A \rangle$	$\dot{V} = \Phi \left(Hq_a - \tilde{\Delta}V \right)$ $f = \Phi^T \left(M(\dot{V} + \dot{V}_g) - \tilde{V}^T M V \right)$ $u = H^T f$

3.1.3 Forward dynamics

Forward or direct dynamics denotes in robotics the calculation of joint accelerations q_a from given applied joint forces u and position and velocity variables q_p and q_v . It is primarily amenable to the purpose of simulation this representation is also required, e. g., for model predictive control schemes and trajectory optimization methods. Recalling the state space equation for inverse dynamics (3.1), the forward dynamics is obtained formally from inverting the positive definite joint space mass matrix \mathcal{M} :

$$\ddot{q} = \mathcal{M}(q)^{-1} (u - \mathcal{C}(q, \dot{q}) - \mathcal{G}(q)) =: \mathcal{M}(q)^{-1} \tilde{u} \quad (3.4)$$

Jain [60] reviews a large number of formalisms to solve this set of equations and categorizes the numerical schemes in three classes: (i) algorithms that require explicit computation of the mass matrix, (ii) algorithms that are completely recursive in nature, and (iii) algorithms of intermediate complexity. A comparison of the numerical efficiency of forward dynamics algorithms, especially w. r. t. coordinate representation and placement of coordinate frames can be found, e. g., in [40, 136, 90]. Numerical properties such as cancellation errors and stability have been investigated in [13, 106].

Forward dynamics based on the explicit computation of \mathcal{M} followed by the $\mathcal{O}(N^3)$ process of solving the linear matrix equation (3.1) for \ddot{q} forms the class of $\mathcal{O}(N^3)$ algorithms, most prominent are the methods developed by Walker and Orin [147], valid for MBS with prismatic and revolute joints. Here N denotes the number of bodies in one kinematic chain.

For brevity this section reformulates one $\mathcal{O}(N)$ articulated body algorithm (ABA) [39] by means of spatial operators expressions, to develop the realization of a PSOA component formulation and a dedicated protocol. The first $\mathcal{O}(N)$ formalisms have been reported by Vereshchagin [144] and Armstrong [12], more efficient formulations can be found in [21, 40, 90]. This class of algorithms is based on the idea of the articulated body [39], which is a chain of joint-connected bodies for which all applied forces are compensated by Coriolis and internal forces are assumed to be zero. This resembles a 'floppy' serial chain. Featherstone [39] observed that the applied spatial force acting on a body in a tree-structure is linear in the spatial acceleration

$$f = P\dot{V} + z,$$

here already shown in stacked notation. Where $z \in \mathbb{R}^{6N_p}$ is a bias force vector and $P \in \mathbb{R}^{6N_p \times 6N_p}$ is the stacked block-diagonal articulated body inertia matrix. Each block in P corresponds to the inertia of the

articulated body starting outboard from the body under consideration. N_p is the number of ports, N_d the number of positional degrees of freedom in the joints.

The key to the algorithm is the construction of an *explicit* inverse of \mathcal{M} , which can be expressed by means of spatial operators. Rodriguez et al. [117] discovered an alternative factorization for the mass matrix, called *innovations factorization*, which has an explicit operator inverse:

$$\mathcal{M} = (\mathbf{I} - K\Phi\mathbf{H})^\top \mathbf{D} (\mathbf{I} + K\Phi\mathbf{H}) \quad (3.5)$$

$$\mathcal{M}^{-1} = (\mathbf{I} - K\Psi\mathbf{H}) \mathbf{D}^{-1} (\mathbf{I} + K\Psi\mathbf{H})^\top \quad (3.6)$$

(3.5) can be interpreted as an LDL^\top factorization of \mathcal{M} where L is left block-triangular and D is block-diagonal. The involved spatial operators are defined by:

$$\mathbf{P} = \mathcal{E}_\psi^\top \mathbf{P} \mathcal{E}_\psi + \mathbf{M} \quad (3.7)$$

$$\mathbf{D} := \mathbf{H}^\top \mathbf{P} \mathbf{H} \quad (3.8)$$

$$\mathbf{G} := \mathbf{D}^{-1} \mathbf{H}^\top \mathbf{P} \quad (3.9)$$

$$\bar{\tau} := \mathbf{I} - \tau = \mathbf{I} - \mathbf{H} \mathbf{G} \quad (3.10)$$

$$\mathcal{E}_\psi := \bar{\tau} \mathcal{E}_\phi \quad (3.11)$$

$$\mathbf{K} := \mathbf{G} \mathcal{E}_\phi \quad (3.12)$$

$$\Psi := (\mathbf{I} - \mathcal{E}_\psi)^{-1} \quad (3.13)$$

In [66] a physical interpretation for the involved operators is given. $\mathbf{P} \in \mathbb{R}^{6N_p \times 6N_p}$ is identified by stacked articulated inertia, $\mathbf{D} \in \mathbb{R}^{N_d \times N_d}$ by articulated inertia about joint axes, $\mathbf{G} \in \mathbb{R}^{6N_p \times N_d}$ by Kalman gain, $\mathbf{K} \in \mathbb{R}^{6N_p \times N_d}$ by shifted Kalman gain, $\bar{\tau} \in \mathbb{R}^{6N_p \times 6N_p}$ by joint articulation operator, $\mathcal{E}_\psi \in \mathbb{R}^{6N_p \times 6N_p}$ by to-next-link articulated shift transformation, and $\Psi \in \mathbb{R}^{6N_p \times 6N_p}$ by articulated manipulator force transformation. Using (3.6) gives the symbolic computation of the forward dynamics for tree-structured systems

$$\ddot{q} = (\mathbf{I} - K\Psi\mathbf{H}) \mathbf{D}^{-1} \left\{ u - \mathbf{H}^\top \Psi^\top \left[K^\top u - \mathbf{M}(\Phi\tilde{\Delta}\mathbf{V} + \dot{\mathbf{V}}_g) - \tilde{\mathbf{V}}^\top \mathbf{M} \mathbf{V} \right] \right\} \quad (3.14)$$

It has been shown in [117, 49] that decomposing this operator expression down to component level recursions leads to a recursive algorithm comprising 3 sweeps:

(i) sweep 1 (outboard): compute

- (a) spatial velocities $\mathbf{V} = \Phi\tilde{\Delta}$,
- (b) gravitational acceleration $\dot{\mathbf{V}}_g = \mathcal{E}_\phi \dot{\mathbf{V}}_g$,
- (c) bias acceleration $a := \Phi\tilde{\Delta}\mathbf{V}$

(ii) sweep 2 (inboard): compute

- (a) local articulated inertia $\mathbf{P} = \mathcal{E}_\psi^\top \mathbf{P} \mathcal{E}_\psi + \mathbf{M}$,
- (b) portion of spatial forces independent of q_a , $b := \mathbf{M}(a + \dot{\mathbf{V}}_g) - \tilde{\mathbf{V}}^\top \mathbf{M} \mathbf{V}$,
- (c) intermediate terms $c := \Psi^\top [K^\top u - b]$, and
- (d) $d := \mathbf{D}^{-1} \{u - \mathbf{H}^\top c\}$

(iii) sweep 3 (outboard): compute

- (a) intermediate term $e := \Psi Hd$ and finally the
 (b) joint accelerations: $\ddot{q} = d - Ke$

An analysis of this equations leads to the protocol exchanged between multibody entities for the articulated body algorithm (ABA) shown in Table 3.9. This again is an extension of the kinematics protocol described in Table 3.1. The interpretation in each coordinate representation is omitted because the messages do not give access to any direct physical interpretation, except for the gravitational acceleration.

TABLE 3.9: Protocol <ABA,6D> required in spatial inverse dynamics recursions.

message symbol	sweep	explanation
a	1	bias acceleration
$\dot{V}g$	1	spatial acceleration due to gravitation
P	2	articulated body inertia
c	2	bias force
e	3	intermediate term

TABLE 3.10: Protocol <ABA,1D> required in 1D scalar forward dynamics recursion.

message symbol	sweep	explanation	semantics
f	2	joint generalized force variable	u
\dot{V}	3	joint acceleration variable	q_a

The transformations derived from the stacked notation can split up w. r. t. the components and associated to the separate multibody entities, which are shown in Table 3.11. The forward dynamics extends the forward position and velocity kinematics transformations shown in Table 3.3. The most expensive computations are the ones concerning the articulated inertia and its projections. A detailed analysis is required to decide which coordinate representation is most efficient. Considering further algorithms, e. g., contact and collision dynamics shown in the next section, the analysis becomes quite involved and depends on the concrete application. When no further computations are required the body-fixed representation again can be shown to be the most compact one [90]. Please note that for expressiveness the equations for the joint in Table 3.11 are given only for one special causality, generalization to reverse causality and other holonomic joints is straightforward [49].

3.2 Advanced and new dynamics algorithms

Completely new or extended special purpose algorithms which go beyond the standard forward and inverse dynamics for rigid robots are presented in this section. This includes some new components and physical effects which has not been captured by the default multibody entities applied in Section 3.1.

3.2.1 New elastic joint inverse dynamics

This section presents the derivation of a new recursive inverse dynamics algorithm for the class of so-called *elastic joint robots*.¹⁾ This includes new components to model the drivetrain effects and a new

¹⁾ This section extends parts of the paper [59].

TABLE 3.11: Recursive articulated body forward dynamics transformations for default MBS entities.

component:ReferenceFrame<FixedBase>			
port semantics: $\{1\} \leftarrow \{3D\}$			
msg.	sw.	symbolic transformation	
a	1	$\langle I, \langle B \rangle, \langle A \rangle$	$\mathbf{a}_1 = \mathbf{0}_6$
$\dot{\mathbf{V}}\mathbf{g}$	1	$\langle I, \langle B \rangle, \langle A \rangle$	$\dot{\mathbf{V}}\mathbf{g}_1 = \begin{pmatrix} \mathbf{0}_3 \\ \mathbf{g} \end{pmatrix}$
e	3	$\langle I, \langle B \rangle, \langle A \rangle$	$\mathbf{e}_1 = \mathbf{0}_6$

component:Displacement<Type1>			
port semantics: $\{1, 2\} \leftarrow \{3D, 3D\}$			
msg.	sw.	symbolic transformation	
		$\mathbf{a} := \begin{pmatrix} \mathbf{a}_1 \\ \mathbf{a}_2 \end{pmatrix}, \mathbf{P} = \begin{pmatrix} \mathbf{P}_1 & \\ & \mathbf{P}_2 \end{pmatrix}, \mathbf{c} := \begin{pmatrix} \mathbf{c}_1 \\ \mathbf{c}_2 \end{pmatrix}, \mathbf{e} := \begin{pmatrix} \mathbf{e}_1 \\ \mathbf{e}_2 \end{pmatrix}$	
a	1	$\langle I, \langle B \rangle, \langle A \rangle$	$\mathbf{a} = \mathbf{T}_\phi \mathbf{a}$
$\dot{\mathbf{V}}\mathbf{g}$	1	$\langle I, \langle B \rangle, \langle A \rangle$	$\dot{\mathbf{V}}\mathbf{g} = \mathbf{T}_\phi \dot{\mathbf{V}}\mathbf{g}$
P	2	$\langle I, \langle B \rangle, \langle A \rangle$	$\mathbf{P} = \mathbf{T}_\phi^T \mathbf{P} \mathbf{T}_\phi$
c	2	$\langle I, \langle B \rangle, \langle A \rangle$	$\mathbf{c} = \mathbf{T}_\phi^T \mathbf{c}$
e	3	$\langle I, \langle B \rangle, \langle A \rangle$	$\mathbf{e} = \mathbf{T}_\phi \mathbf{e}$

component: Joint<Revolute,Type1>			
port semantics: $\{1, 2, 3\} \leftarrow \{3D, 3D, 1D\}$, causality: outboard: $O = \{2\}, I = \{1, 3\}$			
msg.	sw.	symbolic transformation	
a	1	$\langle I \rangle$	$\mathbf{a}_2 = \phi_{2,1} \mathbf{a}_1 + \tilde{\Omega}_2 \mathbf{V}^2 - \phi_{2,1} (\tilde{\Omega}_1 \mathbf{V}^1)$ see 2.20
		$\langle B \rangle, \langle A \rangle$	$\mathbf{a}_2 = \phi_{2,1} \mathbf{a}_1 - \tilde{\Delta}^2 \mathbf{V}^2$
$\dot{\mathbf{V}}\mathbf{g}$	1	$\langle I, \langle B \rangle, \langle A \rangle$	$\dot{\mathbf{V}}\mathbf{g}_2 = \phi_{2,1} \dot{\mathbf{V}}\mathbf{g}_1$
	2	$\mathbf{D} := \mathbf{H}^T \mathbf{P} \mathbf{H}, \mathbf{G} := \mathbf{D}^{-1} \mathbf{H} \mathbf{P}, \bar{\tau} := \mathbf{I}_{6 \times 6} - \mathbf{H} \mathbf{G}$ $\psi_{2,1} := \bar{\tau} \phi_{2,1}, \mathbf{d} := \mathbf{D}^{-1} (\mathbf{f}_3 - \mathbf{H}^T \mathbf{c}_2)$	
P	2	$\mathbf{P}_1 = \psi_{2,1}^T \mathbf{P}_2 \psi_{2,1}$	
c	2	$\mathbf{c}_1 = \phi_{2,1}^T (\bar{\tau} \mathbf{c}_2 + \mathbf{G} \mathbf{f}_3)$	
e	3	$\mathbf{e}_2 = \psi_{2,1} \mathbf{e}_1 + \mathbf{H} \mathbf{d}$	
$\dot{\mathbf{V}}$	3	$\dot{\mathbf{V}}_3 = \mathbf{d} - \mathbf{G}^T \phi_{2,1} \mathbf{e}_1$	

component: RigidBody<1,Full,Type1>			
port semantics: $\{1\} \leftarrow \{3D\}$			
msg.	sw.	symbolic transformation	
		$\mathbf{M}_{[1]} =$ same as inverse dynamics in Table 3.6, $\tilde{\Omega}_{[1]} := \text{diag}(\mathbf{I}_{3 \times 3}, \mathbf{0}_3) \mathbf{V}_1$	
P	2	$\mathbf{P}_1 = \mathbf{M}_{[1]}$	
c	2	$\langle I \rangle$	$\mathbf{c}_1 = \mathbf{M}_{[1]} (\mathbf{a}_1 + \dot{\mathbf{V}}\mathbf{g}_1) - (\mathbf{M}_{[1]} \tilde{\Omega}_{[1]} + \dot{\mathbf{V}}_1^T \mathbf{M}_{[1]}) \mathbf{V}_1$
		$\langle B \rangle, \langle A \rangle$	$\mathbf{c}_1 = \mathbf{M}_{[1]} (\mathbf{a}_1 + \dot{\mathbf{V}}\mathbf{g}_1) - \dot{\mathbf{V}}_1^T \mathbf{M}_{[1]} \mathbf{V}_1$

method to cope with the algebraical task of differentiating the equations of motion several times and reveals several new symbolical identities which are invaluable for comparison of the method a existing approaches and for development of more advanced control strategies.

The main source of vibration in manufacturing robots and certain types of light-weight robots is due to the presence of elasticity between the actuator and the driven links [52, 32, 7]. This is caused by the deformation of transmission elements such as harmonic drives, belts, long shafts during high-speed motion, by large payloads, and/or hard contact with the environment [137]. These effects are subsumed under the term *elastic joint*. Advanced robot controllers aimed at the accurate and stable tracking of trajectories defined on basis of a model with neglected elasticity should be designed on basis of a more complete dynamics model of the robot [33].

When formulating the dynamics of an elastic robot it is convenient to use the Euler-Lagrange formalism, as shown for the cases of elastic links [19] and elastic joints [133]. This approach reveals the equations' structure without the necessity to express them explicitly. But, on the other hand, application of the Euler-Lagrange formalism for non trivial robots having many dof imposes several difficulties. While the demanding formulation of the Lagrangian can be handled with special tools one drawback is inherent: depending on the chosen coordinates the formalism results in a small system of large equations. Differentiation additionally increases their size and simplification by symbolic computational tools is still hardly tractable.

From this point of view it seems favourable to employ a recursive formalism, which leads to more compact equations and is scalable to a large number of dof. Forming the required derivatives of recursive equations is non-trivial for elastic joint robots but by means of the Spatial Operator Algebra symbolic manipulation of the recursive equations is a feasible task. Applications of the operator formulation to linearize forward and inverse dynamics of tree-structured systems by first-order derivatives of rigid MBS equations have been reported by [64, 49, 67, 132] and Murphy et al. [97] calculated the forward dynamics of a simple class of elastic joint robots.

3.2.1.1 Investigated elastic and gyroscopic effects

The robots considered in this section have fixed base, rigid links and linear elasticity in the joints. For simplicity the analysis shown is restricted to chain-structured mechanisms, though all arguments given are valid for tree-structured MBS, too. Joints and links are labelled sequentially from base to tip starting from index $i = 1$ as shown in Figure 3.1, where link i follows joint i . Index 0 denotes the robot's base. Joints are supposed to have one mechanical dof and the total number of joints and dof is N . The drives are rotatory and mounted on the links, and the motor driving joint i is located on link $i - 1$ as shown in Figure 3.1.

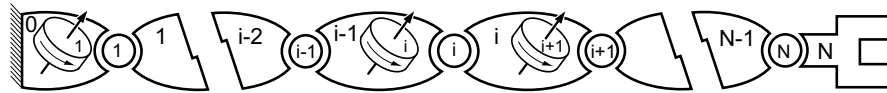


FIGURE 3.1: Schematic view of an N -joint kinematic chain where each link is a gyrostat. Small circles are joints, ellipses rigid links, tilted cylinders are rotors. Numbering ascending from base to tip.

The drivetrain actuating joint i is modeled by the rotor of drive i controlled by torque τ_i , driving an ideal gear connected to a spring which is connected to link i as shown in Figure 3.2. Dissipative effects like bearing friction and gear efficiency as well as nonlinear effects like spring stiffening, hysteresis and backlash are neglected. When the rotors of the drives show axial symmetry, they do not affect the mass distribution of the complete MBS. In consequence the robot can be viewed as a rigid MBS, where each link contains an internal source of angular momentum, i. e., each link is a *gyrostat* [149]. Hence we suppose that all inertial properties of the drivetrain are lumped with the rigid links.

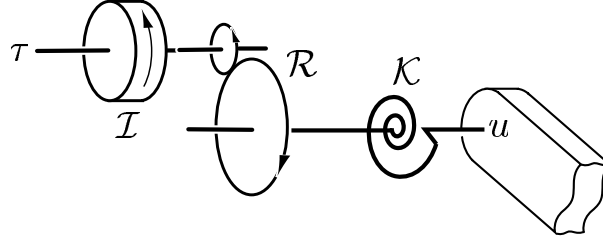


FIGURE 3.2: Schematic view of elastic drivetrain model. From left to right: rotor with inertia \mathcal{I} , gear with ratio \mathcal{R} , torsional spring with spring constant \mathcal{K} , and link.

The following section presents the intricate structure of the governing equations which are obtained from Lagrangian arguments [33]. This is to show the mathematical problems inherent in this class of robot but is not adequate to derive an efficient algorithm.

3.2.1.2 Classical Lagrangian approach

A Lagrangian derivation of the equations of motion for an elastic joint robot using independent joint variables can be found in, e. g. [33], and results in

$$\mathcal{M}(q)\ddot{q} + \mathcal{C}(q, \dot{q}) + \mathcal{G}(q) = u - u_{rotor} \quad (3.15)$$

$$-\mathcal{K}(q - \mathcal{R}\theta) = u \quad (3.16)$$

$$S(q)\ddot{\theta} + \mathcal{C}_{rotor}(q, \dot{q}, \dot{\theta}) = u_{rotor} \quad (3.17)$$

$$S(q)^T \ddot{q} + \mathcal{C}_{carrier}(q, \dot{q}) + \mathcal{I}\ddot{\theta} + \mathcal{R}u = \tau, \quad (3.18)$$

where $q \in \mathbb{R}^N$ and $\theta \in \mathbb{R}^N$ are joint and motor position variables, and $u \in \mathbb{R}^N$ and $\tau \in \mathbb{R}^N$ are generalized applied joint/motor forces. $\mathcal{M} \in \mathbb{R}^{N \times N}$ is the mass matrix, $\mathcal{C} \in \mathbb{R}^N$ the vector of Coriolis and centrifugal terms of the links, \mathcal{K} the diagonal matrix of spring constants, \mathcal{R} the diagonal matrix of gear ratios, \mathcal{G} the vector of gravitational forces, and S is the matrix of inertial couplings between links and motors. \mathcal{C}_{rotor} is due to the spatial motion of the rotors' angular momentum, $\mathcal{C}_{carrier}$ corrects for motion of the 'carrier' links and \mathcal{I} is the diagonal matrix of rotor inertias. \mathcal{R} , \mathcal{K} , and \mathcal{I} are supposed to be constant. When the drives are mounted as described above, S is upper triangular [33]:

$$S = \begin{pmatrix} 0 & S_{12}(q_1) & S_{13}(q_1, q_2) & \dots & S_{1N}(q_1, \dots, q_{N-1}) \\ 0 & 0 & S_{23}(q_2) & \dots & S_{2N}(q_2, \dots, q_{N-1}) \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ 0 & 0 & 0 & \ddots & S_{N-1N}(q_{N-1}) \\ 0 & 0 & 0 & \dots & 0 \end{pmatrix} \quad (3.19)$$

The i^{th} component of \mathcal{C}_{rotor} explicitly depends on link variables and $\theta_{i+1}, \dots, \dot{\theta}_N$

$$\mathcal{C}_{rotor} = \begin{pmatrix} \mathcal{C}_{rotor1}(q, \dot{q}, \dot{\theta}_2, \dots, \dot{\theta}_N) \\ \mathcal{C}_{rotor2}(q, \dot{q}, \dot{\theta}_3, \dots, \dot{\theta}_N) \\ \vdots \\ \mathcal{C}_{rotorN-1}(q, \dot{q}, \dot{\theta}_N) \\ \mathcal{C}_{rotorN}(q, \dot{q}) \end{pmatrix}. \quad (3.20)$$

This model is similar to the one described in [31] but includes the more general case of non-constant S and, hence a non-zero \mathcal{C}_{rotor} . Additionally it shows the dependency on the gear ratios \mathcal{R} .

For calculation of the inverse dynamics using equations (3.15)–(3.20) one starts with the N^{th} component equation of (3.15). Due to (3.19) and (3.20) this equation only depends on link variables and one is able to solve for θ_N . Differentiating this equation twice w. r. t. time gives $\dot{\theta}_N$ and $\ddot{\theta}_N$. Now one can solve the N^{th} equation of (3.18) for τ_N . Proceeding from tip to base it is possible to solve the second-last equation of (3.15) for θ_{N-1} using $\dot{\theta}_N$ and $\ddot{\theta}_N$. Again this equation needs to be differentiated twice to calculate $\dot{\theta}_{N-1}$ and $\ddot{\theta}_{N-1}$ what in turn requires $\theta_N^{(3)}$ and $\theta_N^{(4)}$ and therefore the third and fourth derivative of the N^{th} equation of (3.15). Repeating this way one completes calculating τ_1 which requires derivatives of the equations of motion up to order $\alpha = 2N$ and derivatives of the desired joint positions up to order $2(N + 1)$. Greek index α denotes the number of derivations.

This approach results in N very large algebraic equations for the motor forces which are awkward to handle and inefficient for computation. In the following sections a new non-Lagrangian approach is presented, which is amenable to efficient implementation while preserving the structure of the governing equations.

3.2.1.3 Modeling the drivetrain

One important part of robotic systems are the drivetrains driving the joints. In general a drivetrain contains motors, gears, transmissions, with characteristic phenomena such as bearing friction, damping, elasticity especially in the gears, or bearing clearance. Though being mechanical devices like links there are several simplifications possible to improve efficiency of drivetrain dynamics calculations. The idea is to exploit the rotational or translational symmetry present in many drivetrains.

A glance at Figure 3.2 shows invariance of the mass distribution w. r. t. to rotation angle. This can be used to reduce the effort for establishing a full spatial description of the governing equations to a kind of one-dimensional projection, based on scalar rotational variables and the concept of reduced moments of inertia. This is a well-known approach in the discipline of machine dynamics and applied in many commercial simulation tools, such as ADAMS [121] and SIMPACK [120]. The latter even is able transform a DAE formulation into an ODE form by symbolic transformations if this is possible. The advantages of this description are (i) an easy and efficient way to establish the equations of motion avoiding complex full spatial dynamics, (ii) easy integration of frictional and elastic effects, and (ii) avoidance of *spatial* kinematic loops—a general feature of drivetrains making the spatial formulation awkward without adding much relevant detail. In parts we follow in this section the treatment of Otter [104], who discusses an object-oriented modeling approach to simulate drivetrains in that spirit.

For the description of the elastic drivetrain investigated we first introduce new MBS entities to represent the effects of (idealized) devices, torsional elasticity, ideal gear, inertia and angular momentum of the rotor. One choice of MBS entities representing and separating these effects is

- Spring<Torsional,Linear>: a massless ideal, linear torsional spring with spring constant c between scalar ports IP:1 and IP:2.
- Gear<N,M,Revolute,Ideal>: a massless ideal gear with linear gear ratio matrix R , transmitting from N to M scalar ports. We just consider the simplest case $M = N = 1$, with a scalar ratio R relating scalar ports IP:1 and IP:2.
- RotorInertia<Type2>: a pure 'massless' angular momentum.
 - idea of Type1: A rotator with rotational symmetry around an implicitly defined axis of rotation between 'flange' scalar ports IP:1 and IP:2. The inertia around this axis is the physical property I .

- idea of Type2: Extension of Type1 by one spatial port IP:3, which is able to provide (spatial) reaction forces. The axis of rotation w. r. t. F_3 is denoted the physical property nr .

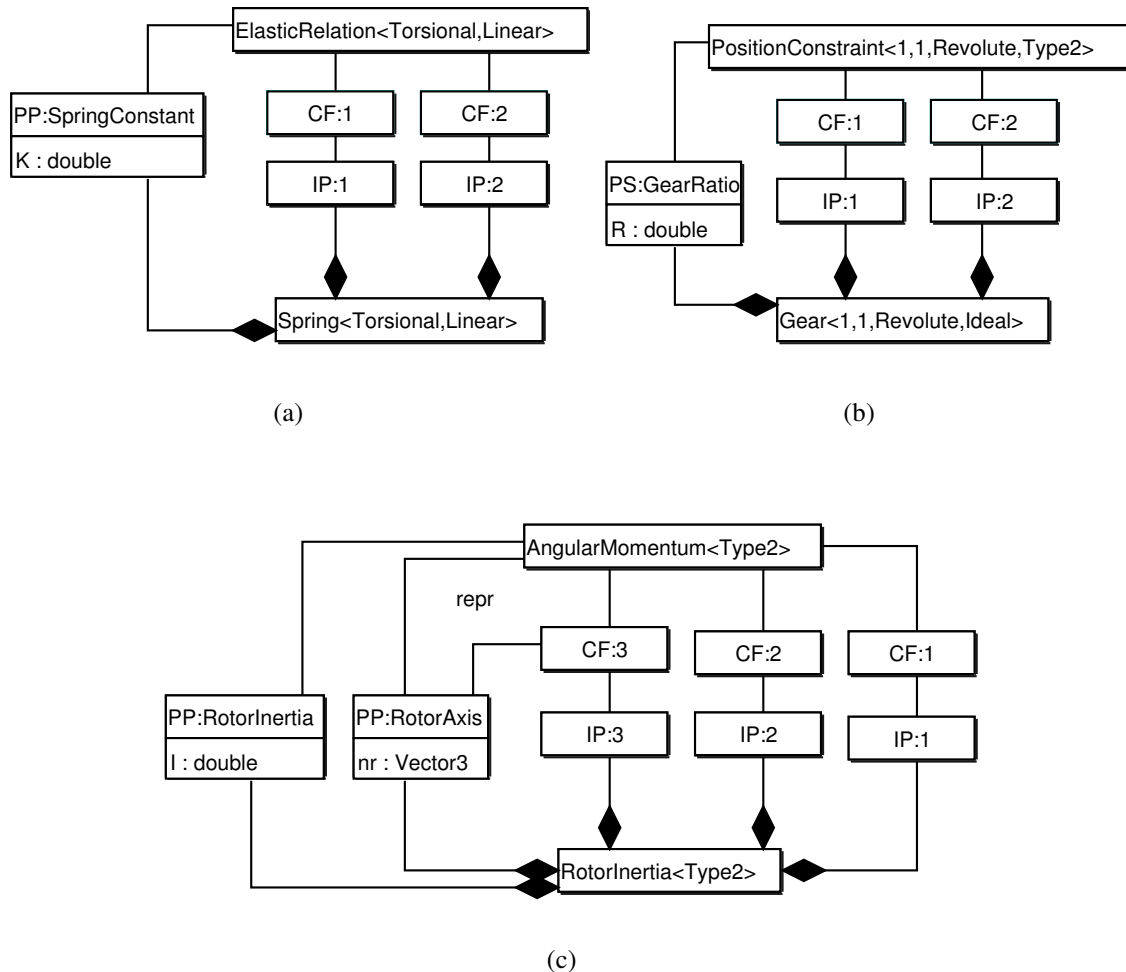


FIGURE 3.3: ER models of MBS entities (a) `Spring<Torsional,Linear>`, (b) `Gear<1,1,Revolute,Ideal>`, and (c) `RotorInertia<Type2>`.

3.2.1.4 New symbolic higher-order time derivatives of multibody equations

In this section a method is developed to calculate derivatives of the multibody equation (3.15) in a symbolic and recursive manner employing the PSOA. This comprises four steps:

- (i) establishing the inverse dynamics equations for the rotor inertia
- (ii) assembling all component equations to a closed form PSOA inverse dynamics
- (iii) differentiating the kinematical expressions (forward recursion)
- (iv) differentiating the dynamics equations (backward recursion)

It turns out that an appropriate coordinate representation of the dynamics is a key issue to obtain compact expressions for higher-order spatial derivatives. The body-fixed representation described in Section 2.1.2 presents an efficient means, because expressing dynamics w. r. t. a body-fixed frame allows for recursions based on *local* time derivatives and essential spatial operators remain constant [66]. Important to note that stacked spatial operators H , H_r , and M remain constant in body-fixed representation which simplifies derivatives w. r. t. time drastically. This leads to simpler construction of the recursions than in [109], so in the analysis below exclusively body-fixed representation will be applied and the prefix ${}^{(B)}$ is omitted for simplicity. For further use the notion of the α -times local derivative $\overset{\circ}{x}^{(\alpha)}$ of an arbitrary spatial vector x is introduced, with α denoting the number of differentiations.

Because the algorithm presented will rely on an extension of the recursive Newton-Euler the standard protocol of inverse dynamics defined in Tables 3.4 on page 39 and 3.10 on page 44 will not suffice to finally calculate the motor torques. The protocol must be able to transfer a number of higher order derivatives of the velocities and forces. This is aggravated by the fact that the maximum as well as the local order of differentiations depends on the topology of the multibody system.

The protocol to exchange data in the elastic joint inverse dynamics is shown in Table 3.12 on this page. In order to support a variable number of derivatives of spatial velocities and forces, the number of messages is not fixed, yet has to be determined from topological arguments. That will be discussed in Section 3.2.1.6. For the moment it suffices to think of an arbitrary number of messages.

TABLE 3.12: Protocol <EJRNE,6D> supporting N times derivatives of spatial forces required in 3D spatial elastic joint inverse dynamics recursions, for body-fixed representation. The column sw. denotes the involved sweep.

msg.	sw.	explanation	semantics ${}^{(B)}$
$V_{(\alpha)}$	$1.\alpha$	α^{th} derivative of spatial velocity, where $\alpha = 0, \dots, N + 1$	${}^{(B)}\overset{\circ}{V}^{(\alpha)}$
$f_{(\alpha)}$	$2.\alpha$	α^{th} derivative of spatial force, where $\alpha = 0, \dots, N$	${}^{(B)}\overset{\circ}{f}^{(\alpha)}$

TABLE 3.13: Protocol <EJRNE,1D> required in 1D scalar inverse dynamics recursion.

msg.	sw.	explanation	semantics, ${}^{(B)}$
$V_{(\alpha)}$	$1.\alpha$	α^{th} derivative of joint velocity variable, where $\alpha = 0, \dots, N + 1$	$q_a^{(\alpha)}$
$f_{(\alpha)}$	$2.\alpha$	α^{th} derivative of joint generalized force variable, where $\alpha = 0, \dots, N$	$u^{(\alpha)}$

Component equations The transformation properties of all the new ME introduced above are listed in Table 3.14. In this section we need the spatial force of the port IP:3 of the rotor inertia entity. This is simply the total time derivative of the spatial angular momentum, which stems from the rotation of the motor. When using the rotor inertia entity a new spatial operator, the rotor axis projection operator H_r , is required. It describes the direction of the angular momentum in Cartesian space. For brevity and because the equations of the torsional spring and the gear are quite simple we already state their transformations for all orders of differentiations in the same table.

The equations for the rotor inertia require some care. The rotor is mounted on any 'carrier' link. The relative internal angular momentum w. r. t. to F_3 attached to that link is $L := H_r \mathcal{I} V_{(1)_2}$, where $\mathcal{I} := I$.

So the reaction force in port IP:3 depends on the relative angular momentum, but drivetrain dynamics (3.18) depends on the *absolute* spatial angular momentum of the rotor $L_{abs} = L + \Omega_3 \mathcal{I}$. Its *absolute* time derivative projected onto its axis of rotation $H_r^T \dot{L}_{abs}$ is the required quantity to establish the torque balance of the rotor $f_{(\alpha)_2} = f_{(\alpha)_1} + H_r^T \frac{d}{dt} L_{abs}$.

TABLE 3.14: Recursive Newton Euler transformations in body-fixed representation for MBS entities required to model elastic joint robots.

component: Gear<1,1,Revolute,Ideal>		component: Spring<Torsional,Linear>	
port semantics: $\{1,2\} \leftarrow \{1D,1D\}$		port semantics: $\{1,2\} \leftarrow \{1D,1D\}$	
msg.	symbolic transformation	msg.	symbolic transformation
p	$p_2 = \frac{p_1}{R}$	p	$p_2 = p_1 + \frac{f_1}{K}$
$V_{(\alpha)}$	$V_{(\alpha)_2} = \frac{V_{(\alpha)_1}}{R}$	$V_{(\alpha)}$	$V_{(\alpha)_2} = V_{(\alpha)_1} + \frac{f_{(\alpha)_1}}{K}$
$f_{(\alpha)}$	$f_{(\alpha)_1} = \frac{f_{(\alpha)_2}}{R}$	$f_{(\alpha)}$	$f_{(\alpha)_1} = -f_{(\alpha)_2}$

component: RotorInertia<Type2>	
port semantics: $\{1,2,3\} \leftarrow \{1D,1D,3D\}$	
msg.	symbolic transformation
p	$p_1 = p_2$
$V_{(\alpha)}$	$V_{(\alpha)_1} = V_{(\alpha)_2}$
$f_{(\alpha)}$	$f_{(0)_3} = \frac{d}{dt} L = \dot{L} + \tilde{\Omega}_3 L = \dot{L} + \tilde{V}_3 L = \mathcal{I}(H_r V_{(2)_2} + \tilde{V}_3 H_r V_{(1)_2})$ $f_{(\alpha)_2} = H_r^T \frac{d}{dt} L_{abs} = f_{(\alpha)_1} + V_{(2)_2} \mathcal{I} + \frac{d}{dt} \Omega_3 \mathcal{I}$

Expressions for the complete system The model of the elastic joint robot is sketched in component oriented form in Figure 3.4 as an MCD. The robot base is RS and each dashed box contains one elastic joint and rigid link. The drivetrain with index i consists of the sequence ST $_i$ –G $_i$ –R $_i$ showing that drive i is mounted on link $i - 1$. The component BN is the outermost link including a payload.

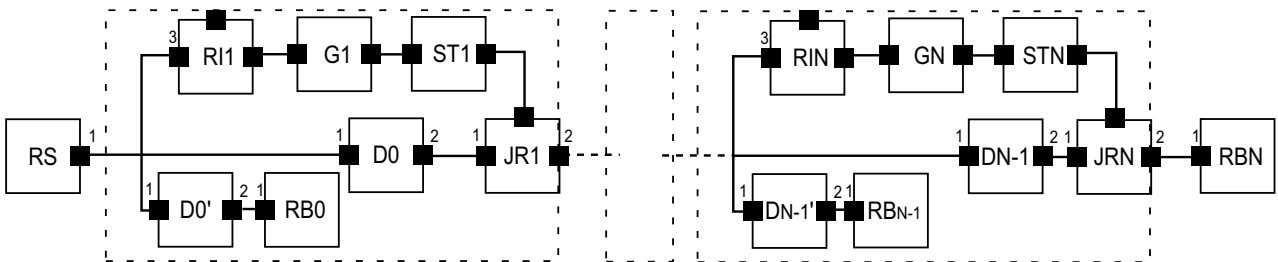


FIGURE 3.4: Multibody component diagram depicting the first and the last link of the elastic joint robot chain (dashed boxes). Link i being a gyrostat is here synthesized from the four components displacement D[i-1] and D[i-1]', rigid body RB[i-1], and rotor angular momentum RI[i].

The component causality is defined that one gets simple outboard and inboard sweeps. For outboard sweep this leads to output set $O = \{RS.IP:1, D_i.IP:2, JR_j.IP:2\}$ and the corresponding inport set results in $I = \{D_i.IP:1, JR_j.IP:1, RB_i.IP:1, RI_j\}$ where $i = \{0, \dots, N - 1\}$ and $j = \{1, \dots, N\}$. Assembling

the component relations leads to the inverse dynamics in body-fixed representation:

$$V = \Phi \Delta \quad (3.21)$$

$$\overset{\circ}{V} = \Phi(H\ddot{q} - \tilde{\Delta}V) \quad (3.22)$$

$$f = \Phi^T \{M\overset{\circ}{V} - \tilde{V}^T M V + H_r \mathcal{I} \ddot{\theta} + \tilde{\Omega} H_r \mathcal{I} \dot{\theta}\} \quad (3.23)$$

$$u = H^T f. \quad (3.24)$$

This is identical to the expressions shown in Table 3.8 except for two extra terms in the calculation of the spatial forces f . These account for the additional gyroscopic effects of the rotors. Gravitational acceleration is considered by accelerating the reference system so matrix $\overset{\circ}{V}_g$ can be omitted for simplicity.

Restricting the symbolic expressions to the sub-set $E = \{JR1.IP : 2, \dots, JRN.IP : 2\}$ (ports IP:2 of all revolute joints) allows to concentrate on one important structural characteristic of this system. The joint projection operator is diagonal $H|_E = \text{diag}(H_{[JR1.IP:2]}, \dots, H_{[JRN.IP:2]})$. Now $H_r|_E$ shows on which link each motor is mounted. In case rotor i is mounted on link $i - 1$ the blocks $[i - 1, i]$ are non-zero and the operator writes

$$H_r|_E := \begin{pmatrix} 0_6 & H_{r[RJ1.IP:3]} & & & & \\ & \ddots & \ddots & & & \\ & & \ddots & \ddots & & \\ & & & \ddots & H_{r[RN-1.IP:3]} & \\ & & & & & 0_6 \end{pmatrix}, \quad (3.25)$$

i. e., changing angular momentum of rotor Rli contributes to torque of the joint JRli-1. The next step is now to establish a recursion w. r. t. differentiation order α for the standard recursive equations w. r. t. link index i analogously to equations (3.21)–(3.24). The goal is to arrive at an algorithm which is amenable to an efficient and straightforward implementation.

Kinematics derivatives When restricting to the rigid MBS model described by (3.1) the calculation of the α -times derivative of the dynamics requires derivatives of \mathcal{M} and \mathcal{C} up to an order of α . As a consequence the spatial velocity V has to be differentiated for $\alpha + 1$ times. In absence of collision and contact it is possible to obtain derivatives of the equations of motion of arbitrary order which is due to the structure of the underlying smooth kinematics and dynamics equations [98]. To obtain a recursion w. r. t. α we assume the α -times local derivative of V obeys the identity

$$\overset{\circ}{V}^{(\alpha)} = \Phi (A_\alpha + B_\alpha). \quad (3.26)$$

From (3.21) follow

$$A_0 = \Delta = Hq^{(1)} \quad \text{and} \quad B_0 = 0 \quad (3.27)$$

which are needed to start the recursion. Differentiating (3.26) once locally w. r. t. time and using operator identities for the time derivatives of \mathcal{E}_ϕ and $\overset{\circ}{\Phi}$ found in [66]

$$\overset{\circ}{\mathcal{E}}_\phi = -\tilde{\Delta} \mathcal{E}_\phi \quad (3.28)$$

$$\overset{\circ}{\Phi} = -\Phi \tilde{\Delta} \mathcal{E}_\phi \Phi \quad (3.29)$$

leads to

$$\overset{\circ}{V}^{(\alpha+1)} = \Phi \left(\overset{\circ}{A}_\alpha - \tilde{\Delta} \mathcal{E}_\phi \overset{\circ}{V}^{(\alpha)} + \overset{\circ}{B}_\alpha \right) \quad (3.30)$$

which is advantageous, because between the brackets there is no Φ , avoiding explicit higher derivatives of Φ . The equivalences

$$A_{\alpha+1} \equiv \overset{\circ}{A}_\alpha = Hq^{(\alpha+2)} \quad (3.31)$$

$$B_{\alpha+1} \equiv -\tilde{\Delta}\mathcal{E}_\phi\overset{\circ}{V}^{(\alpha)} + \overset{\circ}{B}_\alpha \quad (3.32)$$

formally lead to the desired form of equation (3.26). By construction B_α always fulfills

$$B_\alpha = \sum_i b_{\alpha,i}$$

where each summand is a product

$$b_{\alpha,i} := k_i \prod_{j \in J_i} [\tilde{\Delta}^{(\alpha_j)}] \mathcal{E}_\phi \overset{\circ}{V}^{(\alpha_i)}$$

with k_i scalar and $J_j \subset \{0, 1, \dots, \alpha - 1\}$. This follows from identities (3.27) and (3.28).

Dynamics derivatives If motor j is mounted on link i all of its inertia properties $M_{r[j]}$ can be added to the link resulting in a total $M_{[i]}$. The rotor angular momentum *relative* to the link $L_i := \mathcal{I}_j n_{r,j} \dot{\theta}_j$ has to be considered separately in order to calculate the correct dynamics. Hereby $n_{r,j}$ is the rotor axis of rotation and \mathcal{I}_j the inertia about $n_{r,j}$. The rotor spatial angular momentum hence is

$$L_{[i]} := H_{r[i,j]} \mathcal{I}_j \dot{\theta}_j = \begin{pmatrix} L_i \\ 0_3 \end{pmatrix}. \quad (3.33)$$

A change in angular momentum

$$\frac{d}{dt} L_{[i]} = \overset{\circ}{L}_{[i]} + \tilde{\Omega}_{[i]} L_{[i]} = \mathcal{I}_i (H_{r[i,j]} \ddot{\theta}_j + \tilde{\Omega}_{[i]} H_{r[i,j]} \dot{\theta}_j) \quad (3.34)$$

expressed conveniently with circle derivative and stacked representation

$$\dot{L} = \overset{\circ}{L} + \tilde{\Omega} L$$

causes a torque which contributes to the dynamics. There is no need to apply a rigid body transformation from the center of rotor to the center of link, because this pure torque is independent of the point of application. When calculating the MBS dynamics the contribution of each link to the spatial force

$$f_\delta := M\overset{\circ}{V} + \overset{\circ}{L} - \tilde{V}^T M V + \tilde{\Omega} L \quad (3.35)$$

allows for a compact expression of the spatial recursion (3.23), $f = \Phi^T f_\delta$. Analogously to (3.26) one again can assume

$$\overset{\circ}{f}^{(\alpha)} = \Phi^T (C_\alpha + D_\alpha) \quad (3.36)$$

and for $\alpha = 0$ follows

$$C_0 = \overset{\circ}{f}_\delta^{(0)} \quad \text{and} \quad D_0 = 0.$$

Differentiating (3.35) α -times and using a binomial expansion one arrives at a compact non-recursive

$$\overset{\circ}{f}_\delta^{(\alpha)} = M\overset{\circ}{V}^{(\alpha+1)} + \overset{\circ}{L}^{(\alpha+1)} + \sum_{j=0}^{\alpha} \binom{\alpha}{j} \left[-\tilde{V}^{(j)T} M\overset{\circ}{V}^{(\alpha-j)} + \tilde{\Omega}^{(j)} \overset{\circ}{L}^{(\alpha-j)} \right]. \quad (3.37)$$

Differentiating (3.36) w. r. t. time

$$\overset{\circ}{f}^{(\alpha+1)} = \Phi^T \left(\overset{\circ}{C}_\alpha - \mathcal{E}_\phi^T \tilde{\Delta}^T \overset{\circ}{f}^{(\alpha)} + \overset{\circ}{D}_\alpha \right)$$

again leads to a symbolic recursion when identifying

$$\begin{aligned} C_{\alpha+1} &\equiv \overset{\circ}{C}_\alpha = \overset{\circ}{f}_\delta^{(\alpha+1)} \\ D_{\alpha+1} &\equiv \overset{\circ}{D}_\alpha - \mathcal{E}_\phi^T \tilde{\Delta}^T \overset{\circ}{f}^{(\alpha)}. \end{aligned}$$

3.2.1.5 Combining elastic drivetrain and rigid multibody models leading to new operator expressions

A strategy for solution of the complete inverse dynamics problem, i. e., the multibody dynamics coupled to the drivetrain dynamics, becomes clear when expressions from (3.15)–(3.18) are identified in (3.21)–(3.24). Combining the latter together and using (3.34) gives

$$u = \mathbf{H}^T \Phi^T \mathbf{M} \Phi \mathbf{H} \ddot{q} \quad (3.38)$$

$$+ \mathbf{H}^T \Phi^T (-\mathbf{M} \Phi \tilde{\Delta} - \tilde{\mathbf{V}}^T \mathbf{M}) \mathbf{V} \quad (3.39)$$

$$+ \mathbf{H}^T \Phi^T \mathbf{H}_r \mathcal{I} \ddot{\theta} + \mathbf{H}^T \Phi^T \tilde{\Omega} \mathbf{H}_r \mathcal{I} \dot{\theta} \quad (3.40)$$

where (3.38) and (3.39) are standard factorizations of \mathcal{M} and \mathcal{C} [64]. Comparing expression (3.40) with (3.17) leads to the important operator factorizations of S and \mathcal{C}_{rotor} :

$$S(q) = \mathbf{H}^T \Phi^T \mathbf{H}_r \mathcal{I} \quad (3.41)$$

$$\mathcal{C}_{rotor}(q, \dot{q}, \dot{\theta}) = \mathbf{H}^T \Phi^T \tilde{\Omega} \mathbf{H}_r \mathcal{I} \dot{\theta}. \quad (3.42)$$

Equation (3.41) restates (3.19) in an *explicit* manner and it obviously results in the upper triangular shape derived in (3.19) from pure structural arguments. An arbitrary axis of rotation n_i is invariant under its generated rotation, ${}^i\mathbf{R}_{i-1} n_i = n_i$. It follows that

$$\phi_{i,i-1}(q_i) \mathbf{H}_{[i]} = \begin{pmatrix} n_i \\ -{}^i\tilde{r}_{i-1,i} n_i \end{pmatrix} \quad (3.43)$$

is independent of q_i . This observation shows that row i of the operator product $\mathbf{H}^T \Phi^T$ in (3.41) is independent of q_i , thus revealing one more structural property of $S(q)$:

$$S_{i,j} = S(q_{i+1}, \dots, q_{j-1})_{i,j}, \quad 1 < i < j < N. \quad (3.44)$$

Drivetrain dynamics (3.18) depends on the *absolute* spatial angular momentum of rotor j mounted on link i

$$L_{abs[i]} = L_{[i]} + \Omega_{[i]} \mathcal{I}_j.$$

Its *absolute* time derivative projected onto its axis of rotation using (3.22) is in case of revolute joints

$$\mathbf{H}_r^T \dot{L}_{abs} = \mathcal{I} \ddot{\theta} + \mathbf{H}_r^T \Phi (\mathbf{H} \ddot{q} - \tilde{\Delta} \mathbf{V}) \mathcal{I}.$$

Now it is possible to establish the torque balance of the rotors

$$\tau = \mathcal{R}u + \mathcal{I} \ddot{\theta} + \mathbf{H}_r^T \Phi \mathbf{H} \ddot{q} \mathcal{I} - \mathbf{H}_r^T \Phi \tilde{\Delta} \mathbf{V} \mathcal{I}. \quad (3.45)$$

This allows to identify

$$\mathcal{C}_{carrier}(q, \dot{q}) = -H_r^T \Phi \tilde{\Delta} \nabla \mathcal{I}, \quad (3.46)$$

using (3.18) and (3.41) and reveals

$$\mathcal{C}_{carrier}(q, \dot{q}) + S(q)^T \ddot{q} = H_r^T \dot{\Omega} \mathcal{I}. \quad (3.47)$$

These were the last missing identities to express equations (3.15)-(3.18) completely in closed form by means of spatial operators. The algorithm presented in the following section is based on these symbolic operator factorizations.

3.2.1.6 Numerical computation of the inverse dynamics

From operator expressions (3.26) and (3.36) one concludes that the recursive algorithm will comprise two sweeps, one outboard sweep to calculate velocities and all required higher order derivatives followed by one inboard sweep to do force and force derivative computations for rigid body and drivetrain parts. The arguments discussed in Section 3.2.1.2 show how the structural properties of H_r and S modify the simple two sweep execution logic of rigid MBS inverse dynamics. This leads to Algorithm 1 when restricting the evaluation to the port set E .

Algorithm 1 Elastic joint inverse dynamics algorithm

Detect permanent zeros of $S(q)$ from MBS topology

Require: $S(q)$ is upper triangular

Calc. max. order $\alpha_{max}[i]$ for each u_i from (3.17)

$$\alpha_{max} = \max_{i=1, \dots, N} \alpha_{max}[i]$$

$$\overset{\circ}{V}^{(\alpha)}_{[0]} = 0_6, \quad \alpha \in \{0, \dots, \alpha_{max} + 1\}$$

for $i=1$ to N **do** {Outboard sweep}

for $\alpha=0$ to $\alpha_{max}+1$ **do**

$$\overset{\circ}{V}^{(\alpha)}_{[i]} = \phi_{i,i-1} \overset{\circ}{V}^{(\alpha)}_{[i-1]} + A_{\alpha[i]} + B_{\alpha[i]}$$

end for

end for

$$\overset{\circ}{f}^{(\alpha)}_{[N+1]} = 0_6, \quad \alpha \in \{0, \dots, \alpha_{max}\}$$

for $i=N$ to 1 **do** {Inboard sweep}

for $\alpha=0$ to $\alpha_{max}[i]$ **do**

$$\overset{\circ}{f}^{(\alpha)}_{[i]} = \phi_{i+1,i}^T \overset{\circ}{f}^{(\alpha)}_{[i+1]} + C_{\alpha[i]} + D_{\alpha[i]}$$

$$\theta_i^{(\alpha)} = \left(\frac{u_i^{(\alpha)}}{\mathcal{K}_{i,i}} + q^{(\alpha)} \right) \frac{1}{\mathcal{R}_{i,i}}$$

end for

$$\tau_i = \mathcal{R}_{i,i} u_i + \mathcal{I}_{i,i} \ddot{\theta}_i + H_r^T_{[i,i]} \dot{\Omega}_{[i]} \mathcal{I}_{i,i}$$

end for

It is important to note, that the required number of differentiations of the dynamics decreases from tip to base. The maximum order α_{max} is required just for the outermost link.

Concerning code generation it is worthwhile to note a great potential for optimization. H_r is sparse when each $H_r[i]$ is a unit vector in local coordinates and $\widetilde{H_r[i]} \widetilde{H_r[i]} = I_{6 \times 6}$ results in simple expressions for B_{α} needed in (3.26), even for large α . Loss in numerical precision might occur likewise for all evaluations of higher-order Taylor expansions. The explicitness of spatial operator expressions helps in taking precautions against that problem.

The complexity of the Newton-Euler inverse dynamics for a rigid robot is $\mathcal{O}(N)$. The maximum possible number of differentiations is $2(N + 1)$ as discussed in Section 3.2.1.2. The complexity of expressions (3.26) and (3.36) grows linearly in α . Analysis of the nested loops in outboard and inboard sweeps in Algorithm 1 shows that without any further symbolic simplifications this algorithm is at the worst $\mathcal{O}(N^3)$.

The elastic joint robot model is a first step towards a more realistic description of robot dynamics. For industrial robots with very stiff links the rigid body approximation is rather good and gear elasticity is by far the most dominant source of compliance [137]. Even when links are elastic an elastic joint model serves as a first order approximation, however, as kind of 'lumping' the elasticity in the joints. This is exploited for the calibration model in Section 5.3.

Systems where link elasticity can not be neglected are called *elastic multibody systems*. The dynamics of elastic bodies is considered for instance by means of finite element methods. An excellent basis for the task of formulating elastic bodies in terms of a MBS entities is Chapter 6.4. in the book of Schwertassek and Wallrapp [125] which describes the involved data and its meaning. This data is obtained from finite element or continuum models and preprocessed for usage in MBS codes via a standardized description, which has been investigated by Wallrapp [148]. There are several specialized algorithms for elastic link robots. Inverse and forward dynamics for chain-structured robots with fixed base is presented in [75, 20]. An approach using the spatial operator algebra and Ritz functions to establish dynamics equations for robots with elastic links is presented in [62] leading to an algorithm of $\mathcal{O}(N)$ complexity. It can be directly applied within the PSOA framework, but this is beyond the scope of this work.

3.2.2 Obtaining sensitivity information

Calculation of derivatives of the robot kinematics and dynamics is essential in solving problems involving numerical optimization, non-linear analysis, parameter identification and calibration, e. g., see [15, 98]. Kinematics derivatives appear naturally in the form of velocity kinematics, but are generally required for mappings between Cartesian and joint space. Applications are interpretations of rate sensor data and, e. g., so-called Jacobian transposed control schemes [134]. Forward and inverse dynamics models for robots linearized w. r. t. state and control variables are useful in motion planning and control applications [99, 64, 67].

3.2.2.1 Kinematics derivative using pseudo-velocities

The *manipulator Jacobian* $\mathcal{J}(q)$ [134] relates joint rates to the endeffector motion. It states the linear relation between joint velocities and the spatial velocity of a frame F_1 , called *endeffector* frame, which is fixed in the model w. r. t. a reference frame F_0 . The definitions found in literature differ in the way the angular end-effector motion is parametrized. One possible definition which employs the (physical) angular velocity of the end-effector of an N_d dof manipulator is

$$\begin{pmatrix} {}^1\omega_{0,1} \\ {}^1v_{0,1} \end{pmatrix} = \mathcal{J}(q_p)^{0,1} \cdot q_v. \quad (3.48)$$

Here q_p and q_v denote the vectors $\in \mathbb{R}^{N_d}$ of joint position and velocity state variables and $\mathcal{J}(q_p)^{0,1} \in \mathbb{R}^{6 \times N_d}$. Even when using a Jacobian-free formulation of the velocity kinematics, \mathcal{J} can be computed analytically without any discretization errors. Relying on an existing velocity kinematics $V_1 = V_1(q_p, q_v)$ for the end-effector frame and applying unit pseudo-velocities $q_v = e_i$ results in column i of the Jacobian:

$$\mathcal{J}(q_p)_i^{0,1} = V_1(q_p, e_i).$$

This method known from the dynamics algorithms 1 and 2 in Walker and Orin [147] is nearly as efficient as dedicated algorithms used for kinematic chains [30], but applicable to more general topologies such as closed-chained robots as shown by Kecskeméthy [74]. The only prerequisite required is the existence of the velocity kinematics function $V_1 = V_1(q_p, q_v)$, naturally present in dynamics schemes.

Forward kinematics does not only depend on joint positions but on physical properties $\lambda \in \mathbb{R}^m$ of the robot, i. e., $V_1 = V_1(q_p, q_v, \lambda)$. Calibration is a process to identify the real values of uncertain or changing physical parameters of the robot such as link-lengths. Some employed numerical techniques require the sensitivity of the end-effector position of the manipulator in terms of these physical parameters λ . The method used for (3.48) can be applied to compute a generalized Jacobian matrix $\mathcal{J}(q_p, \lambda)$ with respect to kinematical physical parameters $\lambda := (\lambda_1, \dots, \lambda_m)$. For instance the components λ_i may be tilt angles of the joint axes. The term 'kinematical' denotes the nature of parameters which qualify for this method: parameters which influence directly the forward kinematics function. This Jacobian is defined as

$$\mathcal{J}(q_p, \lambda)^{0,1} := \begin{pmatrix} \rho_1 & \cdots & \rho_m \\ \chi_1 & \cdots & \chi_m \end{pmatrix} \in R^{6 \times m},$$

where

$$\rho_i := \frac{\partial^1 r_{0,1}}{\partial \lambda_i} \quad \text{and} \quad \frac{\partial^0 R_1}{\partial \lambda_i} = -\tilde{\chi}_i \cdot {}^0 R_1.$$

This approach applies to all physical kinematic parameters, as long as the transformation of the multibody entity provides a pseudo-velocity with respect to λ_i . The abstract robot description provides an elegant means for re-interpreting a given model during equation code generation. For instance a pure displacement would be mapped to a chained ensemble of three prismatic joints providing the required velocity transformations without any symbolic processing.

3.2.2.2 Dynamics sensitivities

This section sketches symbolical methods and a recursive algorithm to obtain derivatives directly from multibody equations by differentiating them symbolically.

Calculation of sensitivities in Section 3.2.2.1 is restricted to parameters providing naturally a velocity or pseudo-velocity kinematics transformation. This method does not apply for differentiation w. r. t. applied joint accelerations and actions, because they usually do not appear in the kinematics or inverse dynamics equations of rigid MBS in differentiated form. The objectives in trajectory optimization, see Section 5.4, to apply these kind of sensitivities are (i) more robust and faster convergence in gradient based methods and (ii) more reliable approximation of the Hessian from the exact analytical first derivative. In contrast to numerical differentiation this technique is robust, avoids errors, and is scalable to large-dimensional systems such as full three dimensional humanoid models with more than 50 dof.

The sensitivities of inverse dynamics δu and forward dynamics $\delta \ddot{q}$ w. r. t. position, velocity, and control variables for tree-structured rigid MBS are :

$$\delta u = \nabla_{q_p} \delta q + \nabla_{q_v} \delta \dot{q} + \nabla_{q_a} \delta \ddot{q} \quad (3.49)$$

$$\delta \ddot{q} = \nabla_u \ddot{q} \delta u + \nabla_q \ddot{q} \delta q + \nabla_{\dot{q}} \ddot{q} \delta \dot{q} \quad (3.50)$$

Starting from the closed form spatial operator expressions for the inverse dynamics (3.2) and the forward dynamics (3.14) the linearized models are obtained from special operator identities. The following steps are restricted to absolute coordinate representation for brevity, hence the index ^(A) is omitted. It has been

shown in [64] that closed form expressions exist for the single partial derivatives of the inverse dynamics

$$\nabla_q u = H^T \Phi^T (-\tilde{f}^\otimes H + M\tilde{V}_g \Phi H + 2\check{M}\Phi\dot{H} + M\Phi\ddot{H}) \quad (3.51)$$

$$\nabla_{\dot{q}} u = H^T \Phi^T (2\check{M}\Phi H + M\Phi\dot{H}) \quad (3.52)$$

$$\nabla_{\ddot{q}} u = H^T \Phi^T M\Phi H = \mathcal{M} \quad (3.53)$$

and the forward dynamics

$$\nabla_u \ddot{q} = \mathcal{M}^{-1} = [I - K\Psi H] D^{-1} [I - K\Psi H]^T \quad (3.54)$$

$$\begin{aligned} \nabla_q \ddot{q} &= -\mathcal{M}^{-1} \nabla_q u \\ &= -[I - K\Psi H] D^{-1} H^T \Psi^T \left(-\tilde{f}^\otimes H + M\tilde{V}_g \Phi H + 2\check{M}\Phi\dot{H} + M\Phi\ddot{H} \right) \end{aligned} \quad (3.55)$$

$$\nabla_{\dot{q}} \ddot{q} = -\mathcal{M}^{-1} \nabla_{\dot{q}} u = -[I - K\Psi H] D^{-1} H^T \Psi^T \left(2\check{M}\Phi H + M\Phi\dot{H} \right) \quad (3.56)$$

where the latter are obviously strongly related to the linearized inverse dynamics. New operators introduced are the *spatial co-cross operator*

$$\tilde{X}^\otimes := \begin{pmatrix} \tilde{c} & \tilde{d} \\ \tilde{d} & 0_{3 \times 3} \end{pmatrix} \text{ with } X := \begin{pmatrix} c \\ d \end{pmatrix} \in \mathbb{R}^6$$

and the modified first derivative of the spatial inertia

$$\check{M} := \frac{1}{2}(\dot{M} - \widetilde{M\dot{V}}^\otimes) = -\frac{1}{2}(\tilde{V}^T M + M\tilde{V} + \widetilde{M\dot{V}}^\otimes) \quad (3.57)$$

using the identity $\dot{M} = -\tilde{V}^T M - M\tilde{V}$ [49]. The matrix \tilde{f}^\otimes is the stacked block-diagonal matrix $\tilde{f}^\otimes = \text{diag}(\tilde{f}_1^\otimes, \dots, \tilde{f}_{N_p}^\otimes)$ and $\check{M} = \text{diag}(\check{M}_1, \dots, \check{M}_1)$ the stacked blockmatrix of modified inertia time derivatives.

Jain and Rodriguez [64] present a recursive algorithm which is of order $\mathcal{O}(N)$ for numerical evaluation of $\delta\ddot{q}$. Actually, it is closely related to the articulated body forward dynamics and be realized as an extension to this. The coarse structure of the algorithm is the

- (i) computing of the forward dynamics,
- (ii) determining the true spatial accelerations,
- (iii) computing the sensitivities.

The details of this algorithm are described in [64], which finally can be done in 5 sweeps starting by the three sweeps from the forward dynamics, where the second outboard sweep can be used to calculate the accelerations.

3.2.3 Inverse dynamics for closed chains

This section presents a new extension to methods for solving the inverse dynamics for robots with fixed base and including a certain class of kinematic loops. It exploits the existence of a high-level model description and relies on the assumptions that (i) the system is weakly constrained, i. e., the number of

loops is small compared to the number of joints and, (ii) there exists an analytical solution to the closure conditions. An assumption which is reasonable for many industrial robotic manipulators.

The solution of MBS that include kinematic loops is significantly more complicated than solving a tree-structured system. In mathematical terms a closed loop imposes algebraic constraints on the equations of motion, which requires the solution of a non-linear system of equations, sometimes called *closure conditions*. Many techniques exist for their general solution, e.g., Newton-Raphson methods. However, these are all iterative and have a number of inherent drawbacks: (i) the supply of initial values, (ii) a varying number of iterations, (iii) no guarantee of convergence, and (iv) nonunique solutions. As a consequence, it is difficult to apply iterative methods in on-line tasks with fixed timing, especially in a manufacturing system, where robustness and safety are paramount.

3.2.3.1 Automatic processing of partial kinematic loops²⁾

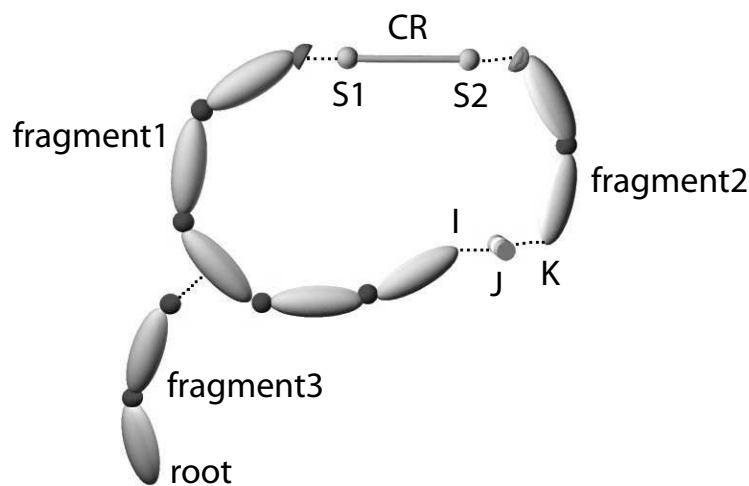


FIGURE 3.5: A kinematic loop partitioned for explicit solution. The generic loop consists of rigid bodies (ellipses), arbitrary joints (balls), one connecting rod (CR), and one free revolute joint (J).

The problems inherent to an all purpose loop treatment can be mitigated by enforcing a restriction to a special class of loops, which allows an *analytic* and hence a fast and reliable solution of closure conditions. This class has been investigated by Woernle [150] especially for finding closed form solutions of the inverse kinematics of manipulators. His method exploits the observation that certain configurations of pairs of joints present in a loop lead to analytic solutions, if the loop is cut at this joint positions. The method hence starts conceptually from a spanning tree of the MBS and 'closes' the branches by joints. For the sake of simplicity this presentation follows the idea of Möller [96] to close the loop by a special MBS entity, a *connecting rod*. Its mechanical interpretation is a massless rigid link with a spherical joint at each end, as illustrated in Figure 3.5.

In kinematic terms the connecting rod imposes one additional constraint, i. e., a constant distance between two coordinate frames, which in turn reduces the number of dof in the loop by one. Therefore, one joint position variable depends on the closure condition. If this variable belongs to a single dof joint, called *free joint*, as depicted in Figure 3.5 for the revolute joint J, then there exists the desired analytical closed-form solution.

The loop preprocessing requires three steps, to (i) detect loop presence and topology, (ii) define a spanning tree, (iii) extract the closing parts, and (iv) detect or define the free joints. That is identical either in

²⁾ This chapter reproduces parts of the paper [57].

manual or automatic treatment. When using automatic processing the advantages of a high-level model description come to light. The presence of kinematic loops can be deduced from the model description, MBS connections and the behaviour of MBS entities. Probably there exists a kinematical coupling between more than one loop, in robotics design often used to enhance mechanical stiffness, e. g., see Section 5.2.

The potential locations to cut the loop and apply the method can be identified by the presence of ensembles formed by two ball joints and one displacement and one single dof joint, which are all not actuated, hence free. In Section 5.2 those are the two spherical joints S1 and S2 with a rigid connection, and the non-actuated revolute joint J. The actual definition of actuated or not is a model semantics detail. Possible solutions are the presence of any drivetrain actuating the joint or simply defining free joints by a tag [57].

3.2.3.2 Solving loop kinematics and dynamics

The inverse dynamics algorithm used in this work allows the efficient and reliable solution of non-linear closure conditions of loops permitting an explicit solution. It is illustrated in Figure 3.5 by a closed-chain mechanism, built from several rigid bodies, arbitrary driven joints, one free revolute joint J and one connecting rod CR. In the first step of the forward kinematics the loop is automatically partitioned into three fragments, as shown in Figure 3.5. Next, the *relative* kinematics in *fragment1* and *fragment2* computes the vector pointing from frame F_I to the center of S1 resolved in F_I , $x := {}^I r_{I,S1}$ and $y := {}^K r_{K,S2}$. Considering the type of constraint imposed by the connecting rod, it can be solved explicitly for joint variable q_J if it is a single dof joint, i. e. a revolute or a prismatic joint [96]:

$$g(q_J) := |{}^K R_I x - y|^2 - |r_{S1,S2}|^2 = 0, \quad (3.58)$$

where ${}^K R_I(q_J)$ is the transformation matrix from frame F_I to F_K . In case of a revolute joint ${}^K R_I(q_J)$ is a rotation about the z-axis, and q_J can be determined by the solution of

$$A \cos q_J + B \sin q_J + C = 0$$

whereas

$$\begin{aligned} A &:= -2 \cdot (x_1 y_1 + x_2 y_2) \\ B &:= 2 \cdot (x_2 y_1 - x_1 y_2) \\ C &:= x^\top x + y^\top y - 2 \cdot u_3^\top x u_3^\top y - r_{S1,S2}^\top r_{S1,S2} \end{aligned}$$

In case of a prismatic joint the expressions can be derived in an analogous manner. Note that the solution is not unique. To choose one solution automatically, the initial configuration of the kinematic loop in the model, given by the model specification, is analyzed. After all joint variables are known, one can employ *global* kinematics to calculate the position of all parts of the MBS.

The algorithm avoids repeated evaluation of the kinematics in parts of the model as follows:

- (i) doing global kinematics in *fragment1* and *fragment3* $\Rightarrow {}^I r_{I,S1}$.
- (ii) computing *relative* kinematics in *fragment2* $\Rightarrow {}^K r_{K,S2}$.
- (iii) solving closure condition (3.58) $\Rightarrow q_J$.
- (iv) computing global kinematics in *fragment2* including joint J.

\dot{q}_J and \ddot{q}_J can be computed analogously from \dot{g} and \ddot{g} .

On the one hand joint J is free which in turn requires the joint driving torque u_J to be zero. On the other hand, the connecting rod CR introduces an unknown constraint force λ_{CR} acting between the spherical joints $S1$ and $S2$. The method relies on the idea of applying the constraint force such that it compensates for a torque τ_{0J} introduced merely by the bodies in `fragment2` and comprises 3 steps:

- (i) computing inverse dynamics in `fragment2` without taking into account for the constraint imposed by CR . The result is the torque τ_{0J} acting in joint J .
- (ii) determining the constraint force $\lambda_{CR} = f(\tau_{0J}, {}^K r_{K,S2})$.
- (iii) computing inverse dynamics in the complete mechanism, including $CR \Rightarrow \tau_{0J} \equiv 0$.

The computation of $q_J, \dot{q}_J, \ddot{q}_J$, and λ_{CR} involves some homogenous transformations, and a second evaluation of kinematics and dynamics in `fragment2`. A computational effort even tolerable in a real-time system.

3.2.3.3 Handling coupled loops

When dealing with more complex configurations, the model topology and, therefore, the structure of the equations may be obscure. To alleviate the user and to avoid errors, the analysis is automatically performed during model setup. First a directed a-cyclic graph is created from the model description, with MBS entities as nodes and MBS connections as edges. The graph is tested for the configurations allowing for explicit solution of the closure conditions according to Woernle [150]. When coupled kinematic loops are present, a second intermediate graph structure is created showing the algebraical dependencies between the loops, in words: which joint variables appear in which closure condition. The dedicated Assignment-algorithm [107] is applied to the intermediate graph in order to determine the association of free joint variables and loops. The final step identifies strongly connected parts of the graph in order to find the correct sequence of solutions [140]. This information is sufficient to solve inverse dynamics in terms of cutting the loops, performing relative kinematics and dynamics, und computing the dynamics of the complete system.

3.2.4 Operational space dynamics

The algorithms presented in Section 3.1 were primarily based on the representation of motion in joint or minimal coordinates, mainly because these schemes are computationally efficient. Many applications in robotic manipulation involve mechanical interaction with the environment which is best described in Cartesian coordinates. Mechanical constraints result either from direct physical contact of two bodies or from an interconnection of the bodies by means of kinematically constraining mechanisms. This section introduces some methods to treat direct physical contact of a mechanism with its environment. The spatial operator formulation with its descriptor-like form helps in deriving these algorithms which are hybrid in the sense of being neither in pure state space nor in pure descriptor representation, but optimally adapted to the application.

The systems under investigation in this section are tree-structured articulated rigid multibody systems which experience mechanical contact with the environment in N_c discrete frames $F_{c,k}$, with $k = 1, \dots, N_c$,

referred to as *contact frames*. Within the presented port-based formulation this corresponds to a subset S_c of all interaction ports in the component description of the system. The more intricate case of more than one contact per body at one instance of time is not discussed here, the reader is referred to the work of Pfeiffer and Glocker [111].

Each contact frame $F_{c,k}$ introduces N_{cc_k} holonomic constraints on position level, so the total number of holonomic contact constraints is

$$N_{cc} := \sum_{k=1}^{N_c} N_{cc_k} .$$

which are described by the column vector of contact constraints $\in \mathbb{R}^{N_{cc}}$

$$c(q) = 0 . \quad (3.59)$$

Differentiating this formally w. r. t. time gives the *contact constraint equations* on velocity and acceleration level

$$\mathcal{J}_c \dot{q} = 0 \quad \mathcal{J}_c := \frac{\partial c}{\partial q} \quad (3.60)$$

$$\mathcal{J}_c \ddot{q} + \frac{d\mathcal{J}_c}{dt} \dot{q} = 0 . \quad (3.61)$$

$\mathcal{J}_c \in \mathbb{R}^{N_{cc} \times N_d}$ is called *constraint Jacobian*.

3.2.4.1 Operational space inertia

It has been shown in the fundamental paper by Khatib [79] that the crucial quantity describing the dynamics of a tree-structured articulated system in mechanical contact is the so-called *operational space inertia* $\Lambda \in \mathbb{R}^{N_{cc} \times N_{cc}}$. The most prominent example is the interaction of the end-effector of a manipulator in assembly tasks, where Λ is the effective inertia of an articulated mechanism in the endeffector frame. The inertia matrix is defined by

$$\Lambda := (\mathcal{J}_c \mathcal{M}^{-1} \mathcal{J}_c^\top)^{-1} , \quad (3.62)$$

where $\mathcal{M} \in \mathbb{R}^{N_d \times N_d}$ is the joint space mass matrix and N_d is the number of positional degrees of freedom of the mechanical system. Obviously Λ is generically invertible, because \mathcal{M} is. The stacked constraint force vector $f_c \in \mathbb{R}^{N_{cc}}$ is the force exerted back by the environment onto the system so that the contact constraint remains fulfilled.

To avoid the expensive explicit calculation of \mathcal{J}_c and \mathcal{M} one can apply a recursive algorithm first mentioned in [118] to compute Λ^{-1} . Here we recast the algorithm presented in [2, 49]. For a free flying manipulator there even exists a possibility to compute Λ directly without inversion [65]. The following derivation relies on body-fixed coordinate representation and is valid for tree-structured systems.

The *manipulator Jacobian* \mathcal{J} is defined to map joint velocities to the unconstrained stacked spatial velocity of the frames under consideration, in this case the frames from the set S_c ,

$$V_c := V_{|S_c} = \mathcal{J} \dot{q} \quad (3.63)$$

so $V_c \in \mathbb{R}^{6N_c}$. The velocity constraint equation (3.60) is equivalent to constraining certain components of the spatial velocity $V_{|S_c}$, so we may rewrite this as $QV_c = 0_{N_{cc}}$. The matrix $Q \in \mathbb{R}^{N_{cc} \times 6N_c}$ is a constant blockdiagonal matrix with $Q := \text{diag}(Q_{[1,1]}, \dots, Q_{[N_c, N_c]})$ where each block $Q_{[k,k]} \in \mathbb{R}^{N_{cc_k} \times 6}$ singles out

which components of the spatial velocity of $F_{c,k}$ are required to be zero. Examples are $Q_{[k,k]} = I_{6 \times 6}$ which represents a fixed contact because all velocity components are constrained, and

$$Q_{[k,k]} = \begin{bmatrix} 0 & & 1 & & & \\ & 0 & & 1 & & \\ & & 0 & & 1 & \\ & & & & & 1 \end{bmatrix}$$

which singles out just the translational dof of motion and represents a tip contact without slipping. It can be shown that the inverse operational inertia matrix can be factorized

$$\begin{aligned} \Lambda^{-1} &= \mathcal{J}_c \mathcal{M}^{-1} \mathcal{J}_c^\top \\ &= Q \Psi H D^{-1} H^\top \Psi^\top Q^\top, \end{aligned} \quad (3.64)$$

where Ψ is the articulated manipulator force transformation defined in (3.13). Using the definition $X := H D^{-1} H^\top$ and supposing that the blockdiagonal matrix X satisfies the identity

$$X = S - \mathcal{E}_\psi S \mathcal{E}_\psi^\top$$

one can show [3] that there exists a Y

$$\begin{aligned} Y := \Psi X \Psi^\top &= \Psi S \Psi^\top - (\Psi - I) S (\Psi - I)^\top \\ &= S \Psi^\top + \Psi S - S. \end{aligned}$$

This recursive algorithm to compute Λ^{-1} is described in [2, 49]. The first step is a three sweep computation of the block entries of matrix Y followed by a projection of Y onto the space of contact velocities using the stacked matrix Q . For the sake of brevity the reader is referred to [2] for the presentation of the complete algorithm.

3.2.4.2 Contact and collision

The method sketched here leads to a solution of the constraint equations on acceleration level, hence results in a zero tip acceleration in the constrained directions in the contact frames. The method developed in [2] assumes that the true joint accelerations are a superposition of the accelerations of the unconstrained system, referred to as the free accelerations \ddot{q}_f , and correction accelerations \ddot{q}_δ which account for contact forces treated as external forces. The equations of motion for a system experiencing constraint forces then can be written

$$\ddot{q} = \mathcal{M}(q)^{-1} (u - \mathcal{C}(q, \dot{q}) - \mathcal{G}(q) + \mathcal{J}_c^\top f_c) = \ddot{q}_f + \mathcal{M}(q)^{-1} \mathcal{J}_c^\top f_c. \quad (3.65)$$

The constraint forces in the contact frames must lead to vanishing contact accelerations which can be shown to be [2]

$$f_c = -\Lambda Q \dot{V}_c. \quad (3.66)$$

The contact algorithm is composed of the following steps:

- (i) compute the free accelerations \ddot{q}_f from standard forward dynamics
- (ii) calculate the spatial accelerations \dot{V}_c in the contact frames from forward kinematics
- (iii) compute f_c from (3.66)
- (iv) apply the resulting contact forces as external forces in the contact frames and calculate $\ddot{q}_\delta = \mathcal{M}(q)^{-1} \mathcal{J}_c^\top f_c$

(v) add \ddot{q}_δ to \ddot{q}_f to obtain the final accelerations

It should be noted that the position and velocity of the contact frames has to be known before starting the algorithm, what requires a solution the constraint condition on position and velocity level. The solution of (3.66) requires to solve a linear system of equations of dimension $\mathbb{R}^{N_{cc} \times N_{cc}}$ using standard methods such as Cholesky decomposition. The case of singular configurations of the branches is not considered here. If the rank of the constraint Jacobian does not have full rank a different solution strategy has been presented in [90].

Mechanical impact on a multibody system results in a sudden drop in total kinetic energy of the mechanical system. If the time scale of the impact process is small enough then the process can be modelled as a Dirac-shaped impulse applied to the robot in the point of interaction at one instance of time. This leads to a discontinuous jump in generalized velocities from \dot{q}^- to \dot{q}^+ . For small δt the terms containing no accelerations can be neglected in (3.65)

$$\lim_{\delta t \rightarrow 0} \int_t^{t+\delta t} \mathcal{M} \ddot{q} dt = \lim_{\delta t \rightarrow 0} \int_t^{t+\delta t} (u - \mathcal{C}(q, \dot{q}) - \mathcal{G}(q) + \mathcal{J}_c^T f_c) dt, \quad (3.67)$$

which leads to

$$\mathcal{M} \Delta \dot{q} = \mathcal{J}_c^T f_{imp}. \quad (3.68)$$

Here $\Delta \dot{q} := \dot{q}^+ - \dot{q}^-$ refers to the change in joint velocities from immediately before and after the collision event and

$$f_{imp} := \lim_{\delta t \rightarrow 0} \int_t^{t+\delta t} f_c dt \quad (3.69)$$

is the total impulse from the collision normed by the time interval δt . Let the change in spatial velocities at the contact frames be $\Delta V_c := V_c^+ - V_c^-$. Multiplying (3.68) by $\mathcal{J}_c \mathcal{M}^{-1}$ and recognizing that $Q \Delta V_c = \mathcal{J}_c \Delta \dot{q}$ results in the expression

$$f_{imp} = \Lambda Q \Delta V_c, \quad (3.70)$$

which gives the amount of impulsive 'force' required for a given drop in spatial velocity between the free and the constrained situation. Assuming completely inelastic collision the tip velocities has to be zero in the constraint directions after the collision requiring $Q V_c^+ = 0_{N_{cc}}$, and $f_{imp} = \Lambda Q V_c^-$. The joint velocities after the collision then are $\dot{q}^+ = \dot{q}^- + \Delta \dot{q}$. The algorithm can be summarized as follows [49]:

- (i) Calculate V_c from forward kinematics
- (ii) Calculate f_{imp} from (3.70)
- (iii) Calculate $\Delta \dot{q} = \mathcal{M}^{-1} \mathcal{J}_c^T f_{imp}$ treating the impulse as an external 'impulse force' in one instance of time in the contact frames
- (iv) Add the correction term $\Delta \dot{q}$ to \dot{q}^- to obtain the new correct joint velocities

Before calling the algorithm the inverse kinematics on position level has to be solved and Λ is to be computed. The complete algorithm can be shown to require a minimum of five sweeps starting with an outboard sweep. It is here omitted for brevity, for a detailed presentation the reader is referred to [2] or [49].

Chapter 4

Methodology for operational robot models

Based on a detailed analysis this chapter proposes a new object-oriented class hierarchy satisfying the requirements from the robot modeling applications leading to (i) modular, (ii) efficient, (iii) consistent, and (iv) reconfigurable characteristics.

Practical realization of software for non-linear dynamics computations of robots must consider

- (i) the complexity of the mechanical structure,
- (ii) a great variety of topologies, components, actuation methods, and demanding environmental conditions,
- (iii) types, efficiency, and interaction of available kinematics and dynamics computer algorithms suitable for such systems, and
- (iv) application scenarios reaching from off-line trajectory optimizations to real-time closed-loop control including integration and communication to external soft- and hardware with tight timing constraints.

The requirements for a software system can be captured by a system model describing what is to be realized. This model forms an abstraction in two ways [51, 126]. First, it is an abstraction from real world details which are not relevant for the intended software system. Second, it also is an abstraction from the implementation details and hence precedes an actual implementation in a programming language. The ultimate goal of the framework is to implement multibody kinematics and dynamics algorithms efficiently. So the primary task is to investigate which type of computations will be required.

All computations in a software-based system controlling a robot reflect distinct physical aspects of the *same* machine in certain states leading to a strong coupling between all parts. Therefore on conceptual and software-level there must exist a representation of this commonalities. This representation is the high-level description described in the preceding chapter which serves as a basis for the required computations and the various types of robots. Hence this approach is able to support for instance manufacturing robots, humanoids and other walking machines, and a large variety of components forming the mechanical structure, different actuators, and contact models.

Efficiency is paramount in most control applications of realistic robots, because of the complexity of calculations and the intended application in real-time systems. The governing equations for full three dimensions, the large number of degrees of freedom, and the possibly huge number of dynamics evaluations during optimization and identification runs renders this problem still challenging. In order to

TABLE 4.1: Main classes forming the specification model.

Base class	Represents specification of . . .
MBSEntSpec	MBS entities
MBSPortSpec	MBS interaction port, is part of an MBSEntSpec
MBSConnectionSpec	MBS connection
MBSModelSpec	Mechanical robot model
SolverSpec	Desired algorithm
ExteriorPortSpec	Exterior port
MBSAttributeSpec	MBS attribute, base class for all attributes which are part of an MBSEntSpec, e. g., physical properties

mitigate the hard time constraints the most efficient and robust algorithms must be chosen. Unfortunately these often are robot specific and not general purpose solutions. The most prominent and vital example in robotics is the analytical solution of the inverse kinematics problem, which gives fast and reliable solutions when compared to a general purpose approach. For integration of general purpose and problem specific algorithms, which have been automatically generated or manually coded, in one formalism a software architecture which is flexible enough is indispensable. In order to reconcile these concurrent requirements in this section a carefully designed object-oriented operational architecture is proposed. This comprises

- objects for the high-level description of the robot model and problem setup,
- objects serving as domain-specific code generators mapping the descriptions and creating
- objects serving as encapsulated algorithms with clear interfaces and semantics to improve reusability.

The intention is not to create a general purpose multibody (simulation) program, but to supply a high-level design for standardized and efficient implementation and integration of dynamics computations by robot domain specialists, who in general are no coding experts.

4.1 Classes for model specification

The benefits of having a high-level description have become obvious in the algorithms chapters. Because robotic systems and algorithms themselves are quite complex entities it is inevitable in a complex software system to use a high-level model representation that is modular and hierarchical.

The *specification model* proposed in this section comprises descriptions of the mechanical system and the type(s) of desired algorithms. This follows Hatley and Pirbhai [51] who propose a model of the requirements and the design of a system. An object-oriented representation of the specification model is presented which is adopted by the main base classes listed in Table 4.1. The relations are shown in the UML package diagram in Figure 4.1. A *package* in the UML notation is the possibility to group arbitrary model elements together. This is graphically expressed by putting the model elements inside a package symbol which is denoted by an identifier string on the flap, e. g., 'Specification Model' in Figure 4.1.

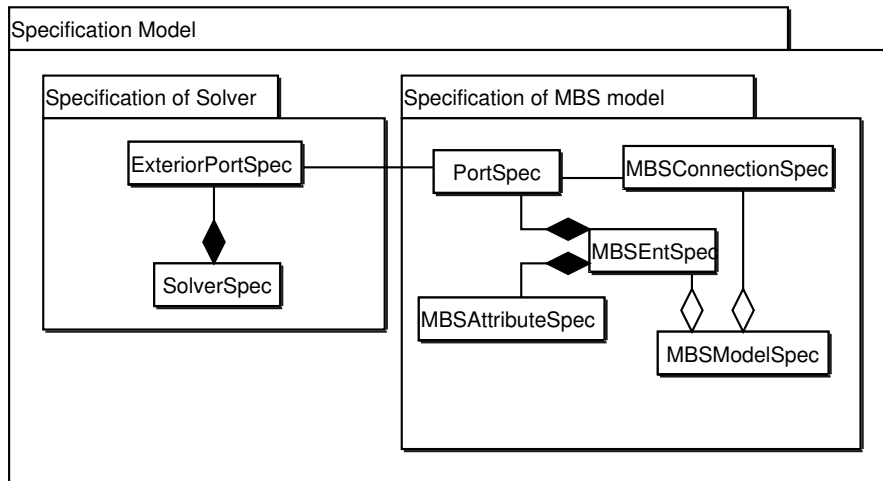


FIGURE 4.1: Conceptual UML package diagram of the main classes forming specification model.

It is important to note, however, that the specification model of the robot does *not* provide any executable code. The purpose is exclusively to decouple the dynamics model *specification* from its implementation and to form an object-oriented basis for other abstract representations such as textual or graphical representations. This additional level of abstraction is amenable to various applications. It is used in the following section to guide code generation, or to perform formal analyses, e. g., for verification or optimization of the specification w. r. t. certain criteria.

The components belonging to the multibody system domain, such as links, bodies, joints, drives, have been classified in Section 2.2 abstractly as *MBS entities*. In object-oriented terms this can be expressed by inheritance from a class `MBSEntSpec`. The domain component library contains a carefully selected, finite set of components representing concrete mechanical parts, such as drivetrains, or mathematical ideas such as contact constraints. The common base class of all attributes of an MBS entity introduced in Section 2.2, such as physical properties, physical state, etc. is the class `MBSAttributeSpec`. The topology of the mechanical model is formed by defining relations, specifications of *MBS connections*, between interaction ports belonging to the class `MBSPortSpec` and being part of each MBS entity. An MBS connection is represented by class `MBSConnectionSpec`.

The set of components and relations between ports forms an a-cyclic graph where the edges are the connections and the vertices are the MBS entities. This is the specification of the mechanical model, which is represented by class `MBSModelSpec`. There is an important semantic issue to note. Numerical schemes often are not available for a complete model, but just for parts, e. g., inverse kinematics of 6-dof manipulator mounted on a linear unit. In order to keep all computations consistent the model specification aggregates *a reference* to `MBSEntSpec` and `MBSConnectionSpec` ensuring especially that all algorithms work on the same structures and data contained in the entities. A realistic scenario where consistency becomes a vital requirement is the tuning of parameters after a calibration cycle. The new parameters must be made available to all models and parts. The `MBSModelSpec` also should reflect changes in components and topology by reconfiguration. This is crucial for robots where changing structural conditions require reconfiguration of the models, for example while walking. Hence it should be conceptually possible to abandon a connection from a given `MBSModelSpec`. This part of the dynamic semantics of the specification can be realized by removing the `MBSConnectionSpec` or by a special construct, for instance called `disconnect`.

The desired activity respectively algorithm is determined by a *solver specification* which are objects derived from class `SolverSpec`. Restricting to the domain of multibody computations these are parametrized in many cases by (i) the expected behaviour, or in other words the matrices or numerical result, (ii) the

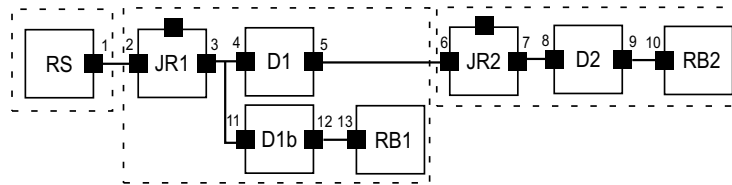


FIGURE 4.2: Multibody component diagram of a two-jointed kinematic chain. The robot base or reference system is represented by the entity RS on the left side, the rigid body payload by the entity RB2. The endeffector frame is identified F_9 which is associated to displacement port D2.IP:2.

type of algorithm applied, and (iii) further tags to denote for instance the coordinate representation used in the computations. Hence solver specifications for the algorithms investigated in this work could be expressed using a C++ template class declaration:

```
template <
typename Computation,
typename Algorithm,
typename CoordinateRepresentation
> class SolverSpec;
```

For instance a forward dynamics algorithm implemented by the articulated body algorithm using body-fixed coordinates might be described in C++ pseudo-code as follows

```
typedef SolverSpec<FwDynamics, ABA, BodyFixed> ABADynamics;
```

Stating the type of computation for a given mechanical model is not complete without further information. The considered example is the forward kinematics for the kinematics chain shown in Figure 4.2. From a mechanical viewpoint the task is uniquely defined: compute all Cartesian positions of all multibody entities (more precisely positions of the frames associated to interaction ports) from given values of position variables, i. e., a mapping from joint space to $SE(3)^{N_P}$ where N_P is the number of ports.

A 'user', for instance a path planning algorithm, might be interested only in the position of some special frames. The most prominent example in the domain of manufacturing robots is the position of the tool center (TCP) for manipulation, in Figure 4.2 the frame F_9 , or the positions of marker locations used for obstacle avoidance. In order to decouple that more *computation* specific information from the mechanical model description this requires one more tool to apply algorithm-specific port semantics, called class ExteriorPortSpec. Currently an MBS connection between ExteriorPortSpec and an interaction port has three applications. First it can be interpreted by a equation code generator to detect which numerical values are to be exposed, i. e., to be calculated explicitly. Second it is used for algorithms such as Jacobians to select the frames to be considered. The third is related to the second one and used to influence state selection (manually or by a state selection algorithm), crucial in MBS dynamics. However, this is a dynamics formalism specific topic and hence not further discussed here, the reader is referred to more detailed discussions for instance in [115].

4.2 Mapping robot models

The main task of the framework presented is to provide mappings from the conceptual 'space' of model specifications to other 'spaces'. These can be code in the shape of numerical algorithms or just textual

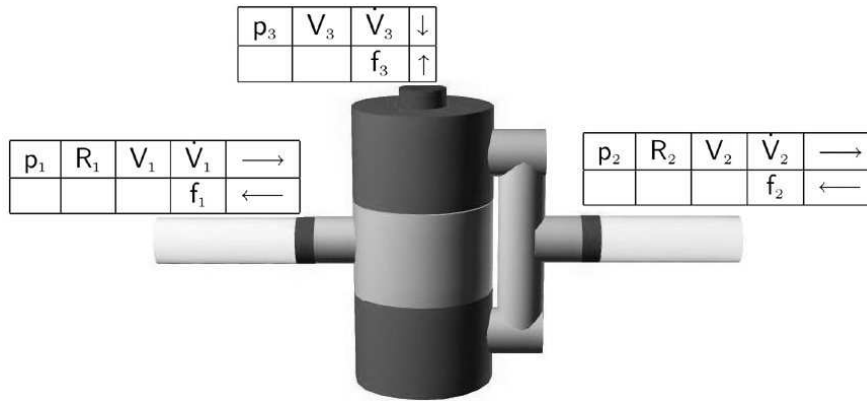


FIGURE 4.3: Sketch of dataflow transformed by the RNE implementation of `Joint<Revolute,Type1>`. Velocities and accelerations 'enter' the component in interaction ports IP:1 and IP:3 and leave it transformed on the right hand side in port IP:2. Actions go vice versa.

representations. This section proposes and discusses some of these possible mappings and issues, when such a mapping is useful and meaningful.

The notion of *formalisms* for multibody systems [115] denotes computer-oriented methods for obtaining the system matrices describing aspects of the robot, most prominently the matrices concerning the equations of motion. The domain of multibody system dynamics distinguishes two approaches to generate simulation codes, either numerical or symbolical programs. Numerical programs assemble the matrices from given model data and states. The symbolical approach provides a complete computer program, mainly in source code, to establish the equations of motion. Examples for practical realizations of both cases can be found in [123, 80].

In the domain of robot control software this approach is not satisfying for several reasons. The matrices required in robot control applications are not restricted to forward dynamics. Limited resources may prohibit a compile-to-code step. Computations are required by several control tasks maybe running on different hardware.

4.2.1 Mapping to a dataflow network

The dataflow interpretation of recursive algorithms from Section 2.3 showed the equivalence of the graph of interconnected multibody entities exchanging kinematics and dynamics protocol data to a graph computational nodes connected by dataflow edges. The basic idea of the model of computation presented in this section is to incorporate this set of local transformations in dedicated objects of class `MBSEntImpl`. These `MBSEntImpl` objects are nodes which represent computations/transformations in a suitably connected graph or network. The edges of the graph are streams of data exchanged between the nodes. This is a realization of the algorithm specific protocol, objects of class `MBSConnectionImpl`, a kind of software connector [8]. Figure 4.3 sketches the in- and outgoing messages of the revolute 'object' in the Newton-Euler recursions. The protocol for each IP is shown in the boxes. Kinematical messages are transformed from IP:1 to IP:2 and IP:3, actions the reverse direction.

This model of computation, having the characteristics of a simple synchronous dataflow process network [84], finally is a *numerical* multibody formalism, because it assembles the required matrices. It is able to map the topology as well as the properties in a probably isomorphic way. Hence it is able to carry over some characteristics of a symbolical formalism, in the sense of maintaining the structural properties of the mechanism/equations, with the potential for, e. g., optimizations.

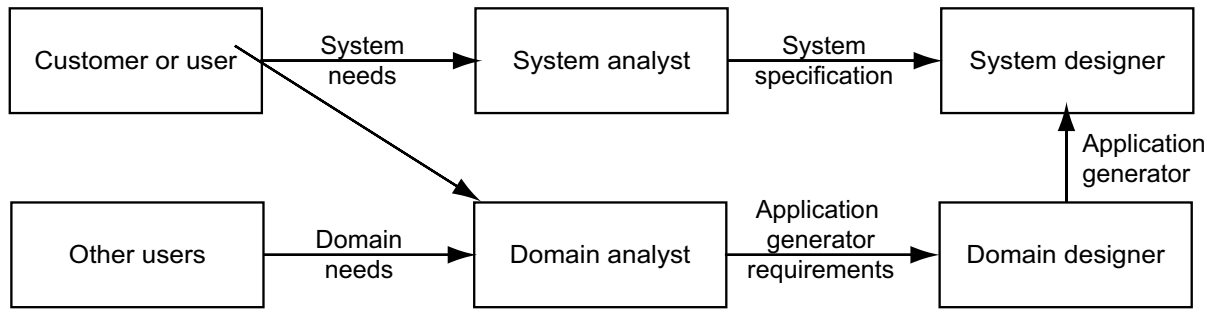


FIGURE 4.4: The relationships between roles in a design process involving an application generator [27].

Due to its appealing properties this approach has been applied for instance to the domain of digital signal processing [139]. It is well-suited for computations in embedded systems with limited resources and hard real-time constraints, as shown in the Ptolemy project [83]. Embedded code will increasingly consist of interacting software components [139], which requires the most applicable domain specific software architectures. This supports a model-based approach to program synthesis, called Model Integrated Computing [138] which exploits the ability to adapt to changing conditions even during run-time. In the following it is shown how to generate on basis of the specification model an ensemble of interacting objects forming the main parts of the computational model of the dynamics architecture.

The two main classes concerned with executable objects are the *Builder* and the *Solver*. The main purpose of a Builder is to automatically transform the specification model into an executable numerical multibody algorithm, a *Solver*. The Solvers discussed in this work are executable units of code, performing the desired computations. Each multibody algorithm has one dedicated Builder, which is able to produce Solvers implementing that algorithm. This behaviour of translating a high-level information into low-level implementation makes a Builder object a domain- and algorithm-specific *application generator* [27].

Application generators in the domain of robot control software are advantageous for several reasons, which can be analyzed best using Figure 4.4 taken from the paper of Cleaveland [27]. Figure 4.4 shows various roles (not necessarily related to persons) in a typical application-generator scenario. In a typical development effort the system analyst and the system designer build the specific applications, e. g., a trajectory planning application for a class of robots. The domain analyst and the domain designer build the application generators used by the system designer. The domain analyst specifies the generator's requirements and thus must know and understand the needs and problems of a wide range of customers. This includes knowing all current practices and whose which are likely to change in future. The domain designer takes these specifications and implements them in a generator. In robotics each concerned domain is mapping quite well to one role. The customer or user can be identified with the robot control engineer, who requires some model computations. The domain designer is the mechanical engineer or multibody expert who knows about mechanical modeling and multibody algorithms. The system designer will often be a software engineer. Because a robot control engineer is not inevitably a coding or multibody expert and vice versa, the presented framework decouples from a conceptual point of view the three domains. In this work the role of the domain analyst is further taken over.

The relations between the specification and the computational model for an application generator and a Builder are shown in 4.5. The involved classes and its purpose are shown in Figure 4.2 and are explained in more detail in the following subsections.

The transformation process of a builder without compile-to-code step is sketched as follows.

- (i) *Check*: The builder first checks if the specification model meets its requirements to create its output, a Solver. This includes:

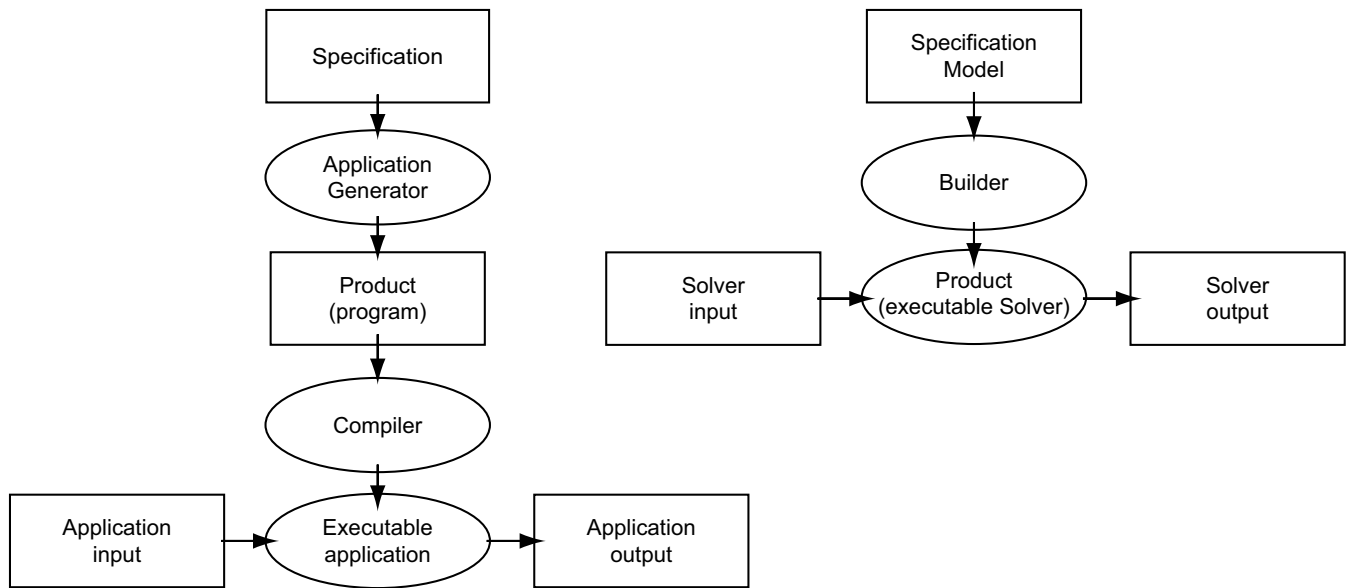


FIGURE 4.5: Schematic diagram showing the involved stages using an application generator [27] (left side) and using a Builder (right side). Ellipses represent executable programs or objects.

- (a) check for correctness of the model description,
 - (b) test model description for presence of ME or constructs which can not be handled by the algorithm, e. g., kinematic loops,
 - (c) test if context meets the builder requirements.
- (ii) *Preprocessing*: In the next step the builder may create intermediate MBSModelSpec specifications, e. g., to adapt desired model properties. In Section 4.2.2 this topic will be discussed. The main goals are:
- (a) improvement of run-time characteristics.
 - (b) application of algorithm specific port and connection semantics, e. g., how to treat ports which are not connected, or how to treat branchings, etc. .
- (iii) *Code generation*: Finally the builder assembles objects containing portions or the entirety of the required mathematical equations.
- (a) apply port semantics, which port is interpreted as spatial and which as scalar.
 - (b) detect in which causality the transformations of the ME are required. That is formation of a spanning tree by graph search methods such as depth- or breadth-first-search [5]. The special properties of an MBS connection according to Section 2.3 are considered.
 - (c) obtain an implementation of the transformation (in inquired causality and coordinate representation) contained in objects of class MBSEntImpl from a kind of database. These MBSEntImpl objects can be viewed as the data base for (pre-coded) component equations.
 - (d) connect the MBSEntImpl by establishing kinds of dataflow channels formed by objects MBSConnectionImpl .

TABLE 4.2: Main classes forming the model of computation.

Class name	Purpose
Builder	Algorithm specific code generator
Solver	Executable algorithm object created by Builder
MBSEntImpl	Algorithm specific mappings of multibody entities. One block represents the behaviour of a single ME or a sub-graph of MEs for one algorithm by implementing a special interface.
MBSEntImplFactory	Objects providing one MBSEntImpl object from a given specification, especially from a given MBSEntSpec.
ConnectionFactory	Topology mapping part of a Builder. Realizes parts of the connection semantics. Creates MBSConnectionImpl attaches them to the MBSPortImpl parts.
MBSConnectionImpl	Software connector, the dataflow 'channel' between MBSEntImpl objects
MBSPortImpl	The terminal objects
MBSAttributImpl	Implementation of MBS attributes, e. g., physical property mass
MBSExteriorPortImpl	Interface to access data from ports

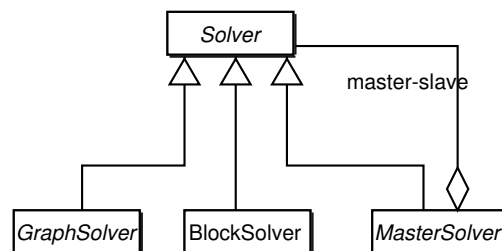


FIGURE 4.6: UML class diagram showing the solver base classes. Each solver type represents a different model of computation: GraphSolver controls a dataflow network, BlockSolver a monolithic equation block, and MasterSolver exploits another solver for related computations.

The main difference between an application generator and a builder is, that the latter is able to immediately create or reconfigure executable Solver instances, thus does not need to invoke a time- and resource-consuming compile-to-code step.

From a specification perspective [43] a solver represents an algorithm interface controlling the desired behaviour, e. g., inverse dynamics computation. The implementation of the of the behaviour is possible through various models of computations represented by the descendants of Solver shown in Figure 4.6. BlockSolver is the base class for algorithms consisting of one single equation block which can not be decomposed on implementation level. BlockSolver is to wrap existing functions or automatically generated equations for whole robots. The purpose of GraphSolver is to control a complete dataflow network of interconnected components. In this case the solver acts like the Multi-Graph-Kernel known in the Model Integrated Computing approach [138, 56] conducting computations by controlling the dataflow in a dataflow network and collecting the results from the graph.

This approach of distributing the computations has two distinct advantages: it enables either possible re-use of equations and code at compile-time or re-use of numerical results during run-time through solver coupling. The simplest form of run-time coupling is performed by objects of class MasterSolver which synthesize the desired numerical results from one or more other solvers, called slaves. This type of solver

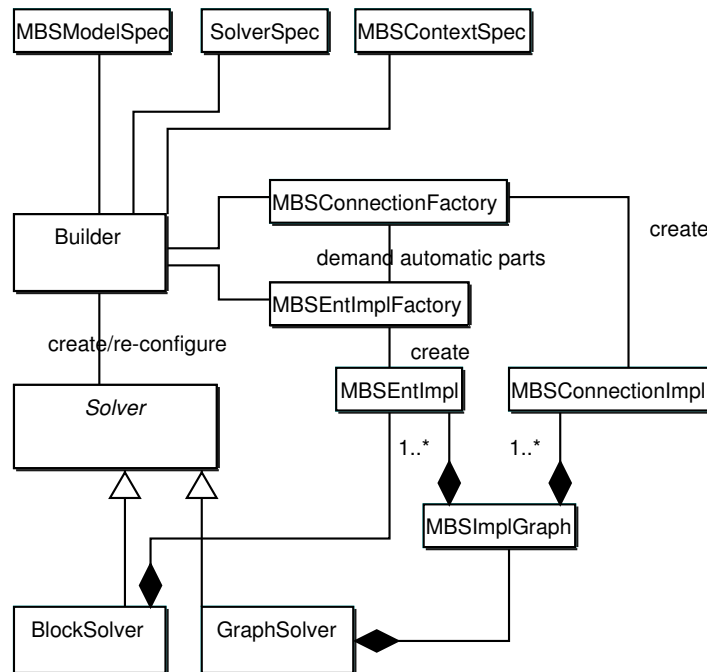


FIGURE 4.7: UML class diagram of classes involved in generating a Solver from a Builder. For a more detailed discussion of the relations see text.

for instance perfectly fits for realizing the method of pseudo-velocity Jacobian computations described in Section 3.2.2.1 using a forward kinematics solver. An overview of the involved classes and their relations is shown Figure 4.7 and explained in more detail below.

MBSEntImplFactory A database-like component returning an object from a given input specification, either a single MBSEntSpec or a subgraph of the MBSModelSpec, and the required causality. One possible realization is using a factory pattern [44].

MBSConnectionFactory Proposed by Fischer and Hörmann [42] to apply algorithm and port specific connections semantics based on local arguments, creating local reconfigurable subgraphs. Handles special cases, for instance branchings in tree-structured systems, which require special treatment in dynamics algorithms.

MBSEntImpl Is a dynamically created algorithm specific object that allows the modeling of dynamically changing numerical schemes while ensuring that all valid communication relationships between the objects are specified explicitly. This ensures architectural and numerical integrity. MBSEntImpl encompasses an algorithm specific interface TransformationSet implemented by equations prescribed by the behaviour of a ME in a specific algorithm. In the multibody system domain this represents the knowledge of the robotics or mechanical engineer, equations coded and precompiled in well-defined causality. It is a generalization of the concept of *transmission objects* introduced in [74]. The advantages are

- *Flexibility*: can be implemented automatically or by manual coding.
- *Transparency*: in many cases direct mapping from one multibody entity to the algorithm, i. e., just a direct implementation of the PSOA transformation equations.

- *Efficiency*: optimization of the implementation contained in the member functions is possible without sacrificing readability, because desired behaviour is well defined via the interface.
- *Re-usability*: the equations of each multibody component are uniquely ordered by the categories multibody entity, algorithm type, coordinate representation, and causality. A possible C++ template declaration of one `MBSentImpl` expressing the dependency from these factors is the following fragment

```
template <
typename MBSentSpec,
typename Algorithm,
typename CoordRepresentation,
typename Causality
> class MBSentImpl;
```

and

```
MBSentImpl< MBSentSpec<Displacement<Type1> >, RNE, BF, IP1IN_IP2OUT>
```

is an example class incorporating the coded recursive Newton-Euler transformations for the ME `Displacement<Type1>` in body-fixed representation (BF) where the causality is defined so that the forward recursion (sweep 1) of the RNE algorithm is directed from IP:1 to IP:2 and the backward recursion (sweep 2) vice versa. The `TransformationSet` interface for this two sweep algorithm might look like

```
template <RNE> struct TransformationSet
{
// Sweep 1 (outboard)
doPosition();
doVelocity();
doAcceleration();
// Sweep 2 (inboard)
doAction();
};
```

where the first three `do...` operations execute the outboard kinematics recursion which is subdivided in calculation of position, velocity, and acceleration, and `doAction()` is the force calculation during the inboard sweep. This interface may be directly implemented using the transformations defined in Table 3.6.

- *Safety*: code implementing well-defined interfaces of well-defined objects is easier to read, to maintain, to debug, and to document. This leads to code with more reliable characteristics, a requirement in the robotics domain where reliable function is vital.

MBSConnectionImpl A simple realization of a *software connector* [8] providing a binary relation between the two `MBSentImpl` objects, or to be more precisely between two `MBSPortImpl` objects contained in the `MBSentImpl` objects. It implements the protocols vital to recursive algorithms but carries no executional semantics.

4.2.2 Model transformation and optimization

The transformation of models is a crucial element in model-based endeavours. As models and meta-models in essence all can be represented by attributed, typed graphs, we can transform them by means

of *graph rewriting* [36]. The rewriting is specified in forms of *graph grammar* models [35] which are highly amenable to any graphical representations. The graph grammars are composed of rules. Each rule consists of left hand side (LHS) and right hand side (RHS) graphs. Rules are evaluated against an input graph, called the *host graph*. If a matching is found between the LHS of a rule and a sub-graph of the host graph, then the rule can be applied. When a rule is applied, the matching subgraph of the host graph is replaced by the RHS of the rule. Rules can have applicability conditions, as well as actions to be performed when the rule is applied. Graph-rewriting systems can have control mechanisms to determine the order in which rules are checked and applied. In case of MBS graphs the order is a characteristic feature of the type of desired solver type to create. After a rule matching and subsequent application, the graph rewriting system starts the search again. The graph grammar execution ends, when no more matching rules are found.

This section proposes the application of graph transformation methods in the context of MBS graphs. Graph transformation reaps the benefits of an abstract robot model specification with well-defined semantics: If a system performs local manipulations of the complex, structural, MBS specification graph then graph transformation is a good formalism to specify and program it. Possible applications are imaginable for:

- (i) modification of a model specification graph to meet the Builder requirements:
 - (a) steps before code generation. That could be pre-treatment of special constructs such as kinematic loops, which are cut at certain positions depending on local properties of sub-graphs.
 - (b) model transformation into another target formalism, especially textual representations.
- (ii) modification of the specification or implementation and dataflow graph to optimize a given model specification w. r. t. certain criteria, hence to reduce complexity. Prominent cases on various levels of abstraction are:
 - (a) optimization from abstract model specification, i. e. before equation code generation: In case of dynamics calculations it is possible to merge, e. g., rigid bodies which have a rigid connection to a single body resulting in the dynamics. That can be used to compactify automatically assembled and highly redundant robot models to reduce the number of inertia parameters and reduce the computational burden.
 - (b) optimization while code generation, in MBSentImpl objects: Depending on local properties of the graph and parameters in MBSentImpl objects one could create optimized code modules, if applicable. A prominent example are rotational joints, whose axis of rotation is aligned to one axis of a cartesian coordinate frame. The numerical effort equals that of a rotation in the plane.

The advantages of graph grammars are that they are a natural, formal, visual, declarative and high-level representation of the graph modification in contrast to implementating them in a traditional programming language. Computations are thus expressed in a graph grammar language expressed in the graph grammar formalism. There is a strong need for modularity, because kinematics models have differing requirements from dynamics models, for instance. These requirements can easily be met because transformations can be stored in libraries and applied to MBS graphs like filters. The theoretical foundations of graph rewriting systems may assist in proving correctness and convergence properties of the transformations applied.

A constraint inherent in graph grammars is that in the most general cases the subgraph testing is NP-complete. Because robot model specification graphs contain numbers of nodes typically of $\mathcal{O}(10 \dots 100)$ this is an issue for systems with extremely frequent model reconfigurations and under hard real time

constraints. A solution is to use graph grammars as specification for manual implementations, an issue not discussed further.

Especially for the application of robot models in a robot control system there remains some challenging and unsolved questions, beyond the scope of this work:

- *Consistency*: reducing complexity means to remove details which might eventually prove relevant in other computations.
- *Transparency*: graph rules could be a great means to formalize the domain knowledge of the robotics and mechanical engineer, e. g., the possibility to lump masses in dynamics computations. It is not known if a model graph after several graph rules is still expressive enough and leads to correct physical results.
- *Hierarchy*: graph rules may apply to parts of hierarchical model components (assemblies). There are several strategies to deal with this situation. Flattening the graph would be one possibility but removes the relations, e. g., between model parameters, expressed by the hierarchy.
- *Reconfiguration*: Modifying the underlying graphs dynamically is a good way to react on structural changes in the model. The question is how to formally capture these modifications by graph grammars without sacrificing efficiency.

4.3 Reusable components of robot models in control

The notion of a solver introduced in Section 4.2.1 denotes (i) the executable version of an algorithm and from a specification perspective [43] (ii) the responsibility for a certain behaviour, in object-oriented terms an *interface*. This section presents a number of interfaces which, e. g., can be implemented by the algorithms discussed in Chapter 3. As complexity of all software based systems grows, software integration is becoming more and more a key enabler. Software integration costs in US robotics industry alone are estimated at \$1 billion annually [114]. This section emphasizes the relevance of well-defined interfaces for dynamics computations and presents interfaces to allow for seamless integration and exchange of dynamics algorithms in a control software architecture.

4.3.1 Basic solver interfaces

This section sketches the responsibilities of the solver base class hierarchy shown in Figure 4.6, the corresponding UML interface of class Solver is shown in Figure 4.8.

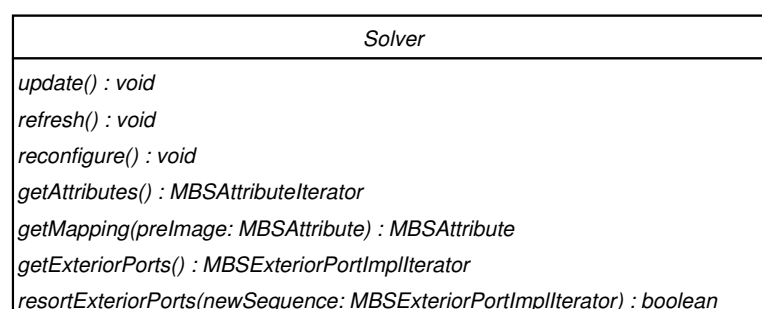


FIGURE 4.8: UML class diagram of interface Solver.

The operations of the base class `Solver` correspond mainly to two fields. Handling changes and the mapping from the model specification to the computational model. This work subdivides changes in three domains, change of time-varying data, change of data which is supposed to be time-invariant, and structural change. The most important operation is `update()` which tells the solver to process new joint angles, rates, etc., for a new time-step or system state. The responsibility is to update internal matrices *without* doing the actual computations, to avoid multiple expensive evaluation of matrices in repetitive computations, which is the norm in robot dynamics. The operation `refresh()` commands the solver to update matrices from data which are supposed to be time invariant, especially data describing mass, inertia, and geometric properties. This is required after changing the tooling or payload or a during a calibration process.

A solver is the product of the builder process described in Section 4.2.1. Operation `reconfigure()` is commanded exclusively by the builder to react on structural changes in the preimage of the solver, the model specification. In the worst case the complete translation process has to be performed which might be expensive, therefore it is realized as a separate method. The method `getAttributes` returns an iterator—a certain method to loop over a collection of objects [44]—which MBS attributes are provided by the solver. This structural information reveals which coordinate frames are present, which positional states are contained, where mass is distributed, etc. The method `getMapping` is also concerned with the mapping established by the builder. It returns if possible the preimage of an attribute of an MBS entity, e. g., the implementation of the joint angle attribute in a certain implementation important to compute the number of positional and velocity state variables.

Access to exterior ports, i. e., the exposed states and frames in the model, either input to provide joint values (tag IN) or output (tag OUT) to retrieve the position etc, is permitted by the operation `getExteriorPorts` which returns an iterator over all exterior ports present in the model. This is intended as a common interface for access to all kinds of protocol data in the model, either of Cartesian or joint space nature. In other words to provide a way to access the involved data independent from the type of algorithm and protocol data. The operation `resortExteriorPorts` is intended to provide a new ordering or a selection of states or frames. This is rather important to influence the layout of matrices, which is crucial when communicating and cooperating with other functions. For example an optimizer may expect information about state variables in a column vector of certain layout.

The interface `GraphSolver` shown in Figure 4.9 extends the interface `Solver` by two operations. These are analogous to the `getAttributes` and `getMapping` operations, but concerned with the direct mapping of MBS components. The idea of the Port-Based Spatial Operator Algebra is to provide the multibody equations in a component oriented form, which are realized in `MBSentImpl` objects. In case of the `GraphSolver` the model specification graph is mapped to a topologically equivalent dataflow graph of interacting `MBSentImpl` objects.

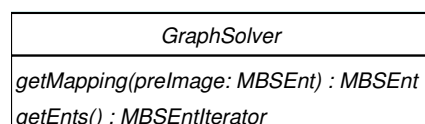


FIGURE 4.9: UML class diagram of interface `GraphSolver` extending the interface `Solver`.

4.3.2 Interfaces for algorithm building blocks

This section presents interfaces for selected algorithms, some of them presented in this work. For the sake of clarity only the most relevant operations are listed in the interface declarations. It is not defined if the

solver is implemented by a graph-, block, or master-solver, because the behaviour to be expected must be the same. All solvers are subtypes of the basic interface Solver.

4.3.2.1 ForwardKinematics

Forward kinematics computes the Cartesian position and velocity of frames belonging ports connected to ExteriorPort objects. The advantage is that this is not restricted to one single endeffector frame, though in robotics often forward kinematics often means computation of the position of a *single* endeffector frame. This is too restrictive, for example in collision avoidance schemes, so the interface provides access to an arbitrary number of frames in a model. The two operations doPosition and doVelocity start the kinematics computation but do not return results. The results are obtained from ExteriorPort interface objects, whose interface is shown on the right part of Figure 4.10. The position data is compactly retrieved using a homogeneous transformation object NumericalMatrixHomTrafo comprising the positional vector and the rotational matrix. The desired joint position and velocities are set through the exterior ports tagged IN.

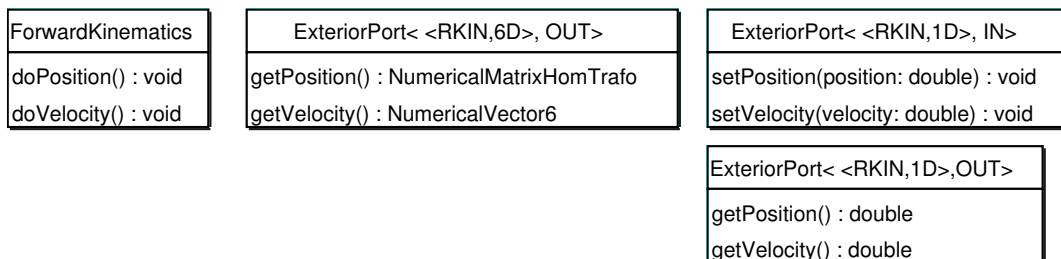


FIGURE 4.10: UML class diagram of interface ForwardKinematics and the interfaces of involved exterior port objects to set and retrieve data.

4.3.2.2 CenterOfMass

This interface is to compute the total robot mass m^+ , position of center of mass frame F_c^+ , and the total moments of inertia w. r. t. the center of mass frame ${}^c J^+$ of a robot model. The center of mass of multibody systems is a uniquely defined frame for a given model in a given positional state. The positional state is again set by accessing the exterior ports through the base class solver, then the computation of the center of mass frame is started by the operation doCOM(), the computation of the total mass by doMass(). The inertial parameters relevant are the total robot mass, and the moments of inertia wrt the center of mass frame, are obtained from the operations getMass, getPosition, and getInertias.

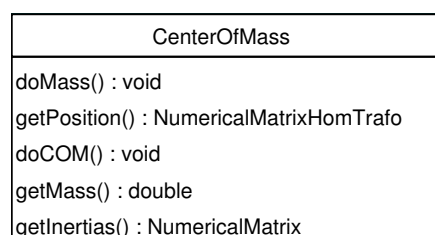


FIGURE 4.11: UML class diagram of interface CenterOfMass.

4.3.2.3 MassMatrix

There are several methods to compute the (joint space) manipulator mass matrix \mathcal{M} explicitly, the classical methods have been presented by Walker and Orin [147] which may be applied using any coordinate representation. The interface shown in Figure 4.12 which can be implemented by these methods is always the same because the mass matrix is uniquely defined. The layout of the mass matrix depends on the ordering of exterior ports which is determined by the method `resortExteriorPorts` inherited from the base class `Solver`.

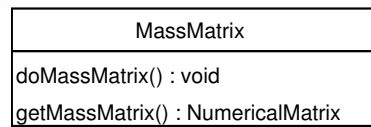


FIGURE 4.12: UML class diagram of interface `MassMatrix`.

4.3.2.4 InverseDynamics

The inverse dynamics can be interpreted as a mapping from joint positions, velocities, and accelerations to generalized applied joint actions. The only service provided is to start the computation as shown in Figure 4.13. The input and output values, the states, are transferred via the exterior ports with tag `IN`. The mapping remains in the space of joint variables, however, the computation often is performed in Cartesian space, e. g. in recursive Newton-Euler schemes. So, if desired, values living in the Cartesian space, for instance cut forces, can be extracted through the exterior port interfaces shown on the right side of Figure 4.13.

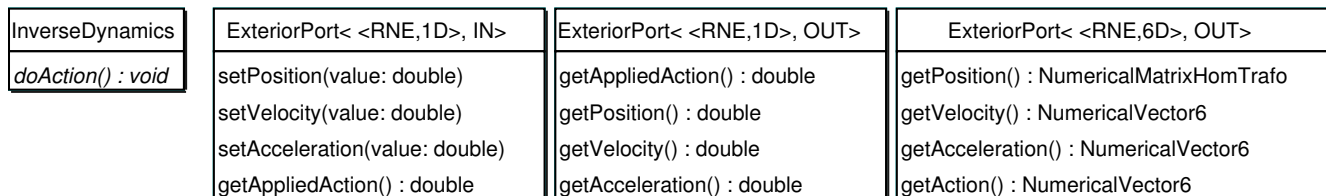


FIGURE 4.13: UML class diagram of interface `InverseDynamics` and related exterior port interfaces.

4.4 Aspects relevant to realizing a dynamics framework

The objective of this section is to describe some topics relevant in the practical realization of a framework for object-oriented robot modeling using the design concepts presented in the preceding chapters. In object-oriented terms a *framework* [70] denotes an infrastructure for integrating software components, i. e., it is a collection of classes obeying a high-level design [69]. Frameworks are one possible type of object-oriented software—design and code—reuse technique and present a possible methodology for practical realization of the concepts developed in this work. Johnson motivates frameworks but does not withhold that there is a price to pay:

[...]. Applications seem infinitely variable, and no matter how good a component library is, it will eventually need new components. Frameworks let us make a new component [...]

out of smaller components [...] and they also provide the specifications for new components and a template for implementing them. [...] but [it] requires domain analysis and domain engineering, so there is a big expense before benefits can be realized. [69]

On the one hand a dynamics framework should prevent reinvention of the wheel by providing a common 'language' to realize and systematize the various complex multibody formalisms. Particularly in the field of robotics emerge continuously new types of robots, dynamics formalisms, and control schemes, e. g., [10, 59], which should integrate seamlessly into an existing framework. The goals are the ability to add a new component without any modifications to existing algorithms or to the main portions of the code itself, and that new algorithms will be interoperable with the elder ones. This requires detailed domain analysis and domain engineering, a big expense before before benefits can be realized—the work done in the preceding chapters.

Choice of the language

Any object-oriented language can be applied to express a framework [69]. Languages supporting natively the object-oriented features required in this thesis are C++, Java, and latest Fortran versions. Computational efficiency of the resulting code must be the primary criterion for the choice. Unfortunately the availability of compilers on the various platforms is currently the most restricting factor.

Because dynamics model computations for robot control are required to run on workstations, real-time systems as well as on digital signal processors involved in joint control, the best compromise is to choose C++ at the time this work was finished. It is currently industry standard in the domains of robotics, signal processing, embedded systems and available on nearly every platform. The computational performance has been shown to be close to Fortran codes, when obeying certain coding rules and constructs.

Guidelines for gaining maximum performance are discussed in [102], a comparison of the efficiency of certain benchmark problems for a large number of compilers and platforms can be obtained from here: <http://annwm.lbl.gov/bench/>. Some results for dynamics computations using a C++ realization of this framework are discussed in [56].

Linear algebra computations

Nearly all multibody dynamics computations are expressed by means of linear algebra. An efficient and reliable implementation of a linear algebra library is fundamental for high performance dynamics computations. Because of the physical nature of most mechanics problems, concerned spaces in computations are \mathbb{R}^3 and \mathbb{R}^6 involving structured matrices $\in \mathbb{R}^3, \mathbb{R}^{3 \times 3}, \mathbb{R}^6, \mathbb{R}^{6 \times 6}$. Object-oriented methods are perfectly suited to represent these special matrices and their operations while exploiting the special structure and operations. Especially for multibody dynamics problems it has been shown that exploiting the structure is computationally very advantageous and can help to reduce the number of floating point operations significantly [90, 136].

Graph abstraction

Wittenburg [149] points out that a natural and powerful representation of a multibody system is a graph. The framework presented in this thesis relies on graph structures on nearly each conceptual level. In the formulation of the equations in the Port-Based Spatial Operator Algebra, in the specification model as well as when implementing algorithms by means of objects of type `GraphSolver` and `MBSentImpl`, or in

certain algorithms such as the solution of kinematic loops in Section 3.2.3 for detection of the solution strategy. Therefore a generic way of abstracting graphs is indispensable for a realization of this framework. A promising candidate is the *Boost Graph Library* [128], an extension to C++, which generically supports all kinds of graph topologies and types and provides powerful mechanisms for graph traversal in form of generic iterators.

Assuring correct function

Robotics is an extremely safety critical domain. For example in automotive industry with a high degree of automatization the economic loss can be considerable if just one welding robot fails in a car production line. In the emerging domain of service robotics with inevitable contact to living beings, damage to humans may occur by no means. General guidelines for safety critical applications C++ can be found in [17, 102]. One general approach to provide correct function is *Extreme Programming* [14], where an integral component of each portion of code is a number of testcases checking inherently the correctness of classes.

Correctness is especially hard to assure in multibody dynamics. Because of the complexity of the equations and the great number of components testing is not as straightforward [115]. The choice of well-behaved and robust algorithms is a first requirement. The presented framework and solver interface presents a good basis for testing results because an interface requires to produce identical results (in the order of the numerical precision) from identical input. So implementation of several algorithms producing the identical results can be exploited for testing purposes. Even better is the existence of an inverse mapping, for example the result of the inverse dynamics can be used as input for a forward dynamics for the same robot. This must lead exactly to the input joint accelerations.

Responsibilities

Taking a look on Figure 4.4 one can identify several roles in the application generator design process. For a dynamics framework the three most significant are:

- System designer: this for example is a robot control engineer who requires a certain type of computation for his robot control scheme under consideration, or a robot design engineer who requires a certain computation executed for a class of robots, for instance to perform optimal design studies. These are the final 'users', who require moderate coding capabilities and multibody domain knowledge.
- Domain engineer: this is multibody domain specialist or mechanical engineer who analyses and implements new dynamics algorithms using the framework and its features. He needs moderate to medium coding capabilities and excellent multibody domain knowledge.
- Domain analyst: a multibody domain specialist with good knowledge in object-oriented modeling and programming and a broad knowledge about the requirements from the system designers. He is responsible for the design, implementation, and extension of the fundamental framework class hierarchy.

Package subdivision

Physical distribution over files and directories is a crucial point in library development [81]. The following list is a proposal for package subdivision in a realization, package names are in sans:

- kernel: Provides the basic services required in all classes, but *no* algorithms and equations
 - specification_model: contains all classes concerning the specification model
 - linear_algebra: contains the crucial optimized linear algebra class library
 - graph: contains the graph abstraction classes
- components: this package contains implemented equations for the multibody entities, the mechanical components and solely depends from package kernel
 - protocols: contains the protocol classes and fundamental interfaces
 - mbsentimpl: contains MBSEntImpl objects implementing the multibody equations
- solver: this package contains builder/solver pairs for the algorithms

Documentation

Of course code has to be well documented. The additional requirement of a dynamics framework is the ability to express mathematical entities of great symbolical complexity. So a tool is favourable which is able to integrate mathematical equations. A recommendable public domain tool which is able to process \LaTeX source code and extracts documentation directly from source code augmented by special control tags is *doxygen* [142].

Chapter 5

Selected applications in research and industry

This chapter reviews some applications of the methods presented in the preceding chapters using a C++ realization of which first results have been published in [57]. The presented applications span real-world industrial applications and challenging research topics. An overview is given below in Table 5.1.

TABLE 5.1: Overview of applications presented in this chapter.

robotic system	model type	joints	contribution	application
light-weight robot (DLR)	rigid body, elastic joint, fixed base	7	structure of gyro coupling matrix elastic joint inverse dynamics	efficient code generation investigation of effect, prerequisite for feedback linearization
pallettizing robot	rigid body, partial kinematic loops	9	modular closed form solution of inverse dynamics, automatic detection of loop coupling	inverse dynamics for on-line trajectory optimizer
manufacturing robot	rigid body, fixed base, elastic joints	6	sensitivity calculation by pseudo-velocities	calibration, identification, elasticity compensation
legged robots	rigid body, free floating, tip contact	12	integrated object-oriented model: reduced dynamics, contact/collision, sophisticated boundary conditions	trajectory optimization of legged robots

5.1 DLR light-weight robot inverse dynamics

In this section the new dynamics method developed in Section 3.2.1 is applied to the DLR light-weight robot LBR 2. The method allows for (i) powerful symbolical equation manipulation such as differentiations and (ii) deriving an efficient algorithm for numerical evaluation of the elastic joint inverse dynamics.

Properties of the DLR LBR 2

Service robotics will become a driving force for robotics industry in the next decades. After robots have been established in factory automation and as toys, there seems a huge market for smart light-weight robots acting in the vicinity of humans. In contrast to purely position controlled standard industrial robots with very stiff mechanics and a payload to weight ratio of 1:10 or even higher a service robot has to be designed and controlled completely differently.

The series of DLR light-weight robots (LBR) follows the vision to form a manipulator arm similar to the human arm [52]. A picture of version 2 is shown in Figure 5.1(a). To approach this vision a mechanical structure with a low dead weight and an optimized payload to weight ratio is required. The LBR 2 manipulator arm with approximately 1 m length and fully integrated actuators and electronics is capable of handling a payload of 8 kg with a total weight as low as 17 kg. The manipulability for service applications requires a considerable number of mechanical degrees of freedom, therefore the LBR 2 is a redundant 7 axes robot. Advanced sensory equipment and feedback capabilities to improve impedance, stiffness, and damping in control is mandatory in order to increase precision and safety while performing tasks in an environment with humans.

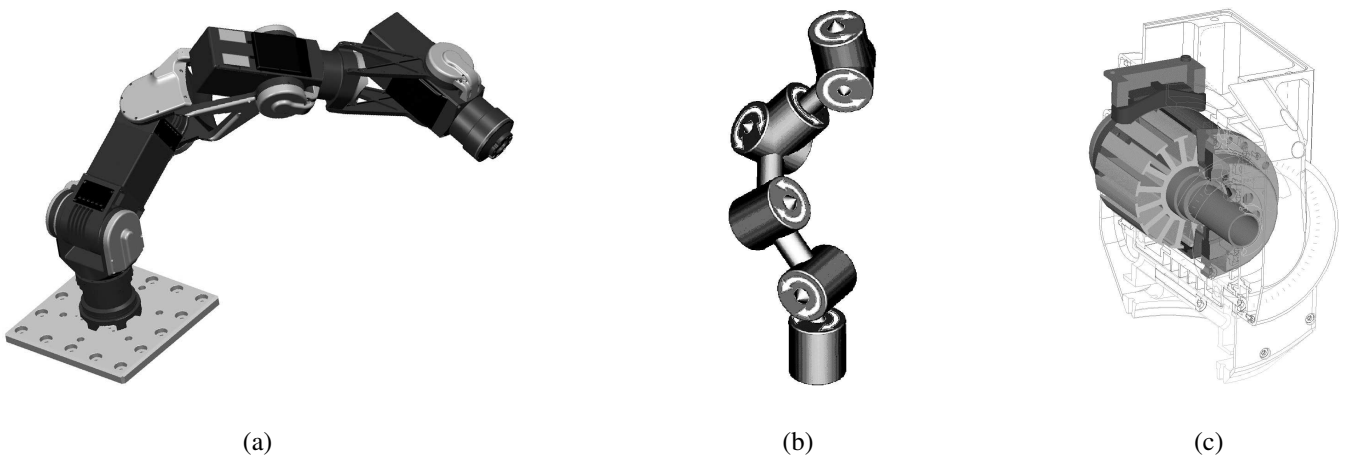


FIGURE 5.1: (a) The DLR light-weight robot LBR 2. (b) The kinematic structure of LBR 2 just showing the pairwise perpendicular joints. (c) Drive used in DLR lightweight robot LBR 2. Drawings courtesy of DLR e.V.

The LBR 2 is a good example for practically relevant robots and clearly indicates the problems in computation of the elastic joint inverse dynamics for non-trivial robots. It is a chain-structured mechanism with seven revolute joints as shown in Figure 5.1(b). The motors are mounted in the joints, their rotors' axes of rotation coincide with the joint axes, as depicted in Figure 5.1(c). The drivetrains are known to be elastic due to harmonic drive reduction gears and torque sensors mainly due to the extreme light-weight design. Joint stiffnesses are roughly $10^4 \frac{\text{Nm}}{\text{rad}}$. A more detailed description of the technical details of the LBR 2 can be found in [6].

Applying the methodology for deriving and implementing the inverse dynamics

The ability to calculate the inverse dynamics is a basic prerequisite for path planning algorithms and feed forward controllers in robotic systems. The elastic joint inverse dynamics presented in Section 3.2.1 is a non-standard new algorithm for calculating all driving forces from predefined trajectories, e. g., joint positions. Therefore it is a perfect example for the application of the presented methodology which provides a complete means to obtain an executable algorithm 'from scratch'. From scratch denotes the sequence of three steps of (i) object-oriented physical modeling, (ii) symbolic equation manipulation, and (iii) object-oriented code generation and implementation using the proposed class hierarchies presented in Chapter 4.

First, step (i) comprises the choice and description of the behaviour of objects representing mechanical components and physical effects of the robot and the attached drivetrains. This is covered by the Multi-body entity paradigm presented in Section 2.2 and specialized drive-train components in Section 3.2.1.3. Second, the topology of the multibody system graph representing the LBR 2 robot is described by means of the specification model presented in Section 2.2.2.

The topological information is crucial in step (ii) to establishing the equations of the elastic joint inverse dynamics, because describing the topology explicitly reveals additional kinematical properties of LBR 2 robot which is indispensable for efficient code generation. The rigid body equations for the whole robot are easily established in symbolical form from the component equations shown in Tables 3.6 and 3.14 by means of the Port-Based Spatial Operator Algebra presented in Section 2.3. As pointed out in Section 3.2.1 the equations of motion need to be differentiated up to an order of $2N$ in the general case, N being the number of joints. For the LBR 2 the first off-diagonal of the matrix $S(q)$ (q is the vector of joint positions) vanishes according to the new Equation (3.41) because consecutive axes of the robot are orthogonal as depicted in Figure 5.1(b). Considering (3.44) the structure of $S(q)$ (see Section 3.2.1) which was derived using the new Port-Based Spatial Operator Algebra presented in Section 2.3 is

$$S(q) = \begin{pmatrix} 0 & 0 & S_{13}(q_2) & \dots & S_{17}(q_2, \dots, q_6) \\ 0 & \ddots & \ddots & \ddots & \vdots \\ \vdots & \vdots & \ddots & \ddots & S_{57}(q_6) \\ 0 & 0 & 0 & \ddots & 0 \\ 0 & 0 & 0 & \dots & 0 \end{pmatrix}. \quad (5.1)$$

This reduces the required number of derivatives to 12 since the solution of the sixth equation of 3.15 on page 47 no longer depends on the position of motor 7 $\ddot{\theta}_7$. The concrete structure of $S(q)$, especially its explicit independence from q_1 and q_7 , is exploited to generate efficient code. It is worth noting, that taking for granted zero and constant S -matrix, a simplification which is applied in control literature [32], is only valid for academic robots, such as planar or elbow robots with two or three joints, but not for this redundant manipulator with seven axes.

In step (iii), the implementation step, the multibody code of all components, i. e. joints, bodies, and drivetrains, for an arbitrary order of differentiation is implemented or better automatically generated from a script using the symbolical recursion formulae presented in 3.2.1 and embedded in `MBSentImpl` objects. Finally the design principles proposed in Section 4.2.1 are applied to form a solver object implementing a dedicated elastic joint inverse dynamics solver interface to conveniently access the dynamics computations and providing the resulting matrices containing the desired motor torques.

Investigation of elastic and gyroscopic effects

In the following the effect of this detailed model in comparison to a rigid model neglecting elasticity and gyroscopic effects of the drivetrains is investigated. The example trajectory shown in the inset of Figure 5.2 was chosen to be a step input of 2 rad at time $t = 0$ s smoothed by a filter of order 16, for all joints $q_1(t) = \dots = q_7(t)$, $t \in [0, 2.5 \text{ s}]$, in order to ensure the required smoothness for differentiation [141]. The filter dynamics chosen provides a trajectory that satisfies motor torque and velocity constraints of the real LBR 2 [6]. The obtained motor torques of three axes τ_1, τ_2, τ_3 are shown in Figure 5.2. The influence of drivetrain effects on the dynamics are shown in Figure 5.3 illustrated by the difference in motor torques between the full elastic model and the rigid model. The results significantly differ even for the very smooth trajectory chosen. For faster motion this effect increases due to proportionally larger derivatives. In case of high-speed movements and high precision requirements it is recommendable taking into account drivetrain effects in model-based control schemes.

Further applications of the elastic joint dynamics

Inverse dynamics schemes also are a central part of many controllers, especially feedback linearization [131]. As pointed out by De Luca and Tomei [33] the complete dynamic model of robots with elastic joints fails to satisfy the necessary conditions for input-output decoupling and/or full linearization by static state feedback. For the general case it is useful to resort to the larger class of dynamic state feedback controllers [32]. In this control scheme, the input torque applied to the robot depends both on the robot state and on the state of a dynamic compensator of proper dimension.

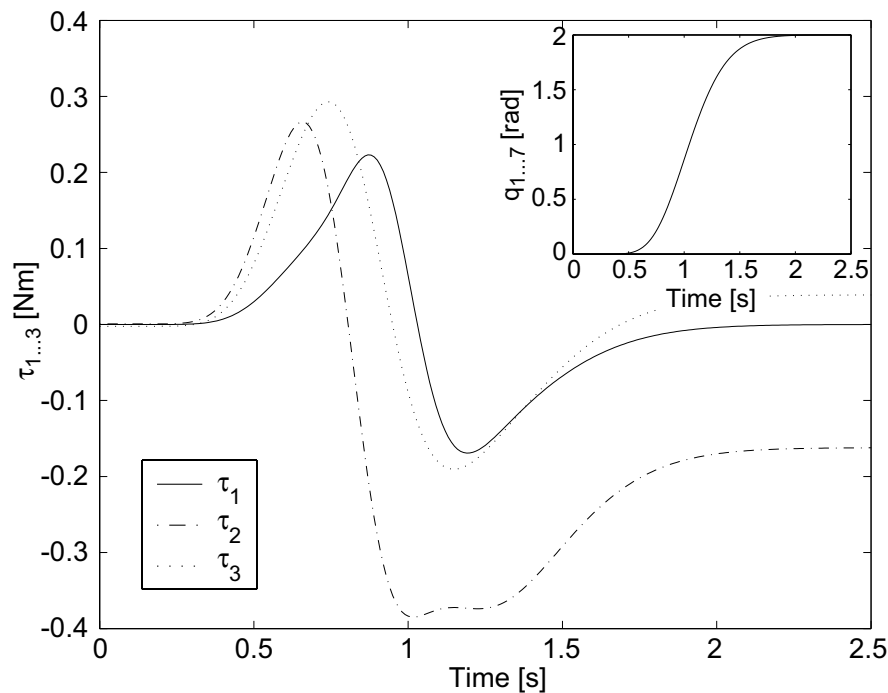


FIGURE 5.2: Inset: Example trajectory used in simulations. Large plot: Resulting motor torques $\tau_1, \tau_2,$ and τ_3 from elastic joint inverse dynamics computed using Algorithm 1 in Section 3.2.1.

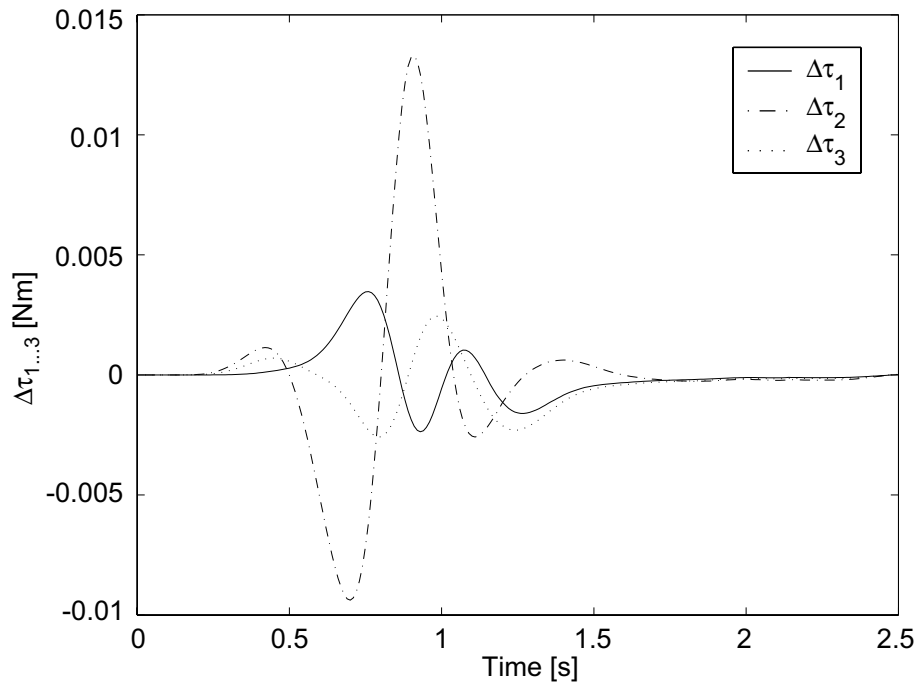


FIGURE 5.3: Differences between elastic model and rigid model inverse dynamics for the trajectory from Figure 5.2.

Up to now no implementation of a dynamic feedback linearization controller, not even for the simplest robots with elastic joints [32] has been carried through. This is due to the complexity of the multibody equations which have to be manipulated to get the dynamic compensator. The completely symbolic PSOA derivation presented in Section 3.2.1 using closed form spatial operator expressions is a promising start. Further improved PSOA expressions will pave the way to solve the problem of feedback linearization of elastic joint robots for the first time in general. A problem left for further research.

5.2 Modeling a palletizing robot

This section describes the inverse dynamics modeling, e. g., required for on-line path-planning, of an industrial robot containing kinematic loops. First results have been published in [57].

Industrial robots often are designed to meet the specific needs mandated by the application. Figure 5.4(a) shows a real-world palletizing robot with four mechanical degrees of freedom optimized for assembling pallets and filling containers from top. Its payload is 180 kg though several links are obviously quite light weighted. This is possible through a parallel kinematics which is sketched in Figure 5.4(b) for a very similar mechanism. It is a primary kinematic chain of 5 revolute axes R_1, \dots, R_5 , augmented by two coupled closed-chain mechanisms forming a parallel kinematics. This has two advantages. First, the mechanism has improved stiffness and requires lower motor power in the plane of operation which is perpendicular to axes R_2, R_3, R_4 . Second this keeps the tool flange permanently parallel to the ground without the need for actuation of a fifth axis. Both properties result in lower weight and reduced power consumption.

Dynamic path planning algorithms are used to determine the control variables, usually joint torque or motor current, required to optimally move the robot's endeffector along a desired contour or between given start- and end-locations. The usual definition of 'optimal' denotes time optimality, i. e., completing

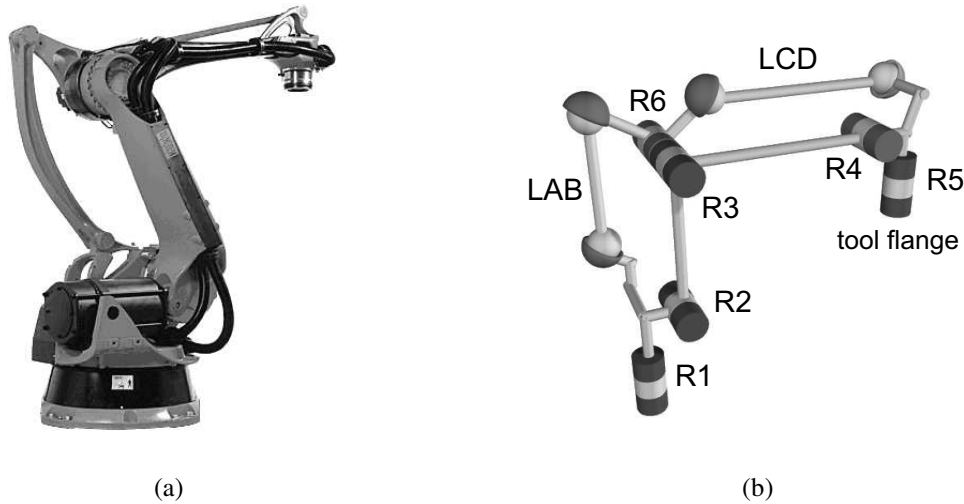


FIGURE 5.4: Left: The KUKA KR180PA palletizing robot. Photo courtesy of KUKA Roboter GmbH. Right side: Schematic view of the resulting palletizing robot model showing two coupled kinematic loops.

a task in minimum time. For robotic systems efficient schemes have been developed to compute these optimal trajectories, e. g., see [18, 112]. These methods are based on the idea that the endeffector motion is parametrized by a single path-parameter $s(t)$. Then the equations of motion (3.1) can be expressed in terms of $s(t)$:

$$a_i(s)\ddot{s} + b_i(s)\dot{s}^2 = g_i(s) + u_i, \quad (5.2)$$

where the column matrices a , b , and g can be computed from the rigid-body inverse dynamics evaluations [112]. Also general purpose numerical optimal control methods are available to compute optimal trajectories for general performance indices and subject to general robot dynamics models and constraints [146, 145].

Obtaining the inverse dynamics for a robot containing kinematic loops is more challenging than for tree-structured systems because not all joints are driven and joint states are primarily unknown. Using the methods presented in Section 3.2.3 the inverse dynamics solver doing the computations is prepared by the following steps: The given model specification graph is analyzed w. r. t. the interconnection topology of the joints, driven or passive joints, and orientation of the joint axes according to a given heuristics discussed in [150]. Analysis of the example mechanism shown in Figure 5.4(a) reveals two coupled kinematic loops. If the criteria for cutting loops to obtain a closed form solution of the constraint equations apply, then the given model description is transformed to a new one, amenable to explicit solution. This means loops are cut and joint pairs are eventually replaced by connecting rods.

For the palletizing robot this results in the topology shown in Figure 5.5, where the four revolute joints from the parallel kinematics are replaced by two connecting rods. This procedure is legit for numerical and efficiency reasons, if the desired values are the driving torques. Loop 1 in Figure 5.5 contains the revolute joint R6 as a free joint and loop 2 R4 and R6. The algorithm presented in Section 3.2.3 automatically determines the sequence for the solution: solve loop 1 for joint angle of joint R6, then solve loop 2 for the remaining angle of joint R4. The procedure is analogous for joint rates and accelerations. A single inverse dynamics evaluation for this closed loop system is about $75 \mu s$ on a 200 MHz Pentium PC, in contrast to $36 \mu s$ for a standard 6-axes robot. Evaluation of the a , b , g matrices requires about N_d inverse dynamics computations [18]. Because the cycle time of industrial robots lies in the order of magnitude of 1 to 10 ms, both qualify for real-time trajectory planning schemes as described above.

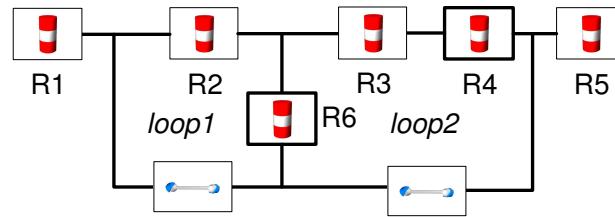


FIGURE 5.5: Schematic view of kinematic structure just showing the joints. Please note, only joints R1,R2,R3, and R5 are driven.

5.3 Calibration models of manufacturing robots

This section sketches the modeling requirements for cyclic calibration and parameter identification of industrial robots, and how they can be fulfilled by the methods proposed in this thesis.

Robot calibration is a method to improve robot positioning accuracy through modification of the robot *software* rather than changing or altering the design of the robot or its control system. It involves a computational model describing the relationship between the joint transducer readings and the desired end-effector position in workspace. In contrast to adaptive control schemes where parameters of models and controllers are continuously adjusted, calibration is a discrete event in time. Physical parameters influencing the end-effector position, such as link lengths and elasticity coefficients, are uncertain or robot hardware is changing with time as are environmental conditions. Unfortunately, these parameters normally are not directly observable. The unknown model parameters have to be identified indirectly from measurements, for instance by recording experimental trajectories, and using a model-based mathematical optimization procedure. The new parameter set is updated to the software after each calibration event. For more details the reader is referred to [54].

Manufacturing robots are designed for high performance in positioning tasks such as welding. This class of robots are assembled from very stiff mechanical parts, joints and links, to increase overall positioning accuracy. Thus a rigid body model is usually a very good approximation. Two inevitable sources of errors, however, lead to a mismatch between the computational model in the robot's software and the hardware it represents. The first originates from tolerances in serial production of industrial robots. The second stems from changing parameters w. r. t. time of operation. The most dominant effects known are temperature drift of geometric dimensions and joint elastic as well as friction coefficients on short time-scales, especially during machine warm-up, and wear of parts and replacement after maintenance on a long time-scale.

Using the methods presented in this work a set of three models has been implemented which provides (i) a rigid body model, (ii) a more detailed model to account for link and gear elasticity in the start and end-position of point-to-point trajectories (trajectories where the manipulator starts from rest and stops), (iii) a sensitivity model based on the method presented in Section 3.2.2.1 calculating the dependency of the end-effector position on changes of uncertain physical parameters. The restriction to start- and end-positions where the robot is in rest allows for a quasi-static approximation of the elasticity.

The goal is to provide a model detailed enough to assure a given positioning accuracy in the whole workspace of the robot. Physical parameters which optionally are subjected to identification and hence must be provided by the sensitivity model are uncertain geometrical dimensions, lengths and tilt angles of axes, errors from mounting the robot's base, elastic coefficients in the joints and the links which are treated as elastic beams. It should be noted that the model complexity can be adapted to the type of robot and the calibrated specimen to meet the precision requirements. Elastic effects which are not primarily of kinematic nature because they depend on the load and mass properties of the robot can be approximated

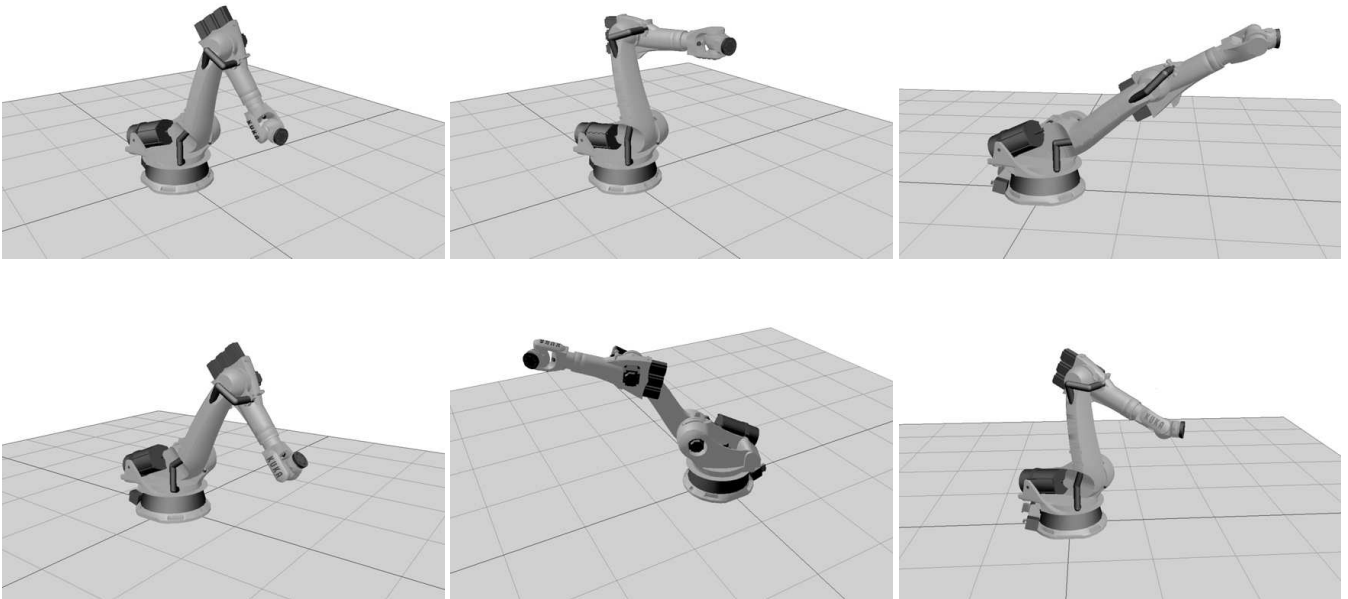


FIGURE 5.6: Selected ISO poses used for calibration of a KUKA KR2000 manufacturing robot. Pictures created by Matej Leskovar, by courtesy of AMATEC Robotics GmbH, Augsburg.

by using an iterative procedure. A calibration process can be described as follows:

- (i) **Measurement:** The specimen is installed in a measurement cell equipped with some camera system to detect the *absolute* position and orientation of the end-effector frame in Cartesian work-space. The robot is moved to a number of well-defined configurations where the nominal and the real end-effector frames are recorded. The goal is to 'excite' all parameters under consideration to allow for proper observability. Figure 5.6 shows a set of common, so-called ISO poses used for calibration.
- (ii) **Identification:** The errors between measured and computed end-effector position and orientation are formulated using a certain objective function as a minimization problem which can be solved for instance by standard least-squares methods.
- (iii) **Correction:** The identified parameters of the model are updated to the software and depending on the precision requirements the calibration is repeated iteratively or cyclically.

The accuracy (repeatability) of modern industrial robots is roughly several 0.1 mm for repetitive tasks. Absolute positioning accuracy in workspace is up to one order of magnitude larger due to non-modelled effects present in the real robot structure. The method sketched above can reduce the errors in absolute positioning effectively. The obtained accuracy lies in the order of magnitude of the repeatability for point-to-point motions.

A different application such as identification of the inertial properties of an unknown load would require a first- and higher-order inverse dynamics model in the mathematical parameter identification process. Schemes such as those presented in Section 3.2.2 can be applied to derive the robot models needed for this 'non-kinematic' calibration.

5.4 Trajectory optimization of legged robots

This section demonstrates how the methods of object-oriented modeling and advanced dynamics algorithms developed in this works can be applied to supply an optimizer package with all modeling information required for the complex task of trajectory optimization of legged robots satisfying the demand for efficient equations of motion and a great variety of boundary conditions.

5.4.1 Dynamics of legged robots

In mechanical and mathematical modeling of legged robots one is facing several challenging conditions,

- (i) many mechanical degrees of freedom and many actuated joints,
- (ii) time-varying contact conditions,
- (iii) tip-environment interaction,
- (iv) impact effects,
- (v) various stability regimes.

The first three topics also apply to multi-fingered robot hands, for instance. In this field grasping is a challenging problem, but often relies on a solely kinematical physical model neglecting all inertial effects. This simplification is no more adequate for legged robots where dynamics plays a dominant role in the motion, especially w. r. t. stability issues.

A gait cycle for one leg comprises a sequence of phenomena starting for instance from a swing-phase, tip impact, transition from impact to contact, contact during support phase with static friction, transition from static to sliding friction, sliding friction until leg lift-off followed by the next swing phase. Algorithms to consider the effect of leg collision and contact are discussed in [2, 4]. Detailed models describing the interaction between the foot of a walking robot and the ground have been studied for instance by Ouezdou et al. [105]. Dynamics modeling of legged robots with focus on trajectory optimization and gait stability has been investigated by Hardt [49] and Hardt and von Stryk [48].

The robots investigated in this section are modeled as full three-dimensional rigid MBS. Legged robots are described advantageously as free-flying systems experiencing contact forces to allow for the modeling of running gaits including phases with all legs in a flight phase and for the flexible application of various ground contact models. In this section feet-ground interaction is modelled by completely inelastic collision.

Figure 5.7 shows the two types of legged robots under consideration. First a humanoid prototype whose torso and two 6 jointed legs are modeled, leading to a number of 18 mechanical DOFs when modeled free-floating. Data for this prototype is presented in [24]. Second a model of a four legged Sony Aibo robot with 15 joints is investigated. For trajectory optimizations the head including three joints is omitted and the four-legged robot is reduced to a torso (6 DOFs) and four legs each having three mechanical DOFs, leading to a total of 18 mechanical DOFs.

From a mathematical point of view a free-floating system experiencing tip contacts leads to algebraical constraints reducing the number of mechanical DOFs of the robot and hence the number of independent states, which is a similar situation for kinematic loops. The numerical difficulties arising from this system of differential algebraic equations (DAEs) of high index, resulting from the general modeling approach

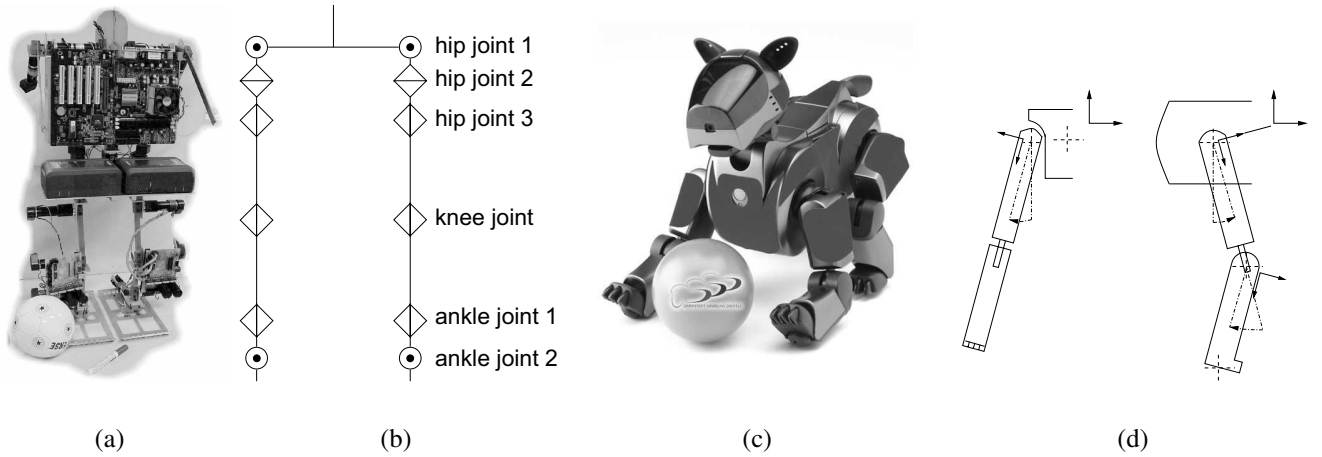


FIGURE 5.7: Legged robot examples. (a) The prototype of a humanoid robot [24]. (b) Kinematical structure of the two humanoid. (c) A four-legged Sony Aibo. (d) One 3 DOF leg of the AIBO model viewed from front and from the side. Please note, models currently used in optimizations consider just the torso and legs.

of multibody dynamics and algebraic equations for contact can be avoided in this case. On the one hand specialized numerical schemes can be applied for time integration of DAEs, for a survey and methods see [130]. On the other hand in this section the problems are avoided by a reduced dynamics method proposed in [49] where the system is transformed to state space form. This is often possible for mechanical systems such as legged robots if the inverse kinematics has a well-defined solution. Only the independent positional variables $q_I \in \mathbb{R}^{N_d - N_{cc}}$ are treated explicitly, which are global orientation and position and variables related to legs not in contact with the ground and explicit inverse kinematics solutions are exploited to determine the dependent states $q_D \in \mathbb{R}^{N_{cc}}$ of the other legs:

$$\begin{aligned} q_I &:= \text{global orientation, position; swing leg(s) states} \\ q_D &:= \text{contact leg(s) states} \end{aligned}$$

q_I may be computed from all states q using a constant mapping $Z \in \mathbb{R}^{(N_d - N_{cc}) \times N_d}$, i. e., $q_I = Zq$. The velocity variables are related by a partitioning of the velocity constraint equation (3.60) $\dot{q}_D = \mathcal{J}_{cD}^{-1} \mathcal{J}_{cI} \dot{q}_I$. The solution of the reduced dynamics can be shown to be of the form

$$\ddot{q}_I = Z \mathcal{M}(\bar{q})^{-1} (u - \mathcal{C}(\bar{q}, \dot{\bar{q}}) - \mathcal{G}(\bar{q}) + \mathcal{J}_c^T(\bar{q}) f_c), \quad (5.3)$$

where \bar{q} contains the independent states and the dependent states determined from inverse kinematics on position level. It can be shown that the solution of (5.3) is the solution of the original system of DAEs [49] for the legged robot MBS with contact. There are two main advantages in using the reduced dynamics approach. First the numerical optimization shows a better numerical convergence behaviour due to the reduced dimension of the state vector and a smaller number of equality constraints. On the other hand the re-use of existing efficient forward dynamics algorithms augmented by inverse kinematics and contact dynamics computations for this constrained mechanical system is computationally efficient.

5.4.2 Formulating the optimization problem

Finding stable gaits for walking robots with two or four legs is still challenging due to the high-complexity of the mechanical structure. Heuristic methods usually do not consider the full dynamics of the robot while on-line computation of walking trajectories is computationally extremely expensive ruling out an on-line calculation of optimal trajectories. In this section the problem of finding periodic and statically

or dynamically stable gaits is realized as an off-line task allows for application of full three-dimensional dynamics models in the computations.

Different gait patterns (such as walk, trot, rack, canter and rotary or transverse gallop for four-legged robots or walk and run for biped robots) differ in the duty factor of each foot, i.e. the fraction of a total stride cycle during which the foot is in contact with ground, and relative phases of feet, i. e., the order and time displacement of feet reaching ground [9].

The optimal control problem is formulated as follows [48]:

$$\begin{array}{ll}
 \min \mathcal{I}[q, \dot{q}, u, t_f] & \text{minimize the merit function } \mathcal{I} \text{ subject to} \\
 \mathcal{M}\ddot{q} = u - \mathcal{C}(q, \dot{q}) - \mathcal{G}(q) + \mathcal{J}_c^T f_c & \text{system of unconstrained MBS ODE} \\
 g_c(q) = 0 & \text{contact algebraic conditions} \\
 b(q_0, q_f, t_0, t_f) = 0 & \text{boundary conditions} \\
 n(q, u) \geq 0 & \text{nonlinear inequality constraints} \\
 q_{min} \leq q \leq q_{max}, \dot{q}_{min} \leq \dot{q} \leq \dot{q}_{max} & \text{box constraints on state} \\
 u_{min} \leq u \leq u_{max} & \text{and control variables.}
 \end{array}$$

Please note that in the optimization the reduced dynamics algorithm is applied. Common merit functions may be final time t_f , energy or a weighted sum of both [48]. Boundary conditions at initial and final time of a complete gait cycle or a proper part of it account for

- exploiting symmetry resp. anti-symmetry of states (see [48] for more details),
- foot placement, i. e., conditions that force the feet to be placed on desired positions, which may depend on parameters and therefore may also be subject to the optimization,
- contact forces at the end of a stance phase, that allow for foot lift-off.

Nonlinear inequality constraints applied are:

- Hips of legs in contact with the ground must stay within a maximum radius of the leg, so that the inverse kinematics solution required for reduced dynamics has a well-defined solution.
- The swing feet are ensured not to be lower than a given contour above the ground, for example a proper sine curve. This property increases stability by avoiding contact with the ground resulting from non-modelled deflexions of bodies and joints, and which could lead to stumbling of the robot in walking experiments.
- Slipping is avoided by limiting the horizontal contact forces relative to the vertical contact forces.
- Contact is unilateral, requiring vertical contact forces to be positive, i. e., the robot may only push to ground but may not pull from ground.
- Additional constraints considered in the problem formulation stem from motor characteristics. By now the box constraints for minimal and maximal values of angular velocities and torques only give a rough estimate of the real actuator data.

Statical or dynamical gait stability may be enforced explicitly by inserting any common criteria into the optimal control problem [48]. A more detailed discussion of the applied types of constraints can be found in [24] for a humanoid model and in [135] for a four-legged Sony Aibo.

The optimal control problem is solved by the direct optimization method DIRCOL [145]. The state and control variables are approximated by piecewise cubic respectively piecewise linear polynomials on a discrete and successively refinable time grid. The optimal control problem thus is transcribed into a non-linear program with the coefficients of the polynomials as variables, which—due to the special structure of the variables—can be solved by a sparse sequential quadratic programming method [45]. A more detailed discussion of applying these optimal control techniques to complex non-linear systems can be found in [49, 50, 145].

5.4.3 Object-oriented dynamics modeling for gait optimization

This section shows how the modeling requirements from the gait trajectory optimization problems can be fulfilled by the object-oriented framework presented in this work (cf. [58]). The developed methodology is very well-suited for integrating general purpose algorithms such as the articulated body algorithm as well as the model- or problem-specific schemes. For instance computing the reduced dynamics relies on explicit inverse kinematics algorithms depending on specific properties of the model description. Especially the rich variety of boundary conditions describing certain constraint or stability regimes can be described and implemented uniformly without losing computational efficiency.

The dynamic gait trajectory optimization algorithm requires the set of equations of motion of the robotic system over one *complete* gait cycle, i. e., including time-varying support phases reflected by a changing dynamics model incorporating the structural changes. The treatment of a complete gait cycle as a multi-phase problem requires additional collision computations to model the discontinuous state transitions between various support phases. Depending on the applied optimization method derivatives of the equations of motion w. r. t. state variables and design parameters might be required. When trajectory optimization is subject to additional gait stability criteria new types of computations must be introduced to determine, e. g., center of mass and zero moment point of the legged robot in its current state.

Originally, the legged robot dynamics has been provided by a set of hand-coded Fortran routines (SOAFOR) for the reduced dynamics etc. developed by Hardt [49] and later developments. Applying the concepts from Chapter 4 the complete object-oriented integrated model for trajectory optimization was realized using the generalized multibody entity paradigm introduced in Section 2.2 instead of the notion of links, applying the Port-Based Spatial Operator Algebra for establishing the component dynamics equations, using the dataflow model of computation presented in Section 2.3 for all involved recursive algorithms, and using C++ as basis for the complete implementation. The resulting package structure is shown in Figure 5.8.

The DIRCOL optimizer interface comprises three main blocks depending on the system under investigation: (i) the differential equation, (ii) the linear- and non-linear boundary conditions, and (iii) phase transition behaviour when switching the phases between support phases. This includes symmetry conditions to be fulfilled between start and end states when treating periodic solutions. For the humanoid walking gait this is a change between single- and double support phase and vice versa. In our realization each requirement is satisfied by a set of coupled Solver objects.

The model parts of the first package 'Differential equations' are implemented by one solver containing the $\mathcal{O}(N)$ articulated body algorithm for free-floating legged robots discussed in Section 3.1.3, a collection of hand-tailored inverse kinematics solvers for each single leg and Jacobian solvers for the velocity inverse kinematics. The contact state is represented by a solver depending on the time-varying contact

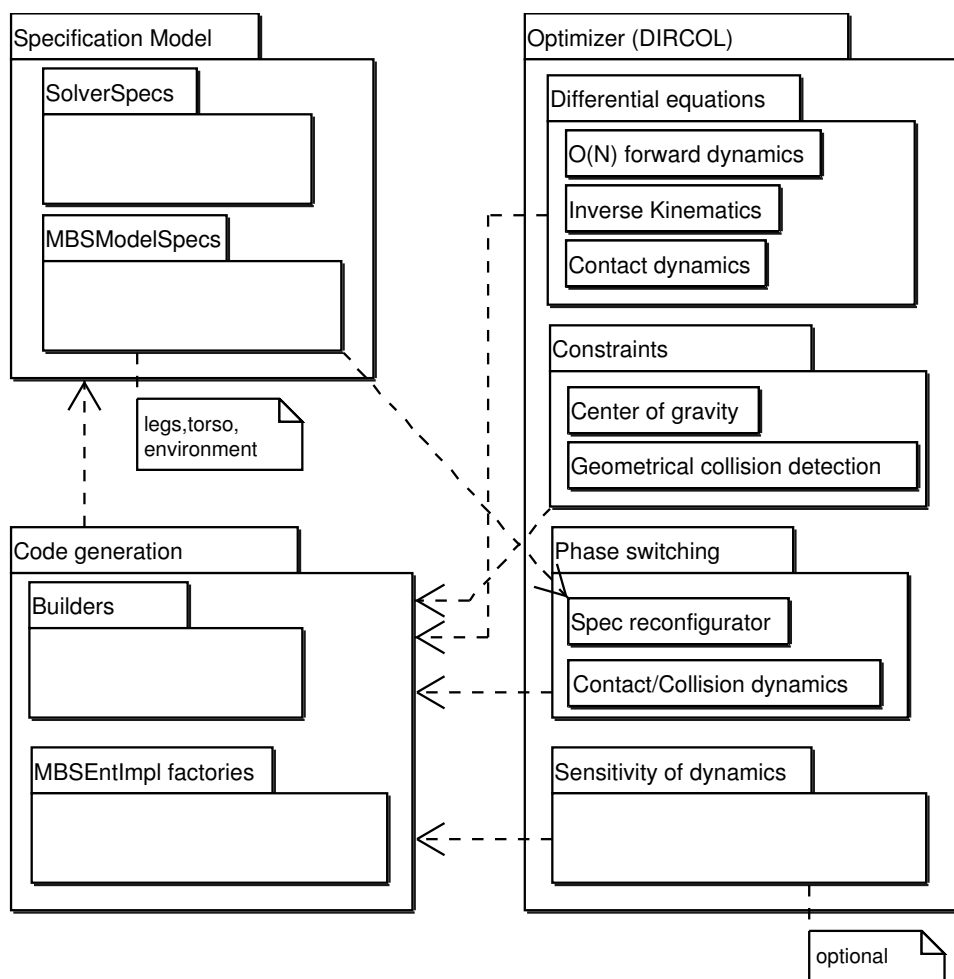


FIGURE 5.8: UML package diagram of gait trajectory optimization within DIRCOL using Spec-, Builder-, and Solver objects.

model specification and is used for the reduced dynamics. Interestingly, the collision algorithm from Section 3.2.4 can be generally applied to compute for inverse velocity kinematics avoiding establishing and inverting explicitly the foot Jacobian. A fact easily exploited in the developed modular object-oriented architecture.

The second package 'Constraints' represents the linear and non-linear equality and inequality constraints which are crucial during optimization to generate a useful solution. This package contains solvers which compute the center of gravity of the complete mechanical system, and which detect foot collisions with the ground and between legs.

The task of the third package 'Phase switching' is to react to requests from the trajectory optimization method DIRCOL to change the support phase by invoking the collision Solver, a realization of the collision method presented in Section 3.2.4. The contact state is changed and the system of solvers reconfigured accordingly to keep all components consistent.

DIRCOL is able to process user-defined gradient information. In order to enhance convergence properties an optional set of solvers provides exact sensitivities of the differential equations. All solvers mentioned are created and reconfigured by a collection of builder objects from a unique high-level specification of all required aspects of the mechanical model. These aspects are descriptions of single legs, of the complete legged mechanism or the way the mechanism interacts with the environment.

5.4.4 Optimized trajectories

The architecture presented in the preceding section allows for trajectory optimizations of almost arbitrary legged robots, which is based on a fully object-oriented description and implementation of the robot model. This allows for flexible application of drivetrain and contact models, and application of appropriate algorithms including dedicated closed form inverse kinematics solutions for the legs and efficient reduced dynamics algorithms. Walking trajectories involving full three-dimensional models have been calculated both for a MBS model of a four-legged robot by Stelzer et al. [135] and results for a two legged humanoid robot are discussed in [24].

The walking sequence shown in Figure 5.9 is a slow optimized *periodic* gait of the two-legged torso with two support phases taking 6 s. A free-flying rigid-body model of the torso and two 6 DOF legs was applied for optimizations of a two-phase half-stride consisting of a single limb support phase and a double limb support phase. Static stability was enforced explicitly by nonlinear inequality constraints. This can be seen from the snapshots, where most time the center of gravity obviously is located above the support leg. The merit function which was minimized is

$$\mathcal{I}[q, \dot{q}, u, t_f] = \int_0^{t_f} (u^T u + \dot{q}^T \dot{q}) dt, \quad (5.4)$$

which is a combination between the common squared actuator torque measure, often referred to as *energy*, and the squared joint rates. The objective to use this merit function was to achieve a very slow trajectory considering all constraints to test the real humanoid prototype shown in Figure 5.7 in first experiments using trajectory following joint control. The average velocity of the resulting trajectory was roughly 0.06 m/s. DIRCOL transformed the optimization problem for the half-stride into a non-linear program on 63 grid points, resulting in 2176 variables, 1479 nonlinear equality constraints, 300 nonlinear inequality constraints, 1 linear inequality constraint. Please note the gait pattern itself was prescribed by the user and is not a result obtained from the optimization. The evaluation of one reduced dynamics computation was roughly 1.6 ms on a 2 GHz Pentium IV Personal Computer including exact analytical sensitivities during the smooth phases of the trajectory.

The efficient and reliable optimization of trajectories for torsos with legs is the prerequisite for investigation of more realistic, hence detailed, systems. Including the dynamics of articulated arms especially in two legged locomotion leads to more realistic 'human-like' motion. The presented framework will pay off in in such an application in several ways: The specification model does not put much restriction on the description allowing for general topologies and components. The builder/solver pairs proposed in the object-oriented modeling architecture in Section 4.2.1 allow for integration of new explicit inverse kinematics reducing the computational burden and increasing robustness. Biomechanical systems require more sophisticated actuator and optimization techniques. Biological systems are greatly overactuated systems, when considering muscle groups separately in a model. As a consequence the control variables will be stimuli of the muscles and not the joint actuator torques—actually the joints themselves are completely passive. These stimuli are parameters describing chemical potentials in biologically inspired muscle models. Compared to the conventional electrically or hydraulically driven robot *joints* this leads to completely new actuation concepts, which has to be reflected by the numerical methods and the operational robot model, too. These additional requirements can easily be handled by the methodology developed in this work.

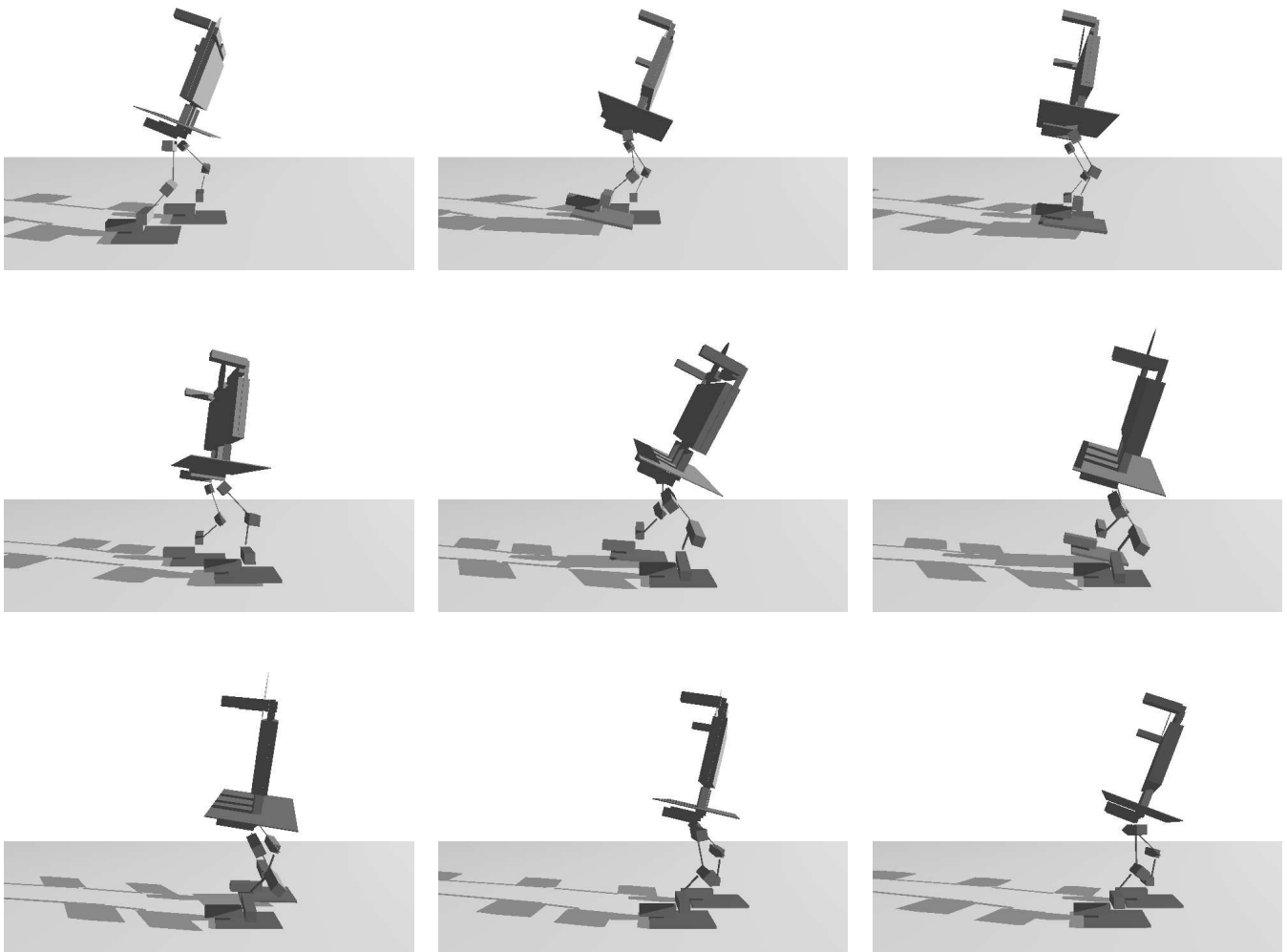


FIGURE 5.9: Sequence of snapshots of a periodic two-phase gait of a humanoid-prototype. The gait is optimal w. r. t. to merit function (5.4) and takes roughly 6 s. Optimization and animation by Michael Hardt, Max Stelzer, and Martin Friedmann, TU Darmstadt.

Summary

In this work a *general purpose robot dynamics methodology* was presented to support robot control design engineers in (i) robot model specification, (ii) algorithm implementation by automatic code generation *and* manual coding, and (iii) integration of the computational models and components in an evolving software architecture for robot control.

Nowadays complex computer-controlled systems increasingly rely on the use of models representing both hard- and software. Fundamental components of every software-based robot control system are executable models reflecting the physical state of the robot, first of all the motion of the robot in its work-space. In the growing robotics domain with its increasing demand for efficiency, flexibility, low cost, short time-to-market, and safety, the integration of all parts becomes a key issue. The chosen 'glue' between the various conceptual models is a new developed abstract description of the mechanical robot model. This includes the classical perspective on multibody models as interacting bodies and mechanical devices. The new extension is a behavioural description of the components, external conditions, *and* the desired computations. The algorithms are described symbolically by means of the new Port-Based Spatial Operator Algebra, extending an existing multibody formalism to object-oriented concepts.

The presented methodology includes a carefully designed new class hierarchy, which allows for integration of the immense variety of numerical robot dynamics algorithms developed by domain engineers during the last decades. The focus is on recursive dynamics algorithms, known to be very efficient and flexible. Among the investigated algorithms are standard and new dynamics methods to treat special closed-loop mechanisms automatically and the dynamics of elastic joint robots. Its derivation reveals explicit symbolical operator identities invaluable for theoretical investigation as well as efficient code generation. The backbone of the model of computation is the new concept of algorithm-specific code-generators, called builders, creating executable instances, called solvers, which perform the desired multibody computations. This leads to efficient and light-weight code without a compile-to-code step required. Builder/solver pairs are able to wrap dedicated problem-specific algorithms and efficient closed-form solutions indispensable in robotics in order to reap maximum performance. To ameliorate code reuse and integration without sacrificing readability the methodology provides new well-defined interfaces for solvers and dedicated objects, capturing the multibody equations.

The presented applications document how implementation and integration of new and existing algorithms and the collaboration of executable algorithm blocks are alleviated by this framework. Tackled problems are dynamics computations required in joint control, robot calibration, and gait optimization of legged robots.

Appendix A

Glossary

ABA Abbrev. for Articulated body algorithm [39].

Absolute Any mechanical quantity represented by a vector or dyadic whose definition is tied to an inertial reference is called absolute[115].

Action encompasses the two terms *force* and *torque* [115].

Application generator is a software to translate a specification into application programs, operating much like a language compiler [27].

Articulated body inertia The spatial inertia of an MBS when viewing it as a ‘sloppy‘ chain of material bodies interconnected by non-actuated joints.

Base body A body fixed in inertial space[149].

Change of basis ${}^{(A)}\mathbf{b}_i$ are base vectors of vector basis B represented w. r. t. the vector basis A . The vector \mathbf{x} represented w. r. t. B ${}^{(B)}\mathbf{x} = (x_1, x_2, x_3)$ is when represented w. r. t. system A of the form

$${}^{(A)}\mathbf{x} = \left({}^{(A)}\mathbf{b}_1 {}^{(A)}\mathbf{b}_2 {}^{(A)}\mathbf{b}_3 \right) \cdot \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix},$$

because the coordinates ${}^{(B)}x_i$ are the projections of \mathbf{x} on the basis vectors of B . Identifying A and B with F_A and F_B , the transformation matrix is a rotation ${}^Bx = {}^B R_A {}^A x$. Remark: The columns of the transformation matrix ${}^B R_A$ are the basis of B resolved w. r. t. to A .

C.M. Abbrev. for *center of mass*, symbolized by index c .

C.M.S. Abbrev. for *center of mass system*, the symbol mostly used in this work is frame F_c .

Constraint Denotes the restriction of the relative motion of two or more bodies. Constraints lead to algebraical relations between position and velocity variables describing the interacting bodies.

holonomic A constraint function $f_c(q, t)$ is holonomic if there exists a total differential $df_c = \nabla_x f_c + \frac{\partial f_c}{\partial t}$, i. e. f_c can be determined by means of integration.

ideal The constraint force do no work during virtual displacements.[149]

nonholonomic For a constraint function f_c there does not exist a total differential df_c , i. e. f_c can be determined by means of integration. Its treatment requires distinguishing between variables of position and motion.

rheonomic A constraint which depends explicitly on time $f_c = f_c(q(t), t)$

scleronomic A constraint which does *not* depend explicitly on time $f_c = f_c(q(t))$

Contiguous Two bodies are contiguous, if and only if they exert force on each other *directly* [149].

DAE Abbrev. for *differential algebraic equation*.

DFS Abbrev. of depth-first-search, one possibility for graph traversal [5].

DOF Abbrev. for *degree of freedom*, in the context of robotics the number of mechanical degrees of freedom, often related to the number of driven joints

Drive A technical component which transforms an input quantity into a mechanical quantity, mainly position/force/moment. A rotational drive at least has one rotor, one stator and one unit to transform energy.

Drivetrain The unit of a drive and connected mechanical components which transform the drive's output quantity and transport it to location of application, e. g. where a motor moment is required. It may be identified with a technical device mounted for instance to the robot.

Dyadic A Cartesian tensor of second rank.

Endeffector One prominent location or frame designated on a robot manipulator. Usually the part where tools can be mounted or interaction to other manipulators takes place.

Entity An entity e is defined as a 'thing' which can be distinctly identified [26].

Entity set Entities are classified into different entity sets E_i . There exists a predicate associated with each entity set to test whether an entity e_i belongs to it [26].

Euclidean space The 3-dimensional space used in classical mechanics.

Euler parameters Normalized version of quaternions.

Force element Element or device which produces external forces and torques on Multibody system components.

Formalisms, dynamical Methods for the generation and implementation of equations of motion of multibody systems. A comparison of various formalisms can be found in [115].

Formalism, modeling A formalism imposes certain syntactic rules, in a way similar to a type. Describes all legal models within the formalism.

Frame A coordinate system (orientation) given by 3 base vectors ${}_x\mathbf{e} = \{{}_x\mathbf{e}_1, {}_x\mathbf{e}_2, {}_x\mathbf{e}_3\}$ attached to a point location \mathbf{O}_x . More precisely the tuple $\{\mathbf{O}_x, {}_x\mathbf{e}\}$. Used symbol throughout this work is F_x .

Framework A certain type of object-oriented software reuse technique.

(i) The *structure* of a framework is described in [69] as:

A framework is a reusable design of all or part of a system that is represented by a set of abstract classes and the way their instances interact.

(ii) Its *purpose* is defined there [69] by:

A framework is the skeleton of an application that can be customized by an application developer.

Free-floating base A robot with a free floating base is one whose base member is free to move so that its acceleration depends on the motion of the rest of the robot [40].

Gyrostad A mechanical system which behaves exactly like a rigid body except for an internal angular momentum, for instance stemming from a rotating axial-symmetric mass [149].

Hinge The totality of interaction forces between *one* pair of bodies [149].

Inertial frame A non-accelerated frame, sometimes called *Newtonian frame*.

Interface Is as class without implementation and therefore has operation *declarations* but no method bodies and no fields. They are often declared through abstract classes. It defines the responsibility of a class.

IP Abbrev. for *Interaction Port*, introduced in Section 2.2.

Jacobian In robotics the mapping from joint rates to the spatial motion of an endeffector [134].

Joint see *kinematic pair*.

Mass matrix Matrix containing all parts proportional to \ddot{q} in state-space formulation of equations of motion of a Multibody system. Used symbol is \mathcal{M} .

Matrix An array of numbers. This work adopts the convention of column-matrices for one-dimensional matrices.

MBS Abbrev. for *multibody system*.

ME Abbrev. for *MBS entity* or *multibody entity*.

Message One member in the set of a protocol.

Modeling Act of abstraction of a real-world system to capture relevant features of this system in a certain context.

Multibody entity A term introduced in Section 2.2.1 to denote the concept of a component of a robotic multibody system.

Multibody system A collection of material bodies interacting through joints and force elements.

OO Abbrev. for *object oriented*.

OOP Abbrev. for *object oriented programming*.

Ontology An ontology is an explicit specification of some topic. For our purposes, it is a formal and declarative representation which includes the *vocabulary* (or names) for referring to the terms in that subject area and the *logical statements* that describe what the terms are, how they are related to each other, and how they can or cannot be related to each other. Ontologies therefore provide a vocabulary for representing and communicating knowledge about some topic and a set of relationships that hold among the terms in that vocabulary.

Operational semantics A set of rules specifying how the state of an actual or hypothetical computer changes while executing a program. The overall state is typically divided into a number of components, e.g. stack, heap, registers etc. Each rule specifies certain preconditions on the contents of some components and their new contents after the application of the rule.

Operational space A term coined by Khatib [78, 79] to emphasize the view on robotic motion in six dimensional space of rigid body motions $SE(3)$ rather than in joint (state) space, i. e., the movement of a certain frame, e. g., the tool of a serial link robot in *operational space*.

OS Abbrev. for *operational space*.

Pair, kinematic A joint constraining the *relative* motion of two bodies.

Lower pair Joints with surface contact, one member of set of joints {revolute, prismatic, screw or helical, cylindrical, spherical, planar}

Higher pair Joints with line or point contact.

Path Take any two bodies of a MBS. Proceed from one body to the other along a sequence of bodies and hinges in such a way that no hinge is passed more than once. The set of hinges defined this way is called path [149].

Pattern A software pattern is a certain type of a object-oriented software reuse technique and could be defined as

a proven, non-obvious, and constructive solution to a common problem in a well-defined context, taking into account interactions and trade-offs ("forces") that each tend to drive the solution into different directions. A Pattern describes the interaction between a group of components, hence it is a higher-level abstraction than classes or objects [44].

A pattern describes a problem to be solved, a solution and the context in which that solution works. It names a technique and describes its costs and benefits.

Patterns are less specialized than frameworks [69]; frameworks have a particular application domain, while patterns should be more generally applicable.

Persistence A property of a programming language where created objects and variables continue to exist and retain their values between runs of the program.

Point-to-point motion A motion pattern where a robot manipulator starts from rest, moves along a prescribed trajectory and stops completely.

Principle A statement on the nature of constraint actions in the case of constrained motion of a multibody system, because the laws of Newton and Euler are not sufficient to describe constrained motion of a rigid body. The main principles are

D'Alembert's, where the virtual actions vanish,

Jourdain's, where the virtual power vanishes,

Gauß's, obeying the least constraint.

Projection operator Idempotent operators, i. e. those satisfying the condition $\hat{P}^2 = \hat{P}$.

Protocol An abstract specification of desired behaviour in the conveying signals (messages) from one port to another [127].

PSOA Abbrev. for Port-Based Spatial Operator Algebra, introduced in Section 2.3.

Quaternion The set of quaternions

$$Q := q_0 + q_1\mathbf{i} + q_2\mathbf{j} + q_3\mathbf{k}$$

is a 4-dimensional vector space over \mathbb{R} and forms a group w. r. t. the quaternion multiplication "·"

$$Q \cdot P = (q_0p_0 - \vec{q} \cdot \vec{p}, q_0\vec{p} + p_0\vec{q} + \vec{q} \times \vec{p})$$

The $\mathbf{i}, \mathbf{j}, \mathbf{k}$ obey the commutation relations, q_0 is called the *scalar* component and the 3-vector $\vec{q} := (q_1, q_2, q_3)$ is called the *vector* component. Given a rotation matrix $R := e^{\omega\theta}$ (ω unit vector $\in \mathbb{R}^3$ and $\theta \in \mathbb{R}$) we can define an associated *unit quaternion*

$$Q := \left(\cos\left(\frac{\theta}{2}\right), \omega \sin\left(\frac{\theta}{2}\right) \right)$$

whose multiplication directly corresponds to multiplication on $SO(3)$. The inverse is given by

$$\theta = 2 \cos^{-1} q_0 \quad \omega = \begin{cases} \frac{\vec{q}}{\sin(\theta/2)} & \theta \neq 0 \\ 0 & \text{otherwise} \end{cases}$$

[98]

RCS abbrev. for *Robot Control System*, denoting the complete hard- and software controlling the robot hardware.

Real-time system A system where timeliness is a common and dominant feature. Crucial are strict timing and synchronization requirements Hatley and Pirbhai [51].

Recursive algorithm In the field of Multibody systems a sloppy expression for *iterative algorithm*, exploiting the sparse structure of multi-body equations for very efficient solution.

Reference node One vertex in a MBS graph representation which is (declared) special. In a directed tree structured system it is the root vertex, or for looped systems it acts as a more or less arbitrary starting point.

Reference frame The frame acting as a reference for kinematical and/or dynamical values. It is *not* necessarily an inertial frame.

Relationship An association between two entities[26].

Relationship set A relationship set R_i is a mathematical relation among n entities, each taken from an entity set $\{[e_1, \dots, e_n] | e_i \in E_i\}$ and each tuple $[e_1, \dots, e_n]$ is a relationship [26].

RNE Abbrev. for *Recursive Newton-Euler algorithm*.

Role The role of an entity in a relationship is the function that it performs in the relationship [26].

Rotation matrix A matrix representation of an element of the group $SO(3)$.

Screw motion A rigid body motion which consists if a rotation around an axis l in 3D space by an angle θ followed by a translation along that axis by an amount d . Definitions:

A screw motion corresponds to motion along a constant twist by an amount equal to the magnitude of the screw [98]

Axis $l := \{q + \lambda\omega \mid \lambda \in \mathbb{R}\}$, $q \in \mathbb{R}^3$, $\omega \in \mathbb{R}^3$, $\|\omega\| = 1$

Pitch $h := \frac{d}{\theta}$, if $\theta \neq 0$

Magnitude $M :=$

Associated twist $\xi = (-\omega \times q + h\omega, \omega)$, ($\theta \neq 0$)

Semantics Meaning of a string in some language, as opposed to syntax which describes how symbols may be combined independent of their meaning. The two main kinds are denotational semantics and operational semantics.

SOA Abbrev. for *Spatial Operator Algebra*.

Software engineering A systematic approach to the analysis, design, implementation and maintenance of software. It often involves the use of CASE tools. There are various models of the software life-cycle, and many methodologies for the different phases.

Spatial Operator Algebra A framework developed by [117] to express recursive multibody algorithms symbolically in closed form by means of so-called spatial operators which reflect certain physically properties of the multibody system.

Structured methods Methods consisting of modeling tools and techniques that illuminate certain aspects of the desired system during the specification process [51].

Sweep A single recursion along a kinematic chain or tree structure in either direction.

Syntax The structure of strings in some language. A language's syntax is described by a *grammar*.

System A systematic grouping of components put together to behave as a whole [51].

Twist Elements of generator group $se(3)$. [98] Every rigid transformation can be written as exponential of some twist $\hat{\xi} \in se(3)$, i. e. there exists a (surjective) mapping from $se(3)$ to $SE(3)$. Screw coordinates of a twist with twist coordinates $\xi = (v, \omega) \in \mathbb{R}^6$:

Pitch $h = \frac{\omega^T v}{\|\omega\|^2}$, if $\omega = 0$ then $\xi \equiv \infty$

Axis $l = \begin{cases} \frac{\omega \times v}{\|\omega\|^2} + \lambda\omega : \lambda \in \mathbb{R} & \omega \neq 0 \\ 0 + \lambda v : \lambda \in \mathbb{R} & \omega = 0 \end{cases}$

Magnitude $M = \begin{cases} \|\omega\| & \omega \neq 0 \\ \|v\| & \omega = 0 \end{cases}$

Vector A Cartesian tensor of first rank.

Wrench Element of the generator group $se^*(3)$.

XML Abbrev. for *Extensible Markup Language*, see <http://www.w3.org/XML/>.

Appendix B

Useful identities

Rotation parametrization

Rotation about an arbitrary axis ${}^1u = {}^2u$ by an angle θ results in a direction cosine matrix [115]

$${}^1R_2 = uu^\top + \cos \theta (I_{3 \times 3} - uu^\top) + \sin \theta \tilde{u} \quad (\text{B.1})$$

which is strongly related to Rodriguez formula¹⁾

$$e^{\tilde{\omega}\theta} = I_{3 \times 3} + \tilde{\omega} \sin \theta + \tilde{\omega}^2 (1 - \cos \theta) \quad \|\omega\| = 1, \theta \in \mathbb{R} \quad (\text{B.2})$$

Tilde identities

This section presents some definitions and useful identities concerning the tilde operator for column vectors $\in \mathbb{R}^3$ and $\in \mathbb{R}^6$. The tilde operator for $p \in \mathbb{R}^3$ is defined as

$$\tilde{p} := \begin{pmatrix} 0 & -p_z & p_y \\ p_z & 0 & -p_x \\ -p_y & p_x & 0 \end{pmatrix}. \quad (\text{B.3})$$

For three dimensional column vectors $p, q \in \mathbb{R}^3$ one can show

$$\tilde{p}p = 0_3 \quad (\text{B.4})$$

$$\tilde{p}^\top = -\tilde{p} \quad (\text{B.5})$$

$$\tilde{p}q = -\tilde{q}p \quad (\text{B.6})$$

(i)

$$\widetilde{(a+b)}q = \tilde{a}q + \tilde{b}q \quad (\text{B.7})$$

$$\widetilde{(\tilde{p}q)} = \tilde{p}\tilde{q} - \tilde{q}\tilde{p} \quad (\text{B.8})$$

$$\tilde{p}\tilde{q} = qp^\top - p^\top q I_{3 \times 3} \quad (\text{B.9})$$

¹⁾for a nice derivation see [98].

The *spatial cross product* is defined by

$$X \times Y := \begin{pmatrix} a \\ b \end{pmatrix} \times \begin{pmatrix} c \\ d \end{pmatrix} = \begin{pmatrix} \tilde{a}c \\ \tilde{b}c + \tilde{a}d \end{pmatrix} =: \tilde{X}Y \text{ where } \tilde{X} := \begin{pmatrix} \tilde{a} & 0_{3 \times 3} \\ \tilde{b} & \tilde{a} \end{pmatrix} \quad (\text{B.10})$$

Please note \tilde{X} is *not* skew-symmetric as its 3 dimensional brother, except for the case $b \equiv 0_3$. For vectors in spatial notation the following identities can be shown [49]:

$$\tilde{X}X = 0_6 \quad (\text{B.11})$$

$$\tilde{X}Y = -\tilde{Y}X \quad (\text{B.12})$$

$$\widetilde{(A+B)} = \tilde{A} + \tilde{B} \quad (\text{B.13})$$

$$\widetilde{(\tilde{X}Y)} = \tilde{X}\tilde{Y} - \tilde{Y}\tilde{X} \quad (\text{B.14})$$

$$\tilde{X}\tilde{X}\tilde{X} = \tilde{X} \begin{pmatrix} -a^\top a I_{3 \times 3} & 0_{3 \times 3} \\ -2b^\top a I_{3 \times 3} & -a^\top a I_{3 \times 3} \end{pmatrix} \quad (\text{B.15})$$

The spatial co-cross product operator is defined by

$$\tilde{X}^\otimes := \begin{pmatrix} \tilde{c} & \tilde{d} \\ \tilde{d} & 0_{3 \times 3} \end{pmatrix} \quad (\text{B.16})$$

where $X := \begin{pmatrix} c \\ d \end{pmatrix} \in \mathbb{R}^6$. The following important identity relates the spatial cross product to the co-cross product

$$\tilde{X}^\otimes Y = \tilde{Y}^\top X$$

Bibliography

- [1] M. Abadi and L. Cardelli. *A theory of objects*. Monographs in computer science. Springer Verlag, Berlin, Heidelberg, New York, Tokyo, 1996.
- [2] D. Agahi and K. Kreutz-Delgado. A star topology dynamic model for efficient simulation of multilimbed robotic systems. In *Proc. IEEE Conf. on Robotics and Automation*, pages 352–357, 1994.
- [3] D. Agahi and K. Kreutz-Delgado. Approximate dynamic decoupling of multilimbed robotic systems. In *Proceedings of the 1995 IEEE/RSJ International Conference on Intelligent Robots and Systems*, volume 3, pages 480 – 485, Aug. 1995.
- [4] D. Agahi and K. Kreutz-Delgado. A star topology dynamic model for multipedal locomotion. In *Proc. IEEE Conf. on Decision and Control*, pages 4868–4873, San Diego, USA, 1997.
- [5] A. V. Aho, J. E. Hopcroft, and J. D. Ullman. *The design and analysis of computer algorithms*. Addison Wesley Publishing Company, 1974.
- [6] A. Albu-Schäffer. *Regelung von Robotern mit elastischen Gelenken am Beispiel der DLR-Leichtbauarme*. PhD thesis, Technische Universität München, 2002.
- [7] A. Albu-Schäffer and G. Hirzinger. State feedback controller for flexible joint robots: A globally stable approach implemented on DLRs light-weight robots. In *Proc. IEEE Conf. on Intelligent Robots and Systems*, Takamatsu, Japan, 2000.
- [8] J. Aldrich, V. Sazawal, C. Chambers, and D. Notkin. Language support for connector abstractions. In L. Cardelli, editor, *17th European conference on object oriented programming*, pages 74–102, Darmstadt, Germany, July 2003. Springer Verlag.
- [9] R. M. Alexander. The gaits of bipedal and quadrupedal animals. *The Int. J. Robotics Res.*, 3(2): 49–59, 1984.
- [10] K. S. Anderson and J. H. Critchley. Improved order-N performance algorithm for the simulation of constrained multi-rigid-body dynamic systems. *Multibody Systems Dynamics*, 9(2):185–212, 2003.
- [11] R. J. Anderson. SMART: a modular architecture for robotics and teleoperation. In *Proc. IEEE Conf. on Robotics and Automation*, pages 416 – 421, May 1993.
- [12] W. W. Armstrong. Recursive solution to the equations of motion of an N-link manipulator. In *Proceedings of the fifth world congress on theory of machines and mechanisms*, volume 2, pages 1343–1346. American society of mechanical engineers, 1979.
- [13] U. M. Ascher, D. K. Pai, and B. P. Cloutier. Forward dynamics, elimination methods, and formulation stiffness in robot simulation. *The Int. J. Robotics Res.*, 16(6):749–758, 1997.
- [14] K. Beck. *Extreme Programming Explained*. Addison Wesley Publishing Company, 1999.
- [15] D. Bestle. *Analyse und Optimierung von Mehrkörpersystemen*. Springer Verlag, 1994.
- [16] J. J. Biesiadecki, D. A. Henriquez, and A. Jain. A reusable, real-time spacecraft dynamics simulator. In *6th Digital Avionics Systems Conference*, pages 8.2.8–8.2.14, Irvine, CA, 1997.
- [17] D. W. Binkley. C++ in safety critical systems. Technical report, NIST, Nov. 1995.
- [18] J. E. Bobrow, S. Dubowsky, and J. S. Gibson. Time-optimal control of robotic manipulators along specified paths. *The Int. J. Robotics Res.*, 4(3):3–17, 1985.

- [19] W. J. Book. Recursive Lagrangian dynamics of flexible manipulator arms. *The Int. J. Robotics Res.*, 3(3):87–101, 1984.
- [20] F. Boyer and W. Khalil. Simulation of flexible manipulators using Newton-Euler inverse dynamic model. In *Proc. IEEE Conf. on Robotics and Automation*, pages 1947 – 1952, Apr. 1996.
- [21] H. Brandl, R. Johanni, and M. Otter. A very efficient algorithm for the simulation of robots and similar multibody systems without the inversion of the mass matrix. In *IFAC/IFIP/IMACS Symposium on robotics*, Wien, Austria, 1986.
- [22] H. Bruyninckx. Open robot control software: The OROCOS project. In *Proc. IEEE Conf. on Robotics and Automation*, pages 2523–2528, Seoul, Korea, May 2001.
- [23] J. T. Buck, S. Ha, E. A. Lee, and D. G. Messerschmitt. Ptolemy: A framework for simulating and prototyping heterogeneous systems. *Int. Journal of Computer Simulation*, 4:155–182, Apr. 1994. special issue on Simulation Software Development.
- [24] M. Buss, M. Hardt, J. Kiener, M. Sobotka, M. Stelzer, O. von Stryk, and D. Wollherr. Towards an autonomous, humanoid, and dynamically walking robot: Modeling, optimal trajectory planning, hardware architecture, and experiments. In *Third IEEE International Conference on Humanoids Robots, Technische Universität München and Universität Karlsruhe, Germany, October 1-3, 2003*.
- [25] I.-M. Chen and G. Yang. Automatic model generation for modular reconfigurable robot dynamics. *Transactions of the ASME*, 120:346–352, Sept. 1998.
- [26] P. P.-S. Chen. The entity-relationship model—toward a unified view of data. *ACM Trans. Database Syst.*, 1(1):9–36, 1976.
- [27] J. C. Cleaveland. Building application generators. *IEEE Software*, 4(5):25–33, July 1988.
- [28] P. I. Corke. A robotics toolbox for MATLAB. *IEEE Robotics and Automation Magazine*, 3(1): 24–32, Mar. 1996.
- [29] J. J. Craig. *Introduction to robotics: mechanics and control*. Addison Wesley Publishing Company, 2nd edition, 1989.
- [30] D. E. Orin and William W. Schrader. Efficient computation of the Jacobian of robot manipulators. *The Int. J. Robotics Res.*, 3(4):66–75, 1984.
- [31] A. De Luca. Feedforward/feedback laws for the control of flexible robots. In *Proc. IEEE Conf. on Robotics and Automation*, pages 233–240, San Francisco, CA, USA, Apr. 2000.
- [32] A. De Luca and P. Lucibello. A general algorithm for dynamic feedback linearization of robots with elastic joints. In *Proc. IEEE Conf. on Robotics and Automation*, pages 504–510, Leuven, Belgium, May 1998.
- [33] A. De Luca and P. Tomei. Elastic joints. In C. C. de Wit, B. Siciliano, and G. Bastin, editors, *Theory of robot control*, pages 179–217. Springer Verlag, Berlin, Heidelberg, New York, Tokyo, 1996.
- [34] J. Denavit and R. S. Hartenberg. A kinematic notation for lower-pair mechanisms based on matrices. *ASME Journal of applied mechanics*, pages 215–221, June 1955.
- [35] H. Dorr. *Efficient Graph Rewriting and its implementation*. Lecture Notes in Computer Science, 922. Springer Verlag, 1995.
- [36] H. Ehrig and G. Taentzer. Computing by graph transformation: A survey and annotated bibliography. *BEATCS: Bulletin of the European Association for Theoretical Computer Science*, 59, 1996.
- [37] G. Engels and L. Groenewegen. Object-oriented modeling: a roadmap. In *Proceedings of the conference on The future of Software engineering*, pages 103–116. ACM Press, 2000.
- [38] G. Engels, R. Heckel, G. Taentzer, and H. Ehrig. A view-oriented approach to system modelling based on graph transformation. In *Proceedings of the 6th European conference on software engineering*, pages 327–343, 1997.

- [39] R. Featherstone. The calculation of robot dynamics using articulated-body inertias. *The Int. J. Robotics Res.*, 2(1):13–30, 1983.
- [40] R. Featherstone. *Robot dynamics algorithms*. Kluwer Academic Publishers, Boston, Dordrecht, Lancaster, 1987.
- [41] R. Featherstone. The acceleration vector of a rigid body. *The Int. J. Robotics Res.*, 20(11):841–846, 2001.
- [42] M. Fischer and M. Hörmann. Specification: ConnectionFactory. private communication, 2002.
- [43] M. Fowler and K. Scott. *UML Distilled: A brief guide to the standard object modeling language*. Addison Wesley Publishing Company, 2 edition, 1999.
- [44] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns - Elements of Reusable Object-Oriented Software*. Addison Wesley Publishing Company, 1994.
- [45] P. Gill, W. Murray, and M. Saunders. SNOPT: An SQP algorithm for large-scale constrained optimization. *SIAM Journal on Optimization*, 12:979–1006, 2002.
- [46] R. Gourdeau. Object-oriented programming for robotic manipulator simulation. *IEEE Robotics and Automation Magazine*, 9:21–29, 1997.
- [47] N. Halbwachs, P. Caspi, P. Raymond, and D. Pilaud. The synchronous dataflow programming language LUSTRE. *Proceedings of the IEEE*, 79(9):1305–1320, Sept. 1991.
- [48] M. Hardt and O. von Stryk. Dynamic modeling in the simulation, optimization, and control of bipedal and quadrupedal robots. *Zeitung für angewandte Mathematik und Mechanik*, 83(10):648–662, 2003.
- [49] M. W. Hardt. *Multibody Dynamics Algorithms, Numerical Optimal Control, with detailed studies in the control of jet engine compressors and biped walking*. PhD thesis, University of California, San Diego, 1999.
- [50] M. W. Hardt and O. von Stryk. Towards optimal hybrid control solutions for gait optimization patterns of a quadruped. In M. Armada and P. G. de Santos, editors, *Proceedings 3rd International Conference on Climbing and Walking Robots, CLAWAR 2000*, pages 385–392, Madrid, 2000. Bury St. Edmunds and London, UK: Professional Engineering Publishing.
- [51] D. Hatley and I. Pirbhai. *Strategies for Real-Time System Specification*. Dorset House Publishing Company, New York, 1987.
- [52] G. Hirzinger, A. Albu-Schäffer, M. Hähnle, I. Schaefer, and N. Sporer. On a new generation of torque controlled light-weight robots. In *Proc. IEEE Conf. on Robotics and Automation*, pages 3356–3363, Seoul, Korea, May 2001.
- [53] J. M. Hollerbach. A recursive Lagrangian formulation of manipulator dynamics and a comparative study of dynamics formulation. *IEEE Trans. Syst., Man, Cybern.*, SMC–10(11):730–736, 1980.
- [54] J. M. Hollerbach and C. W. Wampler. A taxonomy of kinematic calibration methods. *The Int. J. Robotics Res.*, 14:573–591, 1996.
- [55] *DOMÉ Guide*. Honeywell, Honeywell Technology Center, Honeywell, 1999. <http://www.htc.honeywell.com/dome/>.
- [56] R. Höppler and P. J. Mosterman. Model Integrated Computing in robot control to synthesize real-time embedded code. In *Proceedings of the IEEE Conference on Control Applications*, pages 767–772, Mexico City, 2001.
- [57] R. Höppler and M. Otter. A versatile C++ toolbox for model based real-time control systems of robotic manipulators. In *Proc. IEEE Conf. on Intelligent Robots and Systems*, pages 2208–2214, Maui, Hawaii, USA, 2001.
- [58] R. Höppler, M. Stelzer, and O. von Stryk. Object-oriented dynamics modeling architecture for legged robot trajectory optimization and control. In *Proc. of the IEEE Conference on Mechatronics and Robotics*, 2004. Akzeptiert zur Veröffentlichung.

- [59] R. Höppler and M. Thümmel. Symbolic computation of the inverse dynamics of elastic joint robots. In *Proc. IEEE Conf. on Robotics and Automation*, pages 4314 – 4319, New Orleans, Louisiana, USA, 2004.
- [60] A. Jain. Unified formulation of dynamics for serial rigid multibody systems. *J. Guidance, Contr., Dynamics*, 14(3):531–542, 1991.
- [61] A. Jain and G. Man. Real-time simulation of the Cassini spacecraft using DARTS: Functional capabilities and spatial algebra algorithm. In *Proceedings of the 5th annula conference on Aerospace computational control*, Aug. 1992.
- [62] A. Jain and G. Rodriguez. Recursive flexible multibody systems dynamic using spatial operators. *J. Guidance, Contr., Dynamics*, 15(11):1453–1466, 1992.
- [63] A. Jain and G. Rodriguez. An analysis of the kinematics and dynamics of underactuated manipulators. *IEEE Trans. Robotics and Automat.*, 9:411–422, Aug. 1993.
- [64] A. Jain and G. Rodriguez. Linearization of manipulator dynamics using spatial operators. *IEEE Trans. Syst., Man, Cybern.*, 23(1):239–248, 1993.
- [65] A. Jain and G. Rodriguez. Base-invariant symmetric dynamics of free-flying space manipulators. *IEEE Trans. Robotics and Automat.*, 11:585–597, Aug. 1995.
- [66] A. Jain and G. Rodriguez. Diagonalized Lagrangian robot dynamics. *IEEE Trans. Robotics and Automat.*, 11:571–584, Aug. 1995.
- [67] A. Jain and G. Rodriguez. Sensitivity analysis of multibody dynamics using spatial operators. In *Proceedings of the 6th international conference on methods and models in automation and robotics*, pages 573–578, Miedzyzdroje, Poland, 2000.
- [68] A. Jain, N. Vaidehi, and G. Rodriguez. A fast recursive algorithm for molecular dynamics simulation. *Journal of computational physics*, 106:258–268, 1993.
- [69] R. E. Johnson. Frameworks = (components + patterns). *Communications of the ACM*, 40(10):39–42, Oct. 1997.
- [70] R. E. Johnson and B. Foote. Designing reusable classes. *Journal of object-oriented programming*, 1(2):22–35, June 1988.
- [71] G. Kahn. The semantics of a simple language for parallel programming. In *Proc. of the IFIP Congress 74*, Amsterdam, The Netherlands, 1974.
- [72] T. R. Kane and D. A. Levinson. The use of Kane’s dynamical equations in robotics. *The Int. J. Robotics Res.*, 2(3):3–21, 1983.
- [73] C. Kapoor and D. Tesar. *A reusable operational software architecture for advanced robotics*. PhD thesis, University of Texas, Austin, USA, 1996.
- [74] A. Kecskeméthy. *Objekt-orientierte Modellierung der Dynamik von Mehrkörpersystemen mit Hilfe von Übertragungselementen*. PhD thesis, Universität Duisburg, Germany, 1993.
- [75] W. Khalil and F. Boyer. An efficient calculation of computed torque control of flexible manipulators. In *Proc. IEEE Conf. on Robotics and Automation*, pages 609–614, 1995.
- [76] W. Khalil and J. F. Kleinfinger. A new geometric notation for open and closed-loop robots. In *Proc. IEEE Conf. on Robotics and Automation*, pages 1174 – 1179, Apr. 1986.
- [77] W. Khalil and J.-F. Kleinfinger. Minimum operations and minimum parameters of the dynamic models of tree structure robots. *IEEE J. of Robotics Automat.*, 3(6):517 – 526, 1987.
- [78] O. Khatib. Dynamic control of manipulators in operational space. In *Proceedings 6th CISM-IFTToMM congress on theory of machines and mechanisms*, New Delhi, India, Dec. 1983.
- [79] O. Khatib. A unified approach for motion and force control of robot manipulators: The operational space formulation. *IEEE J. of Robotics Automat.*, RA-3(1):43–53, 1987.
- [80] H. G. Kwatny and G. L. Blankenship. Symbolic construction of models for multi-body dynamics. *IEEE Trans. Robotics and Automat.*, RA-11(2):271–281, 1995.

- [81] J. Lakos. *Large-Scale C++ Software Design*. Addison Wesley Publishing Company, 1996.
- [82] E. A. Lee. What's ahead for embedded software? *IEEE Computer*, 33(9):18–26, Sept. 2000.
- [83] E. A. Lee. Overview of the Ptolemy project. Technical memorandum, UCB/ERL, Mar. 2001.
- [84] E. A. Lee and T. M. Parks. Dataflow process networks. In *Proceedings of the IEEE*, volume 83, pages 773–801, May 1995.
- [85] G. Leister. *Beschreibung und Simulation von Mehrkörpersystemen mit geschlossenen kinematischen Schleifen*. PhD thesis, Universität Stuttgart, 1992.
- [86] C.-J. Li. A new Lagrangian formulation of dynamics for robot manipulators. *ASME J. of Dynamic Systems Meas. and Control*, 111:559–567, 1989.
- [87] J. Y. S. Luh, M. W. Walker, and R. P. C. Paul. On-line computational scheme for mechanical manipulators. *ASME J. of Dynamic Systems Meas. and Control*, 102:69–76, 1980.
- [88] A. Mallet and H. Bruyninckx. A specification of generic robotics software components: future evolutions of Genom in the Orocos context. In *Proc. IEEE Conf. on Intelligent Robots and Systems*, Lausanne, Switzerland, Sept. 2002.
- [89] *SimMechanics User's Guide*. The Mathworks, Inc., Natick, MA, USA, 2002.
- [90] S. McMillan. *Computational Dynamics for Robotic Systems on Land and Under Water*. PhD thesis, The Ohio State University, Columbus, OH, 1994.
- [91] S. McMillan and D. E. Orin. Efficient computation of articulated-body inertias using successive axial screws. *IEEE Trans. Robotics and Automat.*, 11(4):606–611, 1995.
- [92] S. McMillan, D. E. Orin, and R. B. McGhee. DynaMechs: An object oriented software package for efficient dynamic simulation of URVs. In *Underwater Robotic Vehicles: Design and Control*, pages 73–98. TSI Press, Albuquerque, NM, U.S.A., 1995.
- [93] D. Miller and R. Lennox. An object-oriented environment for robot system architectures. In *Proc. IEEE Conf. on Robotics and Automation*, pages 352 – 361, May 1990.
- [94] *Modelica Standard Library Specification*. Modelica Association. <http://www.modelica.org>.
- [95] *Modelica - A unified object-oriented language for physical systems modeling. Language Specification*. Modelica Association, version 2.0 edition, 2002. <http://www.modelica.org>.
- [96] M. Möller. *Ein Verfahren zur automatischen Analyse der Kinematik mehrschleifiger räumlicher Mechanismen*. PhD thesis, Universität Stuttgart, 1992.
- [97] S. H. Murphy, J. T. Wen, and G. N. Saridis. Simulation and analysis of flexibly jointed manipulators. In *Proc. IEEE Conf. on Decision and Control*, pages 545–550, Honolulu, Hawaii, USA, Dec. 1990.
- [98] R. M. Murray, Z. Li, and S. S. Sastry. *A mathematical introduction to robotic manipulation*. CRC Press, Boca Raton, 1993.
- [99] C. P. Neuman and J. J. Murray. Linearization and sensitivity functions of dynamic robot models. *IEEE Trans. Syst., Man, Cybern.*, 14(6):805–818, 1984.
- [100] *UML notation guide. Version 1.1*. The Object Management Group, Sept. 1997. doc. no. ad/97-08-05.
- [101] *UML semantics. Version 1.1*. The Object Management Group, Sept. 1997. doc. no. ad/97-08-04.
- [102] M. J. O'Riordan. Technical report on C++ performance. Technical report, Sept. 2002.
- [103] N. Orlandea. ADAMS (theory and applications). In A. DePater and H. Pacejka, editors, *Proceedings of 3rd seminar on advanced vehicle systems dynamics*, pages 121–166, Amalfi, May 1986.
- [104] M. Otter. *Objektorientierte Modellierung mechatronischer Systeme am Beispiel geregelter Roboter*. Fortschrittberichte VDI, Reihe 20, Nr.147, 1995.
- [105] F. B. Oueddou, O. Bruneau, and J. C. Guinot. Dynamic analysis tool for legged robots. *Multibody Systems Dynamics*, 2:369–391, 1998.

- [106] D. K. Pai, U. M. Ascher, and P. G. Kry. Forward dynamics algorithms for multibody chains and contact. In *Proc. IEEE Conf. on Robotics and Automation*, pages 857–863, 2000.
- [107] C. Pantelides. The consistent initialization of differential-algebraic systems. *SIAM Journal of scientific and statistical computing*, 9:213–231, 1988.
- [108] F. C. Park and J. E. Bobrow. A recursive algorithm for robot dynamics using Lie groups. In *Proc. IEEE Conf. on Intelligent Robots and Systems*, 1994.
- [109] F. C. Park, J. E. Bobrow, and S. Ploen. A Lie group formulation of robot dynamics. *The Int. J. Robotics Res.*, 14(6):609–618, Dec. 1995.
- [110] H. M. Paynter. Discussion on paper "The properties of bond graph junction structures". *ASME J. of Dynamic Systems Meas. and Control*, 98:209–210, 1976.
- [111] F. Pfeiffer and C. Glocker. *Multibody dynamics with unilateral contacts*. Series in nonlinear science. John Wiley & sons, 1996.
- [112] F. Pfeiffer and R. Johanni. A concept for manipulator trajectory planning. *IEEE Trans. Robotics and Automat.*, RA-3(2):115–122, 1987.
- [113] S. R. Ploen. *Geometric algorithms for the dynamics and control of multibody systems*. PhD thesis, University of California, Irvine, 1997.
- [114] F. Proctor. Intelligent open architecture control, July 2003. URL <http://www.mel.nist.gov/proj/ioacms.htm>.
- [115] R. E. Roberson and R. Schwertassek. *Dynamics of multibody systems*. Springer Verlag, 1988.
- [116] R. E. Roberson and J. Wittenburg. A dynamical formalism for an arbitrary number of interconnected rigid bodies. with reference to the problem of satellite attitude control. In *3rd IFAC congress, 1966*, pages 46D.2–46D.9, London, 1968.
- [117] G. Rodriguez, A. Jain, and K. Kreutz-Delgado. A spatial operator algebra for manipulator modeling and control. *The Int. J. Robotics Res.*, 10(4):371–381, 1991.
- [118] G. Rodriguez, A. Jain, and K. Kreutz-Delgado. Spatial operator algebra for multibody system dynamic. *Journal of the Astronautical Sciences*, 40:27–50, 1992.
- [119] G. Rodriguez and K. Kreutz-Delgado. Spatial operator factorization and inversion of the manipulator mass matrix. *IEEE Trans. Robotics and Automat.*, 8(1):65–76, 1992.
- [120] W. Rulka. SIMPACK – a computer program for simulation of large-motion multibody systems. In W. Schielen, editor, *Multibody Systems Handbook*, pages 265–284. Springer Verlag, Berlin, Heidelberg, New York, Tokyo, 1990.
- [121] R. R. Ryan. ADAMS – multibody systems analysis software. In W. Schielen, editor, *Multibody Systems Handbook*, pages 361–402. Springer Verlag, Berlin, Heidelberg, New York, Tokyo, 1990.
- [122] D. B. Schaechter and D. A. Levinson. Interactive computerized symbolic dynamics for the dynamacist. *Journal of the Astronautical Sciences*, 36(4):365–388, 1988.
- [123] W. Schiehlen, editor. *Multibody Systems Handbook*. Springer Verlag, Berlin, Heidelberg, New York, Tokyo, 1990.
- [124] R. Schwertassek and K.-H. Senger. Representation of joints in multibody systems. *Zeitung für angewandte Mathematik und Mechanik*, 68(2):111–119, 1988.
- [125] R. Schwertassek and O. Wallrapp. *Dynamik flexibler Mehrkörpersysteme*. Vieweg & Sohn Verlagsgesellschaft, Braunschweig, Wiesbaden, 1999.
- [126] B. Selic, G. Gullekson, and P. T. Ward. *Real-time object-oriented modeling*. John Wiley & sons, 1994.
- [127] B. Selic and J. Rumbaugh. Using UML for Modeling Complex Real-Time Systems. White paper, Rational Software, 1998.
- [128] J. G. Siek, L.-Q. Lee, and A. Lumsdaine. *The Boost Graph Library*. Addison Wesley Publishing Company, 2002.

- [129] W. M. Silver. On the equivalence of Lagrangian and Newton-Euler dynamics for manipulators. *The Int. J. Robotics Res.*, 1(2):118–128, 1982.
- [130] B. Simeon. MBSPACK – numerical integration software for constrained mechanical motion. *Surveys on Mathematics for Industry*, 5(3):169–202, 1995.
- [131] J.-J. E. Slotine and W. Li. *Applied non-linear control*. Prentice-Hall, 1991.
- [132] G. A. Sohl and J. E. Bobrow. A recursive multibody dynamics and sensitivity algorithm for branched kinematic chains. *ASME J. of Dynamic Systems Meas. and Control*, 123:391–399, Sept. 2001.
- [133] M. W. Spong. Modeling and control of elastic joint robots. *Transactions of the ASME*, 109:310–319, 1987.
- [134] M. W. Spong and M. Vidyasagar. *Robot dynamics and control*. John Wiley & sons, 1989.
- [135] M. Stelzer, M. Hardt, and O. von Stryk. Efficient dynamic modeling, numerical optimal control and experimental results for various gaits of a quadruped robot. In *CLAWAR 2003: International Conference on Climbing and Walking Robots, Catania, Italy, Sept. 17-19*, pages 601–608, Aachen, 2003. Shaker.
- [136] W. Stelzle, A. Kecskeméthy, and M. Hiller. A comparative study of recursive methods. *Archive of applied mechanics*, 66(1–2):9–19, 1995.
- [137] L. M. Sweet and M. C. Good. Redefinition of the robot motion control problem. *IEEE Control Systems Magazine*, 5(3):18–24, 1985.
- [138] J. Sztipanovits and G. Karsai. Model-integrated computing. *IEEE Computer*, 4:110–112, 1997.
- [139] J. Sztipanovits, G. Karsai, and T. Bapty. Self-adaptive software for signal processing. *Communications of the ACM*, 41(5):66–73, 1998.
- [140] R. E. Tarjan. Depth first search and linear graph algorithms. *SIAM Journal on Computing*, 1(2):146–160, 1972.
- [141] M. Thümmel, M. Otter, and J. Bals. Control of robots with elastic joints based on automatic generation of inverse dynamics models. In *Proc. IEEE Conf. on Intelligent Robots and Systems*, pages 925–930, Maui, Hawaii, USA, 2001.
- [142] D. van Heesch. Doxygen. URL <http://www.doxygen.org>.
- [143] H. Vangheluwe and J. de Lara. Meta-models are models too. In *Proceedings of the 2002 Winter Simulation Conference*, pages 597 – 605, Dec. 2002.
- [144] A. F. Vereshchagin. Computer simulation of the dynamics of complicated mechanisms of robot manipulators. *Engineering Cybernetics*, 6:65–70, 1974.
- [145] O. von Stryk. *User's Guide for DIRCOL – a Direct Collocation Method for the Numerical Solution of Optimal Control Problems*. Technische Universität Darmstadt, 2.1 edition, Nov. 1999. Updated April 2002.
- [146] O. von Stryk and M. Schlemmer. Optimal control of the industrial robot manutec r3, 1994.
- [147] M. W. Walker and D. E. Orin. Efficient dynamic computer simulation of robotic mechanisms. *ASME J. of Dynamic Systems Meas. and Control*, 104:205–211, 1982.
- [148] O. Wallrapp. Standardization of flexible body modeling in multibody system codes, Part I: Definition of standard input data. *Journal of Mechanics of Structures and Machines*, 22(3):283–403, 1994.
- [149] J. Wittenburg. *Dynamics of systems of rigid bodies*. B. G. Teubner, Stuttgart, 1977.
- [150] C. Woernle. *Ein systematisches Verfahren zur Aufstellung der geometrischen Schliessbedingungen in kinematischen Schleifen mit Anwendung bei der Rückwärtstransformation für Industrieroboter*. PhD thesis, Universität Stuttgart, 1988.