KU LEUVEN

ARENBERG DOCTORAL SCHOOL
Faculty of Engineering Science

Composable Robot Motion Stack

Implementing constrained hybrid dynamics using semantic models of kinematic chains

Azamat Shakhimardanov

Dissertation presented in partial fulfillment of the requirements for the degree of Doctor of Engineering Science

November 2015

Composable Robot Motion Stack

Implementing constrained hybrid dynamics using semantic models of kinematic chains

Azamat SHAKHIMARDANOV

Examination committee:
Prof. dr.-ir. Jean Berlamont, chair
Prof. dr. ir. Herman Bruyninckx, supervisor
Prof. Dr.-Ing. Gerhard Kraetzschmar, co-supervisor
Prof. Dr.-Ing. Erwin Prassler, co-supervisor
Prof. dr. ir. Joris De Schutter
Prof. dr. ir. Tinne De Laet
Dr. ir. Erwin Aertbeliën
Prof. Dr. Klas Nilsson (Lund University, Sweden)

Dissertation presented in partial fulfillment of the requirements for the degree of Doctor of Engineering Science

November 2015

Uitgegeven in eigen beheer, Azamat Shakhimardanov, Celestijnenlaan 300 box 2420, B-3001 Heverlee (Belgium)

Preface

To be honest, documenting and writing are some of those few things that I cannot enjoy. Analysis, problem solving and discussions of the technical details, those are the things I always liked, but when it came to writing I would always spend hours and hours struggling to find the right introductory sentence and this doctoral thesis was not an exception. That is why it was very much to my surprise that I managed to write such a long text and even did it in a foreign language. Such an undertaking would not have been possible were it not for many many people who shaped me as a researcher and as a person I am now, and contributed to a successful completion of my efforts. I made both positive and negative experiences during these relatively short academic period of my life, but those made me grow not only as a scientist but more importantly as a human. Thank you very much everyone!

First of all, it is worth to mention that as a person coming from a different cultural background where there is a clear line between a student and a professor, I was lucky to have great personalities on my supervisory board. Most of them were not only open to scientific discussions, but also had often time to share a cup of coffee, wine and many mugs of beer with Schnitzel or spareribs in a German 'Brauhaus'(brewery).

Everything started when I came to study to Germany in 2004. I enrolled at the master's program in Autonomous Systems at the University of Applied Sciences Bonn-Rhein-Sieg. After graduating the program in early 2007, I was lucky to get a job offer from Prof. Erwin Prassler, who later co-supervised my doctoral studies, in EU FP6 project RoSta on Robot Standards and Reference Architectures. In the RoSta project, I got involved in research on robot software frameworks and architectures. In this context I got to know another member of my current doctoral examination committee, Prof. Klas Nilsson from the University of Lund. I visited Lund multiple times to have small technical workshops on control architectures with a clear goal of bringing high level concepts like ontologies closer to a low level robot control. It is already then,

some of the ideas researched in this thesis started evolving. During one of such workshops I met Prof. Herman Bruyninckx, who later would become my supervisor.

After the RoSta project, I continued my research on robot control software in the context of EU FP7 project BRICS. It was a great chance, because I got an opportunity to work in a team led by Prof. Gerhard Kraetzschmar and Herman was representing University of Leuven in this project. This constellation made future collaborations possible. Gerhard was not only a leader to distribute commands, but he was also very supportive on a personal level. If you had any kind of trouble, he would stop doing his own work and focus on solving your problems. He was also very supportive in coping with German and Uzbek bureaucratic machinery by always writing some official letters, whenever those were needed to extend my visa or passport.

During the BRICS project, I visited Leuven very often and had short meetings and discussions with Herman. The discussions often focused on how robot programming can be improved and how multiple sub-domains involved can be systematically integrated into coherently working software and robot control architecture. After one of such meetings, we ended up in a traditional Belgian restaurant over a plate of Vlaams Stoofvlees and talked about whether I should pursue a PhD. I was warned that my naive enthusiasm might turn into a frustration pretty quickly and that is how I officially started my doctoral studies in December 2010.

Indeed, it was a bumpy road since then. I had to change my research topic several times and tried searching what might be useful and interesting at the same time. Eventually, I got into the domain of the robot dynamics, constrained motion control and software support for them. Being able to speak Russian also helped in this choice, because Herman had re-discovered some constrained dynamics algorithm that was developed by Russian roboticist in 1970's and was ahead of its time, but never realized and released to a bigger community. So, I started working on this topic and tried to extend and improve it with semantic models, in order to simplify its integration in larger robotic applications and to enable a development of model driven engineering tools that rely on such models. It was awesome to work with Herman. Sometimes he would discuss some totally 'meta stuff' that probably only he himself understood, and sometimes he would dig into the source code with you. I was positively forced to swot, in order to keep up with his vision.

I would also like to thank Prof. Joris De Schutter and Prof. Tinne De Laet for their critical but always constructively helpful inputs during the few discussions we had. I also thank you and Dr. Erwin Aertbeliën for providing useful feedback that helped me to improve this doctoral thesis.

I must also say, this PhD endeavor was truly international experience. I come from a foreign country, lived in Germany for almost ten years, commuted between Germany and Belgium for five years. During this time I studied and worked at two sites, University of Applied Sciences Bonn-Rhein-Sieg and University of Leuven, and had a chance to do research and make friends with many great people, whom I would like to acknowledge as well.

During all these years, I shared an office with Nico Hochgeschwender and it was awesome. Once, I remember asking him to help me to carry a huge mattress across the town on his second day at the new job. He was surprised but agreed to help, which is unheard of for a German Schwabe who did not even know me. Since then we made good friend. We always had very good discussions related to our research and soccer. Nico, you are the best FF!

The other colleagues at Bonn-Rhein-Sieg who became good friends and I would like to thank are Mike Reckhaus for being a good friend and sharing time with our families, Fredderik Hegger for doing all the stuff that nobody else wanted to do and being a good friend, Sven Schneider for providing valuable feedback on my research and software, Jan Paulus for showing how to be a pragmatic German. The colleagues at the University of Leuven that I would like to acknowledge are Lin Zhang for his discussions on FPGA and embedded systems, Enea Scioni for discussions on JSON, Enrico Di Lello for coffee and giving tips on doctoral studies, Markus Klotzbücher for Lua and compositionality discussions, Dominick Vanthienen for sharing his thoughts about task solvers, Ruben Smits for an introduction to KDL software, Peter Soetens for his discussions on component-software for robots.

I am also grateful to Allison Williams and Nico's wife Nicole Goebel who proof read parts of this thesis. Also, many thanks to a close friend and my martial arts master Dr. Andrea Raccanelli and his family for his understanding, support and never-ending storage of Italian wine.

Finally, I am wholeheartedly grateful to my wife, Madina Mukhamedova who supported and bore with me and my stubborn personality till the end. I also sincerely thank my parents and parents-in-law who had faith in us and were there for us, whenever we and our children needed them the most. Last but not least, thank you my children Nigina and Rustam for cheering me up all this time.

Azamat Shakhimardanov,
Leuven, November 2015.

Acknowledgments

This work was supported by KU Leuven's *Concerted Research Action* GOA/2010/011, and European FP7 projects *RoboHow* (FP7-ICT-288533), *BRICS* (FP7-ICT-231940), *Rosetta* (FP7-ICT-230902), and *Pick-n-Pack* (FP7-311987). The author is grateful for the many constructively critical interactions within these projects, since they added another four or five decades of experiences with large software design and development projects. The author is also grateful to the administration of Department of Computer Science at University Bonn-Rhein-Sieg for allowing him to use their facilities for the past two years, even though he was not officially affiliated.

Abstract

Over the last 50 years, the controlled motion of robots has become a very mature domain of expertise. It can deal with all sorts of topologies and types of joints and actuators, with kinematic as well as dynamic models of devices, and with one or several tools or sensors attached to the mechanical structure. Nevertheless, the domain has not succeeded in standardizing the modelling of robot devices (including such fundamental entities as "reference frames"!), let alone the semantics of their motion specification and control.

This thesis aims to solve this long-standing problem, from three different sides: semantic models for robot kinematics and dynamics, semantic models of all possible motion specification and control problems, and software that can support the latter while being configured by a systematic use of the former.

The diversity of robot motion tasks has led to the development of (constrained) task control methodologies with origins in force control, humanoid robot control, mobile manipulator control, visual servoing, etc. This has influenced the way these methodologies formulate and compute constrained task control problems, and how software was developed and documented. Hence, the task programming approaches advocated by each framework often build upon the (most often only implicitly specified) semantics of the models and the assumptions of that specific domain. As a result, the semantics and the behaviors of the relationships of the task constituents, "target objects, constraints on object relations, controlled actions, solvers, and cost functions", that seemingly represent the same primitives across the frameworks have become ambiguous. This impedes a deterministic composition of the models and the software implementations of these task constituents and a compositionality of the resultant tasks.

In order to address this problem, this doctoral thesis introduces a systematic constrained motion task control specification and programming approach. The approach first explicitly and formally defines the relationships between the

abstractions of the motion task programming stack and the components of the constrained motion task control architecture. It introduces the concept of semantic models and exemplifies their role in the modelling and the realization of task control applications. This research is presented in the form of two contributions.

- Firstly, this thesis develops composable semantic models to describe kinematic chain structures and associated computations. These models allow decoupling of physical primitives from their coordinate-specific representations. Additionally, they enable specifications of complex computations on the kinematic chains by composing simpler modelling primitives. The compositions take into account the constraints that each primitive needs to satisfy when used with other primitives. This feature is achieved by explicitly separating structural and behavioral aspects of the models and follows the Model Driven Engineering (MDE) methodology. MDE proposes decoupling of the domain specific models, their instances, and their implementations from the generic domain independent meta-models. MDE methodology allowed to programmatically formulate semantic models in the form of a Domain Specific Language (DSL) and its related tool-chain infrastructure.
- Secondly, this thesis expresses each task control problem as a constrained optimization problem; such constrained optimization and optimal control formulations are already at the basis of many of the existing task control and software frameworks. This research focuses on complete and systematic (re-)configurability of such Whole Body Control Architecture (WBCA) and in particular, the solver component that plays a central role in the implementations of the constrained motion control tasks. The solver used in the simulations and the experiments is the Popov-Vereshchagin constrained hybrid dynamics solver, which is extended to account for different constraint-controller configurations. The research also exemplifies the relations between the components of WBCA and their software representations in the form of the DSLs in the motion programming stack. Furthermore, it shows that every component of the WBCA and respectively its software counterpart should comply with their associated semantic models. This is imperative in order to allow a systematic and a physically-valid constrained task specification and its computation.

Finally, every contribution is accompanied by a number of examples that showcase how various semantic models, architectures, and their software implementation should be used in the context of larger applications. All

semantic models and software contributions are provided with open source licenses.

Beknopte samenvatting

Gedurende de laatste 50 jaar is bewegingscontrole van robots een volwassen onderzoeksdomein geworden, dat weet om te gaan met alle soorten van kinematische topologieën, gewrichten en aandrijvingen, met kinematische en dynamische eigenschappen, en met meer dan één werktuig per robot. Desalniettemin bestaat er nog altijd geen standaard-voorstelling voor robots (zelfs niet voor het allereenvoudigste onderdeel, een referentie-assenstelsel!), en al zeker niet voor de semantische modellen van robots en hun bewegingscontrole.

Deze thesis probeert een oplossing aan te leveren, vanuit drie verschillende invalshoeken: semantische modellen voor de kinematica en dynamica van robots, semantische modellen voor alle mogelijke bewegingsspecificaties en controle-problemen, en software ter ondersteuning van deze laatste soort modellen, op basis van een systematisch gebruik van de eerte soort modellen.

De variëteit in bewegingstaken voor robots heeft geleid tot de ontwikkeling van methodologieën vanuit verschillende toepassingsgebieden: krachtcontrole, humanoide robots, mobiele manipulatoren, visuele volgtaken, enz. De toepassingsgebieden hebben een een heel sterke en zichtbare invloed gehad op hoe de bewegingstaken zijn beschreven en opgelost, en op de soort van software-ondersteuning die is ontwikkeld. Met name zijn de specifieke *taken* uit de verschillende toepassingsgebieden nog heel erg zichtbaar in de software en documentatie, en zijn een aantal zaken met elkaar gekoppeld geraakt die enkel zin hebben voor een welbepaald toepassingsgebied. Om dezelfde reden is er dubbelzinnigheid ingeslopen in de gebruikte terminologie, omdat verschillende termen worden gebruikt om dezelfde concepten voor te stellen rond taakobjecten, beperkingen op de relaties tussen die objecten, de beschrijving van de controle-acties, de oplossingsalgoritmes, en de objectieffuncties die men in bepaalde taken wil optimiseren. Dit alles heeft geresulteerd in slechts heel beperkt hergebruik van inspanningen en software, bij het samenstellen van complexe robotsystemen en -taken.

Om dit probleem te verhelpen introduceert deze thesis een systematische en sterk ontkoppelde aanpak voor het hele probleem, met expliciete formele voorstellingen van alle concepten in bewegingsmodellering en -specificatie, controle en toepassingen. Hiervoor worden "semantische modellen" geformuleerd, wiens nut en impact worden aangetoond via allerhande concrete voorbeelden. Meer bepaald brengt dit werk bijdragen aan in twee domeinen:

- *Ten eerste* worden "samenstelbare" (Eng.: "composable") semantische modellen voorgesteld, voor de kinematische structuren van robots, en alle bijhorende berekeningen. De modellen zorgen voor een systematische scheiding tussen de fysische primitieven enerzijds, en hun voorstelling in concrete gegevensstructuren en functies anderzijds. De modellen zijn zo ontworpen dat ze met slechts een minimum aan primitieven toch alle mogelijke complexe robotsystemen kunnen voorstellen, via systematische compositie-functies. Deze composities laten toe om fysische en kunstmatige beperkingen expliciet te modelleren, en om structuur en gedrag netjes van elkaar te scheiden. De inspiratie van de voorgestelde aanpak komt uit het domein van *Model Driven Engineering* (MDE); hieruit ontlenen we de concepten van meta model, model en instantie, waarbij het eerste volledig onafhankelijk is van het concrete toepassingsdomein, terwijl de laatste twee daar de eigenschappen van het toepassingsdomein aan toevoegen. De MDE methodologie maakt het mogelijk om een formele taal te creëren, een zogenaamde *Domain Specific Language* (DSL), om programmatorisch om te gaan met de semantische concepten.
- *Ten tweede* ontwikkelt deze thesis uitdrukkingen om alle robot-taken te formuleren als optimisatieproblemen met beperkingen (Eng: *constrained optimization problems*). Deze aanpak is niet nieuw op zich, maar de bijdrage ligt bij het systematisch gebruik van de semantische modellen (de hogergenoemde "DSLs") bij het samenstellen en configureren van zogenaamde "Whole Body Control Architectures", d.w.z., software systemen die een taakspecificatie omzetten naar concrete controle-software voor de specifieke kinematische ketting van specifieke robots.

Tenslotte worden al deze bijdragen geillustreerd aan de hand van een aantal voorbeelden over hoe deze semantische modellen, architecturen en hun software implementaties kunnen gebruikt worden om grote toepassingen te maken. Al de gerealiseerde bijdragen zijn beschikbaar onder *open source* licenties.

Gedurende de laatste 50 jaar is bewegingscontrole van robots een volwassen onderzoeksdomein geworden, dat weet om te gaan met alle soorten van kinematische topologieën, gewrichten en aandrijvingen, met kinematische en dynamische eigenschappen, en met meer dan één werktuig per robot.

Desalniettemin bestaat er nog altijd geen standaard-voorstelling voor robots (zelfs niet voor het allereenvoudigste onderdeel, een referentie-assenstelsel!), en al zeker niet voor de semantische modellen van robots en hun bewegingscontrole.

Deze thesis probeert een oplossing aan te leveren, vanuit drie verschillende invalshoeken: semantische modellen voor de kinematica en dynamica van robots, semantische modellen voor alle mogelijke bewegingsspecificaties en controle-problemen, en software ter ondersteuning van deze laatste soort modellen, op basis van een systematisch gebruik van de eerte soort modellen.

De variëteit in bewegingstaken voor robots heeft geleid tot de ontwikkeling van methodologieën vanuit verschillende toepassingsgebieden: krachtcontrole, humanoide robots, mobiele manipulatoren, visuele volgtaken, enz. De toepassingsgebieden hebben een een heel sterke en zichtbare invloed gehad op hoe de bewegingstaken zijn beschreven en opgelost, en op de soort van software-ondersteuning die is ontwikkeld. Met name zijn de specifieke *taken* uit de verschillende toepassingsgebieden nog heel erg zichtbaar in de software en documentatie, en zijn een aantal zaken met elkaar gekoppeld geraakt die enkel zin hebben voor een welbepaald toepassingsgebied. Om dezelfde reden is er dubbelzinnigheid ingeslopen in de gebruikte terminologie, omdat verschillende termen worden gebruikt om dezelfde concepten voor te stellen rond taakobjecten, beperkingen op de relaties tussen die objecten, de beschrijving van de controle-acties, de oplossingsalgoritmes, en de objectieffuncties die men in bepaalde taken wil optimiseren. Dit alles heeft geresulteerd in slechts heel beperkt herbebruik van inspanningen en software, bij het samenstellen van complexe robotsystemen en -taken.

Om dit probleem te verhelpen introduceert deze thesis een systematische en sterk ontkoppelde aanpak voor het hele probleem, met expliciete formele voorstellingen van alle concepten in bewegingsmodellering en -specificatie, controle en toepassingen. Hiervoor worden "semantische modellen" geformuleerd, wiens nut en impact worden aangetoond via allerhande concrete voorbeelden. Meer bepaald brengt dit werk bijdragen aan in twee domeinen:

- *Ten eerste* worden "samenstelbare" (Eng.„'composable") semantische modellen voorgesteld, voor de kinematische structuren van robots, en alle bijhorende berekeningen. De modellen zorgen voor een systematische scheiding tussen de fysische primitieven enerzijds, en hun voorstelling in concrete gegevensstructuren en functies anderzijds. De modellen zijn zo ontworpen dat ze met slechts een minimum aan primitieven toch alle mogelijke complexe robotsystemen kunnen voorstellen, via systematische compositie-functies. Deze composities laten toe om fysische en kunstmatige beperkingen expliciet te modelleren, en om structuur en gedrag netjes van elkaar te scheiden. De inspiratie van de voorgestelde

aanpak komt uit het domein van *Model Driven Engineering* (MDE); hieruit ontlenen we de concepten van meta model, model en instantie, waarbij het eerste volledig onafhankelijk is van het concrete toepassingsdomein, terwijl de laatste twee daar de eigenschappen van het toepassingsdomein aan toevoegen. De MDE methodologie maakt het mogelijk om een formele taal te creëren, een zogenaamde *Domain Specific Model* (DSL), om programmatorisch om te gaan met de semantische concepten.

- *Ten tweede* ontwikkelt dezee thesis uitdrukkingen om alle robot-taken te formuleren als optimisatieproblemen met beperkingen (Eng: *constrained optimization problems*). Deze aanpak is niet nieuw op zich, maar de bijdrage ligt bij het systematisch gebruik van de semantische modellen (de hogergenoemde "DSLs") bij het samenstellen en configureren van zogenaamde "Whole Body Control Architectures", d.w.z., software systemen die een taakspecificatie omzetten naar concrete controle-software voor de specifieke kinematische ketting van specifieke robots.

Tenslotte worden al deze bijdragen geillustreerd aan de hand van een aantal voorbeelden over hoe deze semantische modellen, architecturen en hun software implementaties kunnen gebruikt worden om grote toepassingen te maken. Al de gerealiseerde bijdragen zijn beschikbaar onder industrie-vriendelijke *open source* licenties.

Abbreviations

Notation	Description	
ACADO	Automatic Control and Dynamic Optimization	19
ADL	Architecture Description Language	22
AIST	National Institute of Advanced Industrial Science and Technology	1
API	Application Programming Interface	14
ARM	Acorn RISC Machine	19
AST	Abstract Syntax Tree	26
AutomationML	Automation Markup Language	38
BFS	Breadth First Search	58
BRICS	Best Practice in Robotics	5
BVP	Boundary Value Problem	27
CAL	CAL Actor Language	24
CasADi	Symbolic Framework for Algorithmic Differentiation and Numeric Optimization	19
COLLADA	COLLAborative Design Activity	38
COP	Component Oriented Programming	22
CORBA	Common Object Request Broker Architecture	22
DARPA	Defense Advanced Research Projects Agency	1
DE	Differential Equation	27
DFS	Depth First Search	58
DH	Denavit-Hartenberg	20
DLR	Deutsches Zentrum für Luft- und Raumfahrt	1
DoF	Degree of Freedom	39
DP	Dynamic Programming	27
DSL	Domain Specific Language	12
DTD	Document Type Definition	38

Notation	Description	
EJB	Enterprise Java Beans	22
FODA	Feature Oriented Domain Analysis	20
FPGA	Field-Programmable Gate Array	19
HQP	Hierarchical Quadratic Programming	31
ICE	Internet Communications Engine	22
IDL	Interface Description Language	22
iTaSC	Instantaneous Task Specification using Constraints	1
IVP	Initial Value Problem	27
JSON	JavaScript Object Notation	54
JSON-LD	JavaScript Object Notation for Linked Data	67
KDL	Kinematics Dynamics Library	7
KUKA	Industrie-Werke Karlsruhe Augsburg Aktiengesellschaft	5
LWR	Leight Weight Robot	14
MDA	Model Driven Architecture	24
MDE	Model Driven Engineering	12
MOC	Model Of Computation	23
MPC	Model Predictive Control	20
MPS	Meta Programming System	26
NLP	Nonlinear Programming Problem	27
NOK	Not OK	63
OCP	Optimal Control Problem	27
ODE	Ordinary Differential Equation	27
OROCOS	Open RObot Control Software	19
PD	Proportional Derivative	133
PID	Proportional Integral Derivative	133
QNX	QNX Real-Time Operating System	19

Notation	Description	
qpOASES	Quadratic Programming with Online Active Set Strategy	19
rFSM	Reduced Finite State Machine	19
ROS	Robot Operating System	19
ROSETTA	RObot control for Skilled ExecuTion of Tasks in natural interaction with humans based on Autonomy	5
RoSta	Robot Standards and Reference Architectures	5
RTT	Real-Time Toolkit	19
SoT	Stack of Tasks	1
SRDF	Semantic Robot Description Format	38
SWBC	Stanford Whole Body Control Framework	1
TFF	Task Frame Formalism	1
UML	Unified Modelling Language	39
URDF	Universal Robot Description Format	38
ViSP	Visual Servoing Platform	9
WBC	whole-body control	19
WBCA	whole-body control architecture	19
XML	Extensible Markup Language	38

List of Symbols

Chapter 1

$\mathbf{C}$	a model of a controller in a control block diagram
$\mathbf{f}$	disturbances on plant $\mathbf{P}$
$\hat{\mathbf{f}}$	an estimate of disturbance $\mathbf{f}$
$\mathbf{g}$	disturbances on controller $\mathbf{C}$
$\mathbf{M} + \mathbf{E}$	a model updater and estimator in a control block diagram
$\bar{\boldsymbol{E}}(\boldsymbol{r})$	Mayer or end-point term of objective function
$\bar{\boldsymbol{f}}(\boldsymbol{x}, \boldsymbol{u})$	a differential equation model of the robot motion
$\bar{\boldsymbol{h}}(\boldsymbol{x}, \boldsymbol{u})$	a geometric path constraint function
$\boldsymbol{L}(\boldsymbol{x}, \boldsymbol{u})$	Lagrange term of objective function
$\boldsymbol{T}$	an optimization or a prediction period
$\boldsymbol{t}$	a time variable
$\boldsymbol{t_0}$	a time instant when the prediction phase in Model Predictive Control starts
$\boldsymbol{\delta}$	a time frame during which computed control inputs of Optimal Control Problem are applied
$\boldsymbol{x(0)},\ \boldsymbol{x_0}$	an initial state constraint
$\boldsymbol{r(x(T))}$	a terminal state constraint
$\mathbf{P}$	a model of a robot or a plant in a control block diagram

$\boldsymbol{C(q, \dot{q})}$	a joint space pose and velocity dependent bias force: gravity, friction and Coriolis
$\boldsymbol{h(q)}$	a joint space function of holonomic pose constraint
$\boldsymbol{\dot{X}}$	a Cartesian space velocity variable
$\boldsymbol{M(q)}$	a joint space inertia mass matrix
$\boldsymbol{q}$	a joint space pose variable; pose in generalized coordinates
$\boldsymbol{\dot{q}}$	a joint space velocity variable; velocity in generalized coordinates
$\boldsymbol{\ddot{q}}$	a joint space acceleration variable; acceleration in generalized coordinates
$\boldsymbol{J_r(q)}$	a robot Jacobian matrix
$\boldsymbol{\tau(q, \dot{q})}$	a joint space net force that includes active joint torque, constraint joint torque, a force due to the gravity, friction force and Coriolis force
$\boldsymbol{\tau_a(q)}$	a joint space active input force/effort in generalized coordinates
$\boldsymbol{\tau_c(q)}$	a joint space constraint force/effort in generalized coordinates
$\mathbf{u}$	an output of controller $\mathbf{C}$
$\mathbf{y}$	a measured output trajectory
$\hat{\mathbf{y}}$	an estimate of the output $\mathbf{y}$
$\mathbf{y}_d$	a desired input trajectory
$\mathbf{z}$	observed state of plant $\mathbf{P}$

Chapter 2

$\boldsymbol{a_i}$	an Euclidean linear acceleration three vector
$\boldsymbol{cm_i}$	a pose of the center of mass point of an ideal rigid body
$\boldsymbol{f_i}$	an Euclidean linear force three vector
$\boldsymbol{F_j}$	a spatial force six vector transmitted over the joint that constraints the relative motion of two rigid bodies

$\boldsymbol{I_i}$	a moment of inertia of a rigid body measured about an axis passing through the center of mass point
$\boldsymbol{m_i}$	mass of a rigid body
$\boldsymbol{n_i}$	an Euclidean angular moment three vector
$\boldsymbol{O_i}$	a pose of a point that lies on the axis of rotation $\boldsymbol{s}$
$\dot{\boldsymbol{\omega}}_{\boldsymbol{i}}$	an Euclidean angular acceleration three vector
$\boldsymbol{r_i}$	a position vector from the center of mass point to a point on the axis of rotation $\boldsymbol{s}$
$\boldsymbol{s_i}$	an Euclidean axial vector representing the axis of rotation
$\boldsymbol{F_{i/(\cdot)}}$	a spatial force six vector/wrench with coordinate semantics
$\boldsymbol{H_{i/(\cdot)}}$	a spatial inertia of a rigid body with coordinate semantics
$\boldsymbol{X_{(\cdot)}}$	pose with coordinate semantics
$\dot{\boldsymbol{X}}_{\boldsymbol{(\cdot)}}$	velocity twist with coordinate semantics
$\ddot{\boldsymbol{X}}_{\boldsymbol{i/(\cdot)}}$	a spatial acceleration six vector/acceleration twist with coordinate semantics
$F_{(\cdot)}$	wrench with semantics
H_i	a rigid body inertia (contains its mass and moment of inertia) with semantics
p	a spatial momentum of a rigid body with semantics
$X_{(\cdot)}$	pose with semantics
$\dot{X}_{(\cdot)}$	velocity twist with semantics/a spatial/Cartesian representation of a velocity
$\ddot{X}_{(\cdot)}$	acceleration twist with semantics
$\boldsymbol{A}$, $\boldsymbol{B}$, $\boldsymbol{C}$	body names when used in context of geometric relations semantics
$[F]$, $[L]$	orientation frame names when used in context of geometric relations semantics
f, l	point names when used in context of geometric relations semantics

$\{f\}$, $\{l\}$	pose frame (an orientation frame with a point at its origin) names when used in context of geometric relations semantics
$\bar{F}^6$	a six dimensional space of force vectors
$\hat{S}_C$	a subspace of $\bar{F}^6$ of force vectors
$\bar{M}^6$	a six dimensional space of motion vectors
$\mathbb{N}$	a natural number set
$\hat{S}_M$	a subspace of $\bar{M}^6$ of motion vectors
$\mathbf{L}_{joints}$	an ordered set of joints
$\boldsymbol{S}, \boldsymbol{S(q)}$	a matrix representation of $\hat{S}_M$ subspace of motion vectors
${}^{i+1}\boldsymbol{Tf}_i$	a coordinate transformation for force vectors or a screw force wrench and a momentum transformation matrix that transforms a vector in coordinate $\{i\}$ to coordinate $\{i+1\}$
${}^{i+1}\boldsymbol{Tv}_i$	a coordinate transformation for motion vectors or a screw velocity and an acceleration twist transformation matrix that transforms a vector in coordinate $\{i\}$ to coordinate $\{i+1\}$

Chapter 3

Z	one of the three pairwise orthogonal axis of a coordinate frame in a three dimensional Euclidean space
$\boldsymbol{Z}$	a unit vector along Z axis of a coordinate frame in a three dimensional Euclidean space
$\hat{\boldsymbol{b}}\boldsymbol{(q,\dot{q})}$	bias acceleration term obtained by differentiating holonomic position constraint
$\boldsymbol{\alpha}$, $\bar{\boldsymbol{\alpha}}$	a scalar factor
$\boldsymbol{A}_N$	a $6 \times m$ matrix of direction of unit constraint force applied at Nth segment
$\boldsymbol{b}_N$	a $m \times 1$ matrix of acceleration energy at Nth segment
$\boldsymbol{w}_{ee}$	a scalar weighing factor of the acceleration energy in end-effector space

$\boldsymbol{w}_{posture}$	a scalar weighing factor of the acceleration energy in posture/joint space
$\boldsymbol{F}^A$	a articulated body bias force
$\boldsymbol{F}^a$	an apparent articulated body bias force
$\boldsymbol{F}_{bias}$	a spatial bias force on a rigid body
$\boldsymbol{F}$	a spatial active net force on a rigid body
$\boldsymbol{F_c}$	a spatial constraint force
$\boldsymbol{F^{ext}}$	a net external spatial force on a rigid body
$\boldsymbol{f}$	a linear three vector component of a spatial force/screw force wrench
$\boldsymbol{F^b_{bias}}$	a bias force component of a net spatial force that arises because of the non-vanishing angular velocity in moving coordinates
$\boldsymbol{F^b}$	a net spatial force computed from a spatial acceleration of a rigid body
$\boldsymbol{\tau}$	an angular three vector component of a spatial force/screw force wrench; torque applied at the joint
$\boldsymbol{g}$	a gravitional acceleration
$\boldsymbol{H}^A$	a positive definite articulated body inertia matrix
$\boldsymbol{H}^a$	a positive definite apparent inertia matrix
$\boldsymbol{J_c(q)}$	constraint Jacobian matrix in generalized coordinates
$\boldsymbol{\lambda}$	Lagrangian multiplier or magnitude of a force vector
$\boldsymbol{P}^A$	a positive semi-definite projection matrix
$\{p\}_i, \{d\}_i$	i^{th} proximal and i^{th} distal feature/joint attachment frames on a segment
$\boldsymbol{r}^{i,j}$	a position vector connecting origins of frames $\{i\}$ and $\{j\}$
$\boldsymbol{\tau_{fb}}$	a feedback contribution of computed torque
$\boldsymbol{\tau_{ff}}$	a feed-forward contribution of computed torque
$\dot{\boldsymbol{X}}\times$	a spatial cross product operator

$\mapsto$	a mapping operator that maps vectors from one space onto another
$\boldsymbol{v}$	a linear three vector component of a spatial velocity/screw velocity twist
$\dot{\boldsymbol{v}}$	a linear three vector component of a spatial acceleration/screw acceleration twist
$\ddot{\boldsymbol{X}}_{\mathbf{bias}}$	a bias acceleration twist component of a complete spatial acceleration of a moving rigid body due to a non-vanishing angular velocity
$\dot{\boldsymbol{X}}_{\mathbf{joint}}$	a velocity twist contribution at the joint
$\ddot{\boldsymbol{X}}_{\mathbf{joint}}$	an acceleration twist contribution at the joint
$\boldsymbol{\omega}$	an angular three vector component of a spatial velocity/screw velocity twist
$\dot{\boldsymbol{\omega}}$	an angular three vector component of a spatial acceleration/screw acceleration twist
$\boldsymbol{\mathcal{Z}}$	a scalar quantity representing 'Zwang', a degree of constraint
$\boldsymbol{D}$	combined inertia on the joint shaft
$\boldsymbol{D}^{-1}$	inverse of combined inertia on the joint shaft
$\boldsymbol{d_i}$	inertia of a transmission unit connecting a joint to a motor
$\boldsymbol{\mathcal{L}}$	$m \times m$ acceleration constraint coupling matrix
$\boldsymbol{\nu}$	a $m \times 1$ vector of magnitudes of constraint forces; Lagrange multiplier
$\boldsymbol{U}$	constraint acceleration energy generated by inertial and external forces

Chapter 4

$\boldsymbol{\gamma^\circ, \beta^\circ, \alpha^\circ}$	rotation angles about X,Y and Z axes respectively.
$\bar{\boldsymbol{b}}(\boldsymbol{q}, \dot{\boldsymbol{q}})$	a stabilized bias acceleration term; Cartesian acceleration resolved control law
$\boldsymbol{\mathcal{I}}$	a cost function

$\boldsymbol{\Delta q}$	vector of joint pose error
$\boldsymbol{\Delta \dot{q}}$	vector of joint twist error
$\boldsymbol{\Delta \ddot{q}}$	vector of joint acceleration twist error
$\boldsymbol{\Delta X}$	vector of Cartesian pose error
$\boldsymbol{\Delta \dot{X}}$	vector of Cartesian velocity twist error
$\boldsymbol{\Delta \ddot{X}}$	vector of Cartesian acceleration twist error
$\boldsymbol{f(q,t)}$	a polynomial function that is used to parametrize a trajectory
$\boldsymbol{\alpha, \hat{\alpha}}$	a differential gain for stabilization
$\boldsymbol{\eta}$	a integral gain for stabilization
$\boldsymbol{\beta, \hat{\beta}}$	a proportional gain for stabilization
$\boldsymbol{K_D}$	a differential gain/a damping
$\boldsymbol{K_{Dc}}$	a Cartesian differential gain/a Cartesian damping
$\boldsymbol{K_{Dj}}$	a Joint space differential gain/a Joint space damping
$\boldsymbol{K_P}$	a proportional gain/a stiffness
$\boldsymbol{K_{Pc}}$	a Cartesian proportional gain/a Cartesian stiffness
$\boldsymbol{K_{Pj}}$	a Joint space proportional gain/a Joint space stiffness
$\boldsymbol{J}$	a Jacobian matrix
$\boldsymbol{L_i}$	a distance between two joint attachment frames in a serial kinematic chain on a plane
$\boldsymbol{W}$	a positive semi-definite weighing matrix

Chapter 5

$\boldsymbol{h^{ee}(q)}$	a holonomic position contstraint in Cartesian/end-effector coordinates
$\boldsymbol{\hat{J}_c^{ee}(q)}$	a Jacobian matrix that maps Cartesian space motion constraints onto the posture space motions
$\boldsymbol{\hat{J}_c^{p}(q)}$	a Jacobian matrix that maps posture space motion constraints onto itself, often an identity matrix

$\hat{\boldsymbol{J}}_{\boldsymbol{c}}^{\boldsymbol{s}}(\boldsymbol{q})$	a Jacobian matrix that maps sensor space motion constraints onto the posture space motions
$\boldsymbol{h}^{\boldsymbol{p}}(\boldsymbol{q})$	a holonomic position contstraint in joint/posture coordinates
$\boldsymbol{h}^{\boldsymbol{s}}(\boldsymbol{q})$	a holonomic position contstraint in sensor coordinates
$\boldsymbol{\tau}_{\boldsymbol{c}}^{\boldsymbol{ee}}(\boldsymbol{q})$	a constraint force in generalized coordinates incurred by Cartesian space end-effector constraints
$\boldsymbol{\tau}_{\boldsymbol{a}}^{\boldsymbol{p}}(\boldsymbol{q})$	an active input force in generalized coordinates required to satisfy posture space constraints
$\boldsymbol{\tau}_{\boldsymbol{c}}^{\boldsymbol{s}}(\boldsymbol{q})$	a constraint force in generalized coordinates incurred by the sensor space constraints
$\boldsymbol{E}_{(\cdot)}$	error function of position, velocity, acceleration, torque, and controller gains in a cost/ penalty function in MPC formulation
$\boldsymbol{K}_{(\cdot)}$	gains in error function of controller gains
$\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c}, \boldsymbol{d}, \boldsymbol{f}, \boldsymbol{g}$	weights of the error terms in a total cost/penalty function in MPC formulation
$\boldsymbol{Q}_{(\cdot)}$	weighing matrix in error function of position, velocity, acceleration, torque, and controller gains

Other Symbols

$\dot{(\cdot)}$	a first order complete derivative of a matrix
$\ddot{(\cdot)}$	a second order complete derivative of a matrix
$(\cdot)^{-1}$	an inverse of a matrix
$(\cdot)^{\boldsymbol{T}}$	a transposition of a matrix
$(\cdot)_{\boldsymbol{d}}$	a desired value of a variable

Contents

Acknowledgments v

Abstract vii

Contents xxvii

List of Figures xxxi

List of Tables xxxv

1 Introduction 1

1.1 Motivation 5

1.2 Problem Statement 11

1.3 Approach 12

1.3.1 MDE in Task Programming 13

1.3.2 Task Programming Stack as a Composition of DSLs . . 14

1.4 Contributions 19

1.5 Related Work 20

1.5.1 Variability of Robot Models 20

1.5.2 Software Composition and Physical Systems 22

1.5.3 Design and development of DSLs in robotics 25

1.5.4 Robot Control and Optimization 27
1.6 Overview of the Thesis 30

2 Semantic Modeling for Composable DSLs 33
2.1 Introduction 33
2.2 Example 34
2.3 Related work 37
2.4 Semantic models for the kinematic structures 41
2.4.1 Models of the structural primitives 42
2.4.2 Models of the operational primitives 48
2.5 Software design 54
2.6 Example 60
2.7 Discussions and conclusions 65

3 Popov-Vereshchagin Hybrid Dynamics Solver 69
3.1 Introduction 69
3.2 Related work 71
3.3 Segment-to-segment recursive dynamics 72
3.3.1 Outward position, velocity and acceleration recursions . 75
3.3.2 Inward force and inertia recursions 76
3.3.3 Tree-structured chains 79
3.3.4 Outward recursion of control torques and segment accelerations 80
3.4 Popov-Vereshchagin's acceleration-contrained dynamics algorithm 81
3.4.1 Integrating acceleration constraint into the dynamics recursions 84
3.5 Extensions 88
3.5.1 Multiple constraints in a tree 88

3.5.2 Posture/Null space control . . . 90
3.5.3 Novel linear-time solution for task prioritization and constraint weighting . . . 91
3.5.4 Unilateral and maximum joint torque constraints . . . 92
3.5.5 Numerical considerations . . . 93
3.6 Discussions and conclusions . . . 94

4 Role of Popov-Vereshchagin Solver in Control 97
4.1 Related work . . . 97
4.2 Validation in simulation . . . 102
4.2.1 Results and analysis . . . 107
4.3 Discussions and conclusions . . . 137

5 Examples of implementation of motion control stack 139
5.1 Introduction . . . 139
5.2 General structure of WBCA . . . 141
5.3 WBCA with the inverse kinematics and dynamics solvers and the Cartesian impedance controller for the youBot mobile manipulator robot . . . 144
5.4 WBCA with the hybrid dynamics solver and a controller with adaptive gains for the youBot manipulator robot . . . 146
5.4.1 Controller based on predictive gain adaptation. . . . 148
5.5 An implementation of WBCA using the motion programming stack . . . 150
5.6 Task control applications: Linked data with semantics . . . 154
5.7 Discussions and conclusions . . . 157

6 Discussions and conclusions 159

A JSON representation for the semantic models of kinematic chains 165

B Receding horizon predictive gain adaptation algorithm **169**

Bibliography **177**

Publications **199**

List of Figures

1.1 Complex robotic systems. 2

1.2 A symbolic representation of a task. 4

1.3 Geometric primitives, constraints and constraint controllers on a kinematic chain. 6

1.4 The layers of a task control framework. 10

1.5 MDE model. 15

1.6 Task programming stack as a composition of DSLs. 16

1.7 The orthogonal relationship between the task programming stack and the models in MDE. 18

1.8 A variability space of a robot application. 21

1.9 A control block diagram implementation of some task. 24

2.1 Target object and robot specification DSLs and their position in the task programming stack. 34

2.2 The motion of a two-body articulated system is described using 3D and 6D vectors. 35

2.3 The relations between the physical, geometric and coordinate models. 41

2.4 The relationships between physical primitives and their concrete representations. 42

2.5 The semantics of the Link primitive. 43

2.6 The semantics of the Segment primitive. 44
2.7 The semantics of the Joint primitive. 45
2.8 The semantics of the Chain primitive. 47
2.9 A Model-to-code transformation process for the models represented in JavaScript Object Notation(JSON) and C++ languages. 55
2.10 A model composition using JSON schema 56
2.11 Class diagram of the structural primitives. 59
2.12 Class diagram of the operational primitives 60
2.13 A serial kinematic mechanism and its segment and joint frames 62

3.1 A position of a solver specification DSL(s) in the motion stack. 70
3.2 Reference attachment pose frames and notation for segments in a tree-structured kinematic chain. 73
3.3 Computational sweeps in a domain specific solver 74
3.4 A feedforward and a feedback input decomposition for priority and weighting-based control. 92

4.1 A position of a control specification in the motion stack. 98
4.2 Three complementary ways to build controllers around the Popov-Vereshchagin algorithm 101
4.3 A control block diagram of the interactions between the controllers and the constrained dynamics solver 103
4.4 Three configurations of the serial kinematic chains and the constraints used in simulations. 104
4.5 The simulation data for a 2-DoF planar manipulator with constraints on the end-effector position in X and Y DoF 114
4.6 A comparison of six different controller setups using priority and weighting based approaches with the non-conflicting constraints. 121
4.7 A comparison of six different controller setups using priority and weighting based approaches with the conflicting constraints . . 125

4.8 Joint torque data for the setups with conflicting and non-conflicting constraints on 4-DoF serial chain 127

4.9 A comparison of the acceleration energy and impedance based end-effector controllers for the 5-DoF serial chain under the conflicting constraints. 132

4.10 A comparison of constraint stabilization using impedance and joint space controllers for a 7-DoF manipulator model 136

5.1 An implementation of the WBCA can be expressed using a composition of various DSLs that are part of the (motion)task programming stack. 140

5.2 A block diagram of a generic whole-body control architecture. . 142

5.3 An 8-DoF serial chain model of youBot mobile manipulator . . 144

5.4 A whole-body control architecture for the youBot mobile manipulator using a virtual external force input to the dynamics solver 145

5.5 The behavior of the greedy receding horizon prediction-based adaptation algorithm. 147

5.6 The integration of the receding horizon prediction-based adaptation algorithm in WBCA. 149

5.7 A software component architecture for the mobile manipulation application. 151

5.8 State machine diagram of the application that realizes the architecture in Figure 5.4. Here, the transitions between the states are triggered by the events $\boldsymbol{e}\_(\cdot)$. 153

5.9 A linked data graph model of a robot platform 156

B.1 A performance comparison between a conventional and a predictive gain adaptation algorithm based control schema for the tracking of a circular trajectory 171

B.2 A performance comparison between a conventional and a predictive gain adaptation algorithm based control schema for the tracking of an infinity sign trajectory 173

B.3 Dynamics of the controller gains during the prediction phase. . 174

B.4 The influence of the cost function on the performance of the tracking control 175

B.5 The influence of the cost function on a set-point control with constraints on joint torques. 176

List of Tables

2.1 Summary of the motion and effort vector relations using two 3D vectors and one 6D vector representations. 36

2.2 Minimal geometric semantics for the pose, twist and wrench of a body. 40

2.3 Composition semantics of the structural models 49

2.4 The geometric relations and a list of semantically valid operations. 50

4.1 Various combinations of constraints, controllers and kinematic chains that were used in the simulations with the conventional control scheme (Figure 4.3). 106

4.2 Controllers and their gain configurations as used in priority-based multitask control with non-conflicting constraints for 4-DoF robot 108

5.1 Cost functions and the errors $\boldsymbol{E}_{(\cdot)}$ they penalize. Here, a, b, c, d, f and g are weights. 150

Chapter 1

Introduction

In the past several years, there has been a large growth in the development of robotics systems, both in numbers as in the variety and complexity of their tasks. However, no progress at all has been made in giving the software driving these systems explicit formal models, with automatically verifiable and validatable semantics. Hence, all of the mentioned efforts typically remain limited to closely cooperating development teams, who have the opportunity to bypass formal validation via informal interactions between developers, and via lots of studying of the internals of the source code. This is not a sustainable development approach, if complex robotics systems should ever enter real-world markets, with long maintenance and update lifetimes, and third-party integrations of ever more complex systems-of-systems. Bringing a fundamental change in this unfortunate lack of formalization and standardization is the major motivation behind this PhD research.

A good example of the above-mentioned systems and demonstrations that were shown in the recent past are the DARPA Grand and Urban Challenges [DARPA, 2004a, DARPA, 2005, DARPA, 2004b], DARPA Robotic Challenges [DARPA, 2013], RoboCup@Home [RoboCup, 2015a], RoboCup@Work [RoboCup, 2015b]. These demonstrations focus on the development of autonomous capabilities for a range of different platforms, such as wheeled autonomous robotic cars that could traverse different terrains, or humanoid platforms that could assist or re-place a human in dangerous and routine everyday activities. These robots usually have a quite capable hardware with many sensors, actuated degrees of freedom, and a sophisticated software platform. Figure 1.1 shows examples of such complex robot hardware.

The challenging conditions under which the robots should operate require

(a) (b)

Figure 1.1: Complex robotic systems: (a) HRP-4C humanoid robot developed for the entertainment industry [Humanoid Research Group, AIST, 2009]. (b) DLR Rolling Justin platform [Fuchs et al., 2009] is developed to assist with tasks in space.

the hardware to be accompanied by an equally capable software platform that allows the realization of various complex tasks. The dynamic nature of the interactions with the environment and more and more with humans, unavoidably leads to the development of complex models and programming approaches for these platforms. Moreover, the variety of these interactions, i.e., pure motion, or contact and motion, that need to be addressed by the programming approaches impose further requirements on the systems' design and development. This led to the development of many task control and programming methods, which had their origins in different robot application domains. Some of the well-known frameworks include: Task Frame Formalism (TFF) [Bruyninckx and De Schutter, 1996] with the initial foci on force-controlled robot applications and later extended to the Instantaneous Task Specification using Constraints (iTaSC) [De Schutter et al., 2007, Rutgeerts, 2007, Vanthienen et al., 2011b], the Stanford Whole-Body Control Framework(SWBC) [Khatib, 1983, Khatib, 1985] with its origins in (mobile) manipulator control in operational space and the Stack of Tasks (SoT) [Mansard and Chaumette, 2007] with the origins in the Task Function paradigm [Samson et al., 1991] and the extensions to visual servoing [Espiau et al., 1992] and humanoid robot control [Mansard et al., 2009] applications.

From a functional point of view, all these frameworks compute some form of

constrained task control. A constrained task control problem is complex and involves multiple other sub-problems that need to be addressed too, such as vision-based estimation, control, and scheduling. Therefore, many task control and programming frameworks strive for some symbolical representation that describes *what* the task problem is, rather than *how* its concrete realization looks like. Figure 1.2(a) shows one possible form of such a symbolic representation of the task. Often, this representation is also referred to as a task specification.

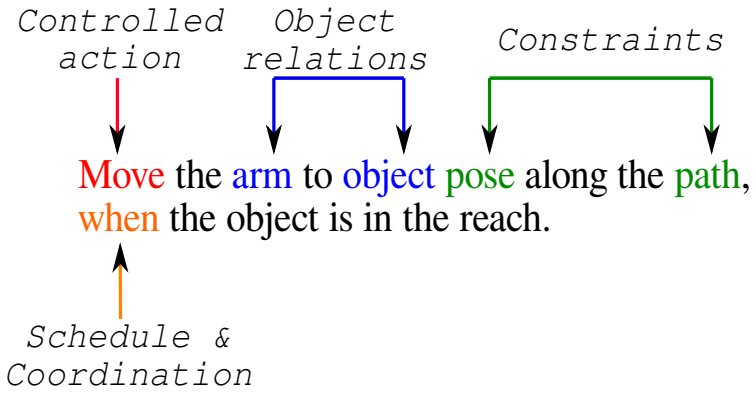
The task specification focuses on the definitions of the relations among the objects in a scene and the actions that satisfy these relations. Hence, it is also referred to as object and action centric specification [Smits, 2010]. As it is seen in Figure 1.2(a)–(b), in addition to *objects* (e.g., robots, environment objects) and *controlled actions*, there are two further components that are part of the task, task or application *constraints* on the object relations and *a schedule/coordination* of the controlled actions. Each of these components belongs to a domain of its own and complies with models and constraints in that domain's context. For instance, the task constraints are imposed on the physical relations, such as time, pose, twist, or wrench. They follow either from motions or interactions of the robot with its environment. Figure 1.2(c) depicts a visual servoing application where the constraints are imposed on various feature frames involved in the tasks, e.g., a relative pose of the robot base in the world, a pose of the object in the camera space. In order to complete the task successfully, every constraint need to be controlled accordingly. Figure 1.3(c)–(d) summarize various constraints and the ways to control them. Depending on the task that the robot is to perform, the constraints and the controllers are defined on the physical quantities that are either in joint, Cartesian, or sensor spaces. The integration of multiple such constraints and controllers allows realizations of different complex tasks.

Furthermore, the symbolic task specifications are attractive because they are independent of the application being developed and the type of robot platform used. This may lead to the assumption that if a set of basic recurring modelling primitives for the constraints, the controllers and the other task constituents were identified, one could develop correct-by-construction complex robot applications by *composing* the individual tasks according to some well defined set of rules, Definition 1. This is a vision of any task control and programming framework.

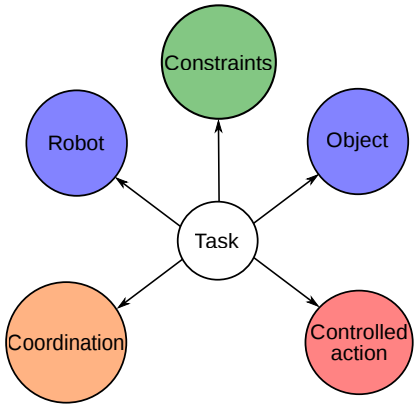
Definition 1. *Composition refers to the systematic construction of applications from components that implement abstractions pertaining to a particular problem domain [Nierstrasz and Meijler, 1995, Sanatnama et al., 2008].*

It is in an analogy to a superposition principle in linear-time dynamical systems, where the behavioral semantics of the constituting dynamical components, and the composition rules to integrate them are widely accepted. An equivalent of

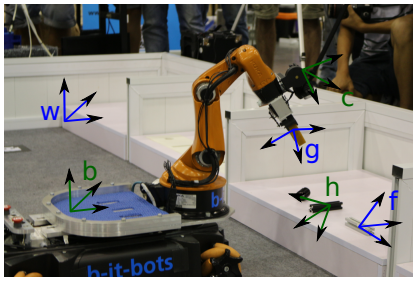
this principle in the context of software for physical systems is *compositionality*, Definition 2. On the other hand, for the programming of robotic systems no similarly clean, structured and unambiguous semantics has been defined yet. Therefore, the integration of the pieces of the robotic application still remains an artisanal practice.



(a)



(b)



(c)

Figure 1.2: Many task control and programming frameworks strive to describe tasks in a symbolic form. This abstract specification allows to focus on what the task is, rather than how this task is concretely implemented. Often, such symbolic formulations contain knowledge about the objects in the scene that are part of the task, the constraints on the objects' relations, the controlled interactions between the objects and the scheduling or the coordination of these interactions.

Definition 2. *Compositionality refers to a condition of any system, where the system's behavior or property can be inferred from the behaviors or the properties of its constituting components [Chakrabarti et al., 2003, Doyen et al., 2008, de Alfaro and Henzinger, 2001, Goessler and Sifakis, 2005].*

The concept of *domain semantics* or *semantic models* is at the basis of the modelling approach and all hypotheses put forward in this doctoral thesis. Therefore, any following use of this concept complies with Definition 3.

Definition 3. *Semantics of a domain is defined in terms of a necessary minimum number of structural primitives, their operations and the constraints they need to satisfy to unambiguously define all models in the domain.*

This thesis sets out to research the factors that influence a correct-by-construction composition of components in task specification and programming. In particular, it will look into the definition of coordinate-independent semantic models for the specification of the robot kinematic structures and algorithms. It will also analyze and extend these semantic models in the broader scope of motion task control architectures.

1.1 Motivation

Recently, the robotics research community has been flooded with interesting robotic platforms such as Willow Garage's PR2 [Wyrobek et al., 2008], KUKA youBot [KUKA, 2010], Baxter [Rethink Robotics, 2012], and Care-O-bot [Fraunhofer IPA, 2015]. Many of these platforms come with different degrees of software support. One of the main aims during the development of these platforms was to reuse some of the existing software functionalities across these platforms or port mature technologies onto the industrial applications. In order to support these activities, there have been a number of large funded projects such as RoboHow [RoboHow, 2015], RoSta [Hägele, 2011], BRICS [Bischoff et al., 2010], ROSETTA [ROSETTA, 2015], SMERobotics [Hägele and Hans, 2005], etc. Even though these projects advanced progress in developing software for robot applications to some extent, they still fell short of delivering the promised results.

For instance, the youBot platform, which was developed as a promising educational platform in the context of BRICS project, comes as bare-bone system without any software support for the motion control primitives. As a consequence, it requires an engineer with expertise in multiple domains to

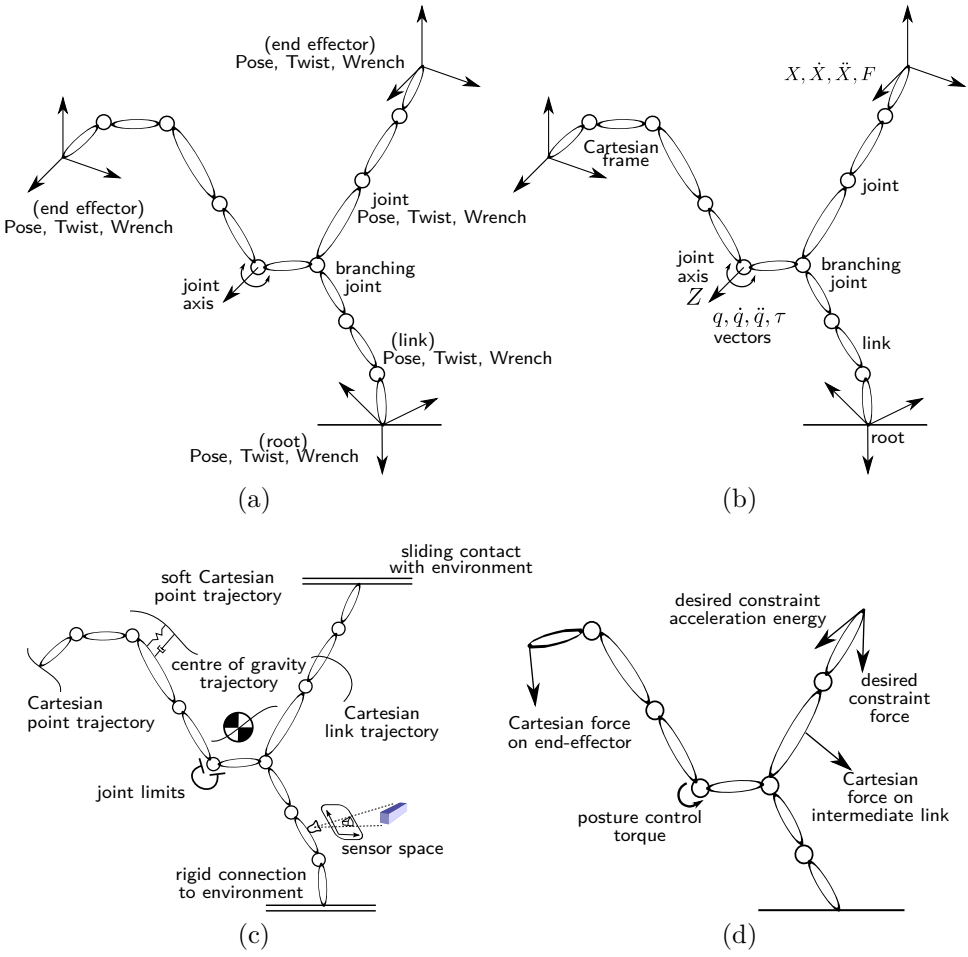


Figure 1.3: (a)Various geometric relations that are associated with joints and links of a kinematic chain; (b) notations and specific physical representations that are used to define the geometric relations; (c) a possible combination of task constraints. The nature of these constraints vary, e.g., geometric, contact, physical constraints of a platform, etc., and they can be applied at the same time. All constraints build upon geometric relations; (d) variety in the ways constraints can be controlled.

build and understand complete functional motion control applications. The PR2 platform has all the software functionalities for motion and interaction programming, but it requires a lot of effort to configure and to adapt this software to the different application setups. In the context of the Robo-

```
1  class JTask : public opspace::Task
2  {
3  public:
4    JTask() : opspace::Task("application::JTask") {}
5    virtual jspace::Status update(jspace::Model const & model)
6    {
7      // Update the state of our task.
8      actual = model.getState().position;
9      // Compute PD control torques and store them in command_
           for
10     // later retrieval.
11     command = kp * (goal - actual) - kd * model.getState().
           velocity;
12     jspace::Status ok;
13     return ok;
14   }
15   double kp, kd;
16   jspace::Vector goal;
17 };
```

Listing 1.1: An example of joint space task specification and control in one of the Whole-Body Control (WBC) task control frameworks. This is an excerpt of code from [Stanford Robotics and AI Lab, 2011, Philippsen et al., 2011].

How [RoboHow, 2015] project, it was a real challenge to integrate constraint solvers from the SoT [Escande et al., 2010, Mansard and Chaumette, 2007] task programming framework in another task programming framework iTaSC [De Schutter et al., 2007, Vanthienen et al., 2011b], though both of the frameworks had the same type of components and target the same types of applications. So, why is it difficult to prototype a robot application, in spite of so much progress in robotics research and multitude of robot task programming and control frameworks? In order to exemplify some of the issues, take a look at two following examples.

Example 1: Code listing 1.1 shows an excerpt of a joint space task control program for the two branched robot in Figure 1.3 and implemented in the Stanford Whole-Body Control (WBC) software framework [Philippsen et al., 2011]. Here, on line 5 *jspace::Model* contains data on the robot's geometrical and dynamical properties such as a link's length, a joint's type, a link's mass and inertia, a joint's inertia, etc. Line 8 gets an update of a *position vector* from the model and this is used together with a *velocity vector* of the joints to define a control command on line 11. The joint's geometric quantities that are used in the controller are depicted in Figure 1.3(b) and associated torque controller in Figure 1.3(d).

Now assume that the desired robot model is not available in the form that SWBC framework needs, but is present in another library implementation, e.g.,

```
1  // The vector v_base_AB is expressed in the same base as the
       twist
2  // The vector v_base_AB is a vector from the old point to the
       new point.
3
4  joint.twist.RefPoint(const Vector& v_base_AB) = Twist(vel+rot*
       v_base_AB,rot)
5  segment.twist = joint.twist(qdot).RefPoint(joint.pose(q).M *
       f_tip.p)
```

Listing 1.2: A common form of representation in conventional kinematics and dynamics libraries. It requires a lot of effort to check semantic validity and the results of such expressions. This is an excerpt of code from [Smits et al., 2001].

KDL [Smits et al., 2001]. One is still interested in joint *positions* and *velocities* for the controller. But, as is seen on the lines 4-5 of listing 1.2, the joint uses different representations, i.e., *pose* and a *twist* instead, where *pose* seems to consist of an *orientation matrix* and a *position vector*. This is a problem, because the implications of these primitives on the models are not explicit. Further, in order to use the implementation of a given robot model, the physical quantities in KDL model need to be converted to their WBC counterparts. In order to implement this correctly, the physical constraints associated with those quantities need to be made explicit, while one single invalid conversion might easily be overlooked and cause various unpredictable problems across the system, as well as logging debugging times.

Such problems are very common in robotics, because every functional library opts for its own specific *representation* to describe the same physical (geometric) quantities. Additionally, every representation may impose extra representation-specific *constraints* on computations which need to be accounted for. For instance, minimal orientation representations as Euler angles always have singularities, while non-minimal representations as orientation matrices do not. The problem of representation can also be elaborated with the help of figures 1.3(a) and 1.3(b). In 1.3(a), the physical quantities (e.g., wrench, pose) do not have any coordinate or representation specific data associated, whereas in 1.3(b) the notation already implies that the *pose* of an end-effector point is using a frame primitive and joint values $q, \dot{q}, \ddot{q}$ are *vectors*.

Example 2: another problem, which is complementary to the representation problem in example 1, is described by Figure 1.4. It shows a generic layered view on the structural organization of contemporary robot task programming software. The view distinguishes three tiers of specification.

- Task layer specifications (programs) are generic representations of the task model primitives, i.e., controlled actions, constraints among objects,

etc. They describe *what* needs to be done. Figure 1.2 depicts an example of such a task specification program.

- Motion layer specifications describe *how* task models are realized using constrained motion programming approaches. It represents a transformation of controlled actions, constraints on object relations, schedule and coordination into their concrete counterparts, e.g., minimum time third-order polynomial trajectory tracking defined between robot's end-effector pose frame and some object's center of geometry pose frame, where the tracking controller is acceleration-resolved.

- Robot layer specifications implement concrete constrained motion control approaches using software architectures on some specific computational hardware platform. On this layer robot-specific constraints such as geometric and inertial data, physical constraints on joints, etc., are taken into account.

Most of the contemporary constrained task control frameworks, such as iTaSC [De Schutter et al., 2007, Smits et al., 2008, Vanthienen et al., 2011a, Vanthienen et al., 2011b], SWBC [Khatib et al., 1999, Khatib et al., 2004b, Khatib et al., 2003, Khatib et al., 2008], SoT [Mansard and Chaumette, 2007, Mansard et al., 2009] and Visual Servoing Platform (ViSP) [Chaumette, 1998, Chaumette and Hutchinson, 2006, Chaumette and Hutchinson, 2007] implement motion, robot, and to some extent task layer specifications.

The main problem here is that, even though these task control approaches address the same set of problems and applications, they originated in different application domains and had different research foci. This legacy is reflected in their software realizations that often adopt a specifically tailored combination of representations, approaches, and components. This is shown in Figure 1.4 with examples of hypothetical frameworks A, B, and C, which represent some task programming frameworks. Such a situation inadvertently leads to the fact that the same constrained task specification problem cannot be or difficult to implement by the different frameworks. This is directly related to the *composability* of the task framework's components, Definition 4.

Definition 4. *Composability refers to a condition of a system, where system properties will remain valid under some local conditions after it is composed [Chakrabarti et al., 2003, Doyen et al., 2008, de Alfaro and Henzinger, 2001, Goessler and Sifakis, 2005].*

Thus, one can deduce that *if the semantics and constraints of the modelling or programming primitives are coupled with their specific representations and constraints, the result of a new configuration of the same primitives is often*

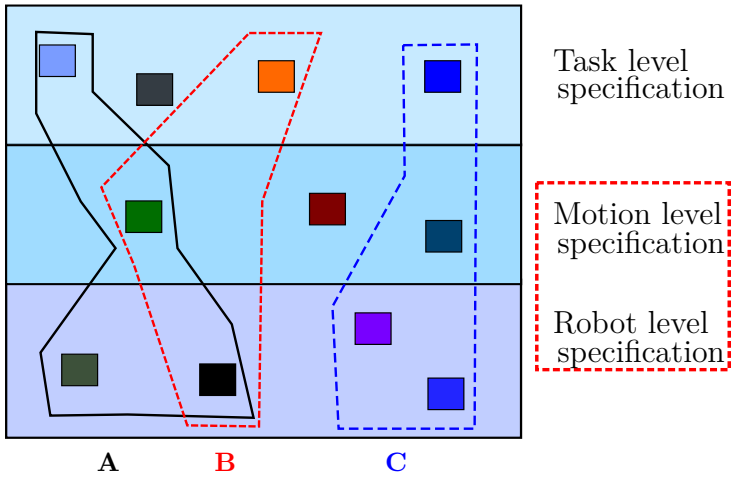


Figure 1.4: The layers in task control frameworks may represent the model abstractions, interacting library implementations or the direction of information flow. For instance, a path planner on the task layer can interact with a Cartesian velocity controller on the motion layer. The issue here is that every framework uses its own specific combination of representations and methods for the same functional components. For instance, the velocity output of the controller can be a screw twist or a pose twist.

unpredictable [Goessler and Sifakis, 2005]. This statement relates to many traditional task programming frameworks and it can concretely be exemplified with the following set of questions with respect to the setup in Figure 1.3.

- How would the task control program using a concrete representation behave, if some of the constraints on two-branched kinematic chain were removed or modified?
- How would the constraint controllers react, if an additional number of branches were added to the kinematic chain?
- What would the relevance of the previous two actions be, if the controlled motion were either velocity or acceleration-resolved?

Each of these questions touches different aspects of the robot's and task's models at different levels of detail. The more detailed is the model, the more conditions and as a result, the more composition constraints need to be defined and satisfied. It also implies that the composition process needs to take place across all layers of task control and programming.

This thesis focuses on some of the issues of the composition and composability of the task components in the context of motion layer and robot layer specifications (Fig. 1.4) for the applications using robot manipulators. In order to identify the motion primitives and to define the associated composition constraints, this research will specifically look into various configurations of the whole-body motion control architecture and motion task programming primitives.

1.2 Problem Statement

This PhD thesis addresses the following complementary research problems in the broader context of constrained (motion) task programming and control:

- **Semantic consistency in kinematic chain models:**

  Problem - Current approaches to kinematic chain modelling are frequently created with a specific set of use cases in mind. These are pragmatic solutions that are meant to work with some concrete set of libraries and tools. Therefore, many of these modelling approaches are locked into, on one hand, a specific representation language semantics and syntax, and on the other hand, a specific coordinate representation.

  Implications - The absence, in kinematic chain specifications, of explicit semantic/physical models, which are invariant to specific coordinate representations, can lead to the design of incorrect computational algorithms and control architectures around these kinematic chains. Some common questions that arise in this respect are: which coordinate-specific constraints do external wrenches applied to the tip of each segment introduce? How do these constraints influence the design of impedance controllers? In what coordinates should the joint pose and twist errors be expressed in computed-torque controllers?

  A solution to this problem is presented in Chapter 2.

- **Composability of computational algorithms for kinematic chains:**

  Problem - Most of the constituting components of the constrained task control applications, e.g., the controlled action, the solver, are not composable. This is because they often hide implicit knowledge about the conditions under which they are valid and should interact with each other. Furthermore, the models and the realizations of these task components couple the structural aspects, e.g state information, with the behavioral aspects, e.g., state update computations.

  Implications - The absence of the explicit constraints on the structure and the behavior of the task components hinders their incremental and

deterministic composition. Some common questions that arise in this respect are: how can a hybrid dynamics algorithm for a complex kinematic chain be correctly constructed from the available pose, twist and wrench computations on a single segment? Here, the kinematic chain is the composition of multiple segments and joints. Another example is the composition of the complex computations from the simpler ones. Is it possible and meaningful to compute an acceleration twist of the end-effector before its pose and velocity twists are computed?

A solution to this problem is presented in Chapters 2 and 3.

- **Variability in whole-body control architecture:**

  Problem - Many task control frameworks are often presented as monolithic architectures that implement some form of whole-body control and specialize on a particular application domain, e.g., a visual servoing. This implies that they stick to a specific set of choices of components and architectures, which frequently couple the task specification problem with its context specific realization. This is the result of the failure to recognize variability points in the implementation of the whole-body control architecture, i.e., which task control components and their parameters should be configurable, in order to satisfy the requirements of different applications?

  Implications - This limits not only a diversity of the tasks that the framework can implement, but also future extensions of its constituting components. This can also lead to the situations, where the same problem specification can not be realized in the frameworks with the same underlying paradigm.

  A solution to such problems is presented in Chapter 5.

1.3 Approach

The approach follows two hypotheses that will help to address the problems in Section 1.2.

- First, it is assumed that there are two aspects of any model: its semantics in some application context and its concrete implementation. Thus, the approach focuses on decoupling these two aspects, in order to compose them in a systematic way. It relies on a Model Driven Engineering (MDE)-based methodology to realize this procedure [Bézivin, 2005]. In this context, the identified semantic models and constraints are allocated on the appropriate levels of abstractions in

MDE. Further, these models are encoded in the form of Domain Specific Languages (DSLs) [Voelter et al., 2013, Fowler, 2010]. A composition of multiple these DSLs defines either functional or non-functional aspects of the task control application. These DSL programs can then be translated by the modelling and code generations tool-chains into concrete implementations.

- Second, it is assumed that the principle of separation of concerns allows a decomposition of the models into their structural and their behavioral constituents. This is particularly important on the implementation level, because it allows a predictable composition of concrete implementations of the models. This is achieved by employing metaprogramming techniques [Abrahams and Gurtovoy, 2004, Alexandrescu, 2001] in combination with MDE.

The following sections provide detailed information on how to apply these hypotheses in the context of constrained task programming and control.

1.3.1 MDE in Task Programming

MDE is a software development approach with an emphasis on creating and using domain specific models to develop applications. It identifies recurring patterns and primitives of the domain. These are at the core of the domain specific tool-chains, which can be utilized to implement further applications or models. Often, an integral part of such a tool-chain is a code-generation process. The output of the code-generation is a programming language specific code that implements modelled applications. Such an approach to an application development focuses on the design of the system architectures, rather than their specific implementations. Therefore, it improves the productivity and avoids language specific common programming problems [Kleppe et al., 2003, Kent, 2002, Bézivin, 2005]. There can be as many layers of modelling as desired, though the usefulness of higher layer models is questionable. Therefore, there are usually four [Atkinson and Kühne, 2003] of these modelling layers as depicted in Figure 1.5. The descriptions below are not strict one-to-one interpretation of the (meta) metamodelling and domain modelling as it is defined in MDE, but adapted in the context of this research.

- **M3**: *Domain independent* models, also known as *meta meta model* layer covers generic models that are invariant of the domain specific concepts and relations. A common example of such models are graphs. In the context of the robotics, the graph can represent a connectivity structure of a kinematic mechanism. Most of the real world robots can be

modelled as a specialized directed graphs. These are serially connected nodes and edges with either single child (a serial robot) or multiple children (branching humanoid robots) as depicted in Figure 1.3. Parallel robots or loop mechanisms can also be modelled by the graph primitives. Another prominent use of the graph models is an automaton model for a coordination.

- **M2**: *Domain specific* models, also known as *meta model* layer adds domain semantics onto the M3 layer models. For instance, in relation to our example of the kinematic structures, a joint primitive is a node in the graph model. It is the node with additional domain specific properties attached to it, e.g joint inertia, offset, pose, twist, axis of rotation, etc. These properties can themselves be modelled as the graph, e.g a graph of the physical states of the robot's joints or links.

- **M1**: *Application specific* models are specific instances of the M2 models tailored with respect to some application. Going back to our example in the previous item, a concrete robot's structural description, e.g., KUKA Light Weight Robot (LWR) [Albu-Schäffer et al., 2007a] or Universal Robot's UR10 [Universal Robot, 2015] robot, using M2 layer modelling primitives such as joints, segments, transmissions, etc., fall under this category.

- **M0**: *Implementation* layer is a concrete *computer language representation* of the modelling primitives on the layers above. Conventionally, it is created by the code generation tools that are developed using the domain independent and specific models.

It also needs to be noted that the research in this dissertation is not a pioneer in applying MDE to robot software. There have been others who have done it [Schlegel et al., 2012, Steck and Schlegel, 2010, Vanthienen, 2015, Vanthienen et al., 2014]. But the focus of those efforts were on the structural descriptions of the software components and architectures, whereas this PhD research applies the MDE methodology to the specification and the implementation of the functional algorithms and data, i.e., kinematic chains and dynamics algorithms.

1.3.2 Task Programming Stack as a Composition of DSLs

The applications and their models are often realized by the developers directly in the form of the functional libraries. The main issue with such an approach is that the domain specific semantics barely show in application programming interfaces (APIs) and are mostly hidden in the implementations.

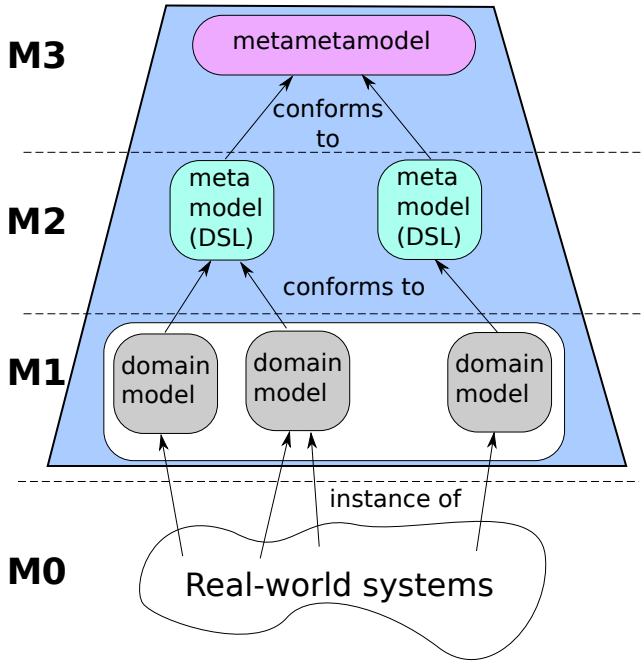


Figure 1.5: MDE metamodelling and modelling layers and their relationships. Moving down from M3 to M0 modelling and the resulting models become more concrete and constrained.

A representative example of such a situation was discussed in Section 1.1. The applications that rely on these functional APIs are bound to use some specific representations or algorithmic constraints that are imposed by the library. This can also affect the applicability context of that library. One approach to resolve this problem is to put as much domain specific semantics as possible into the APIs [Biggs and Makarenko, 2010]. Because of the limited expressivity of the interface signatures, such a solution can quickly become cumbersome *. Another, more elegant approach is to define separate computer representations, i.e., languages, for the semantics and its specific implementations as the MDE approach recommends. This leads to a collection of DSLs that are used to model each aspect, i.e., coordination, solver, robot structure, of the application independently of its specific realization. This approach is also completely *orthogonal* to the conceptual layered organization of the contemporary task control architectures. Figure 1.6 shows a group of DSLs across the conceptual

*The specific semantic information can also be put into interface contracts with pre- and post-conditions. But not many language infrastructures support them. Therefore, in many cases they are documented textually.

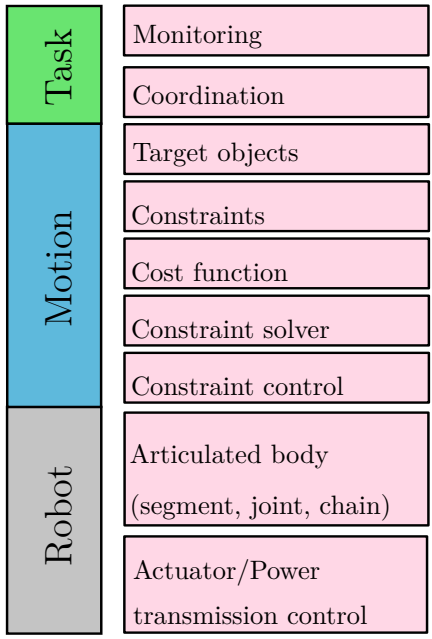


Figure 1.6: A constrained task control problem involves multiple sub-problems. Each of these sub-problems belongs to its own domain. This suggests that every aspect of these sub-problems can be described in its own DSL and the constrained task control be expressed as the composition of these DSLs. The conceptual stacked model groups a number of such DSLs together, in order to show that they contribute to the semantic description of the larger domain, such as Task, Motion and Robot. This stack of loosely coupled modelling and software components is required to turn task commands into motions for the robot tools, and further down to concrete actuator motions.

layers of the task programming stack. Each of the DSL blocks is responsible for the specification of one sub-problem of the task and the associated tool-chain generates its concrete implementation. Ideally, this loosely coupled stack of DSLs should allow generation of concrete software implementation of the constrained task control application by defining explicit transformations from task commands into motions for the robot tools, and further down to concrete actuator motions. The organization of task layers in the form of the stack reflects this transformation flow as well.

A decision on where a specific DSL block belongs to is based on the analysis of

the existing implementations of the task specification frameworks and control architectures. An affiliation of the DSL with one of the layers does not exclude another DSL with the same functionality on the others. For instance, the monitoring functionality is not exclusive to the task layer, but can also appear at different levels of detail on the motion control and robot layers. Then, these DSLs of the same functionality need to explicitly interact with each other and with other DSLs on the same level. Such detailed relationships between the various DSLs across the layers are complex and impossible to convey by the conceptual models as in Figure 1.6. It requires a decomposition of every single model according to the MDE discussed in Section 1.3.1.

This is partially presented in Figure 1.7, which extends the conceptual stacked model in Figure 1.6 with the knowledge of the semantics and implementation of the DSLs as defined in MDE. It shows that for the robot and motion layers, the semantic models for the rigid and articulated body motions can be decomposed into the geometric and the coordinate-specific models. Since the semantics in kinematics and dynamics of kinematic chains is often reflected in their topology, the figure introduces topological models instead. It shows a one-to-many relationship between meta layer models and the lower layer concrete models. These relationships support the *transformation* process in MDE, where the (meta) meta models are refined to concrete models by the additional domain specific knowledge. This is valid for all the components that are part of the motion task specification, e.g., controllers, constraint solvers. For the motion models, there are $1 - to - N$ and $1 - to - M$ relationships between the *topological–to–geometric* and *geometric–to–coordinate-specific* models, respectively. Considering the number of DSLs on each layer, one can deduce how complex the relationships between DSL models can become. Chapter 2 provides insights on how to transition from the top level conceptual models as in Figures 1.6 and 1.7 to the more detailed complex models which can also express complex inter-DSL relations.

Thus, identifying the domain specific semantics of each task constituent, as well as making the relationships between their semantic models explicit allows to develop a systematic approach to address a correct-by-construction of the task specification by composing DSLs. For instance, a robot is modelled as connected rigid and flexible bodies. If the robot's joint is flexible, then an appropriate transmission type and actuator control need to be implemented. The specifications of the rigid bodies and the actuator control primitives are modelled by the separate DSLs [Ott, 2008, Modelica Association, 2015, Consortium, 2008], which are then composed. The physical correctness of this composition is only ensured, if the semantic models of each DSL are done correctly, and not because DSLs use the same syntactic form.

After each component in the motion stack[†] gets its coordinate or method specific representations, they are realized in software architecture while taking into account other imposed constraints. For example, let a constrained motion task be specified and controlled in an iTaSC framework. Then, according to the DSL-based approach, the whole application can be composed using a set of the DSLs such as

- a DSL for the coordination and the scheduling of the tasks - rFSM [Klotzbüecher, 2013].
- a DSL for the robot and the object structural models - KDL extensions with geometric relations semantics [Smits et al., 2001, De Laet and Bellens, 2012].
- a DSL for the constraint specification - expression trees [Aertbeliën and Schutter, 2014].

[†]Motion stack refers to a layered organization of the DSLs, software libraries, and architectures that are relevant to the motion specification.

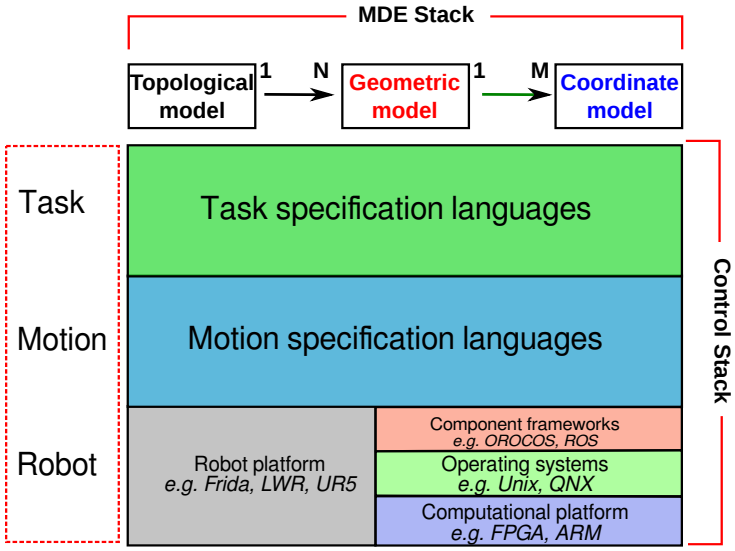


Figure 1.7: The diagram shows the orthogonal relationship between the task programming stack of DSLs and their models in MDE. Each layer in the stack has its own set of DSLs and tool-chains, which are realized using MDE methodology. This figure shows the separation of topological, geometric and coordinates models that are relevant for the realization of Motion and Robot layer DSLs.

- a DSL for the constraint solver - ACADO toolkit [Houska et al., 2009] or CasADi [Andersson, 2013] frameworks with qpOASES [Ferreau et al., 2014] solver.
- a DSL for the software components - OROCOS RTT deployment script [Soetens, 2006] and ROS launch and interface definition files [Quigley et al., 2009].

Almost all components in the list represent a combination of the DSLs and the tool-chains that can interpret them. But not all frameworks and components listed above have a clear separation of semantics and their implementation. This is especially the case for the models and implementations of the kinematic chains and computations.

1.4 Contributions

The thesis distinguishes two tracks of contributions:

Contributions to the modelling

- A definition of semantics for kinematic chain and computation modelling primitives. These semantic models build upon existing geometric relations semantics (Chapter 2).
- A modelling of task control applications using the MDE approach. The model decouples the specification problem from the specific representations of robot structures, geometric relations, solvers and controllers (Chapter 5).
- A determination of the variabilities in the whole-body control architecture (WBCA) and its realization through the MDE-based task programming approach (Chapter 5).

Contributions to the software implementation

- An implementation of the C++ templates-based DSL for the modelling and semantic checking of the kinematic chains and the associated computations (Chapter 2).
- An implementation, an analysis, and a validation of Popov-Vereshchagin's linear-time constrained hybrid dynamics solver using geometric relations semantics (Chapter 3).

- An implementation of the WBCA scheme for different application setups on KUKA youBot robot platform (Chapter 5).
- An implementation and an integration of a controller, which uses an MPC-like prediction algorithm to adapt its gains, with the linear-time constrained hybrid dynamics solver on youBot robot platform (Chapter 5).

1.5 Related Work

This section provides a top-level overview of the existing related work which has influenced and been taken into account in this PhD thesis. Later, each chapter performs a detailed analysis of the state of the art related to the material explained in that chapter.

1.5.1 Variability of Robot Models

Let's take the symbolic task representation given in Figure 1.2. Here, a set of motion primitives as a path, a path controller or a geometric constraint were identified. An application developer should have a necessary domain knowledge (robot control, constraint optimization, planning, dynamics, etc.) and be able *to ground* the symbolic representation onto the concrete piece of the executable software. There are a number of possibilities to implement each task primitive and to formalize its interactions with the other components in the task.

Let's take the *path* primitive, for instance. It has a geometric property. This geometry could be modelled as a set of straight lines with via-points or a spline. Further, each line may have its own representation. It could use Denavit-Hartenberg (DH) parameters or Plücker coordinates. Finally, for control purposes one might be interested in the poses of the points on the lines. The pose relations may have geometric models of their own. Each configuration of the model implies its own semantic and coordinate constraints. The similar procedure holds for the other components of the task, i.e., controllers, kinematic structures, etc. In software engineering, a family of such model configurations is often referred to as a *variability* of the model [Czarnecki and Eisenecker, 2000]. The identification and modelling of the variations of the models can become a daunting activity, because choosing the right level of modelling and model representations requires a detailed domain analysis and knowledge of experts.

There has been a lot of research in software engineering related to this topic. In [Kang et al., 1990], the authors introduce a feature oriented domain

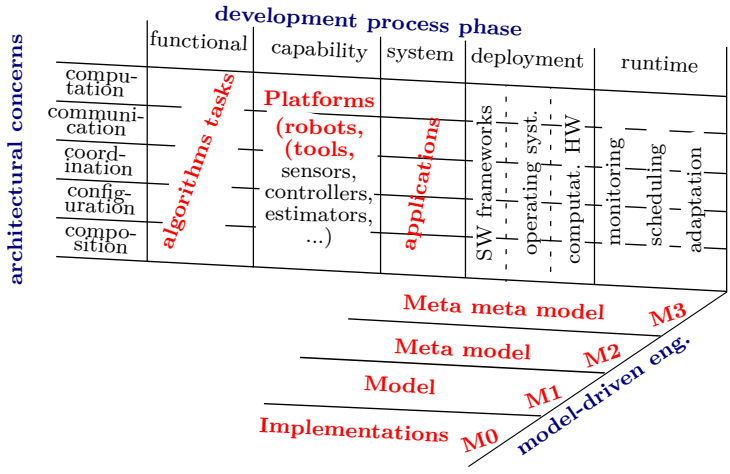


Figure 1.8: A variability space of a robot application is defined by three modelling dimensions: *development process phases* [Reiser, 2014], *architectural concerns* [Radestock and Eisenbach, 1996, Radestock and Eisenbach, 2003] and *model driven engineering layers* [Kent, 2002, Bruyninckx et al., 2013]. These dimensions also represent various modelling views or approaches on the same application during its life-time. For example during the deployment phase, a software architecture of some distributed application is commonly modelled or viewed as a composition of software components with specific roles and interaction options. The same application can be viewed in terms of the functionalities of each component and the programming languages they are realized in during the design phase.

analysis (FODA) methodology. FODA allows to reuse functionalities and architecture of the applications in the specific domain. It is achieved by creating their generic and widely applicable models, which are refined for the specific use cases. FODA relies on three modelling concepts: *aggregation/decomposition, generalization/specialization and parametrization.* Further, a number of variability modelling languages have been developed, in order to facilitate and automate the process of the variability modelling and its resolution for the specific applications in the domain. In [Berger et al., 2013, Chen et al., 2009], and [Eichelberger and Schmid, 2013], the authors provide a survey of features of such modelling languages and the methods they implement. In [Voelter et al., 2007] and [Voelter and Groher, 2007], the authors discuss how the model variabilities expressed in variability modelling languages is handled in model transformation and generation tools.

In robotics the concept of model variability has barely been expressed explicitly.

Some of the few examples are [Inglés-Romero et al., 2012, Gherardi, 2013], and [Hochgeschwender et al., 2013]. The latter two have been conducted in the context of the BRICS project [Bischoff et al., 2010]. In both efforts, the authors identify the variabilities in the deployment of the robot software architecture. These variations of the deployment model are taken into account in the transformation and code generation process. One of the general outcomes of the BRICS project with respect to the variability modelling is depicted in Figure 1.8. It depicts a relation between three modelling views on a robot application. Each of the views has its own models with their own constraints and variations. Here, of a particular interest is the dimension of MDE, because most implementations and methods in robot motion planning, specification, and control implicitly follow this layered organization [Hägele, 2011]. The models and approaches that are relevant in this thesis are marked in bold red color.

1.5.2 Software Composition and Physical Systems

A holy grail of software engineering is the process of software composition. Its main goal is to define and to implement software components that are used to build complex applications with deterministic behavior. The software composition is complementary to the topic of variability. The latter is needed to identify and explicitly model multiple configurations of the models and implementations involved in the composition. The authors in [Nierstrasz and Meijler, 1995] identify three major items to be related to software composition:

- *Languages* - address how to specify components and connectors [Mehta et al., 2000, Taylor et al., 2009]. They should be able to describe not only basic structural elements but also specific application architectures and generic architectures and component frameworks that those applications can comply with. A related work in this area is Architecture Description Languages (ADL) [Bass et al., 2003, Medvidovic and Taylor, 1997, Clements et al., 2002].
- *Tools* - address a possibility of requirement specifications, a software management in component frameworks and support for the application construction and maintenance.
- *Methods* - address technologies and methodologies that are useful to the development and maintenance of the component frameworks.

In [Lau and Wang, 2007, Lau et al., 2006], the authors analyze various forms of composition in component oriented programming (COP). They identify three concepts that are used to implement component software:

- *Semantics* - what they are meant to be, e.g., an object or an architectural unit.
- *Syntax* - how they are defined, constructed, and represented. For instance, the authors refer to component syntax as the syntax of component definition language, which is different from component implementation language. In 'object' components this definition language coincides with the implementation language, e.g in EJB it is Java. In a framework, the role of the definition language may coincide with that of an interface definition language which may provide mappings to different programming languages, e.g in CORBA it is an IDL language, in ICE it is Slice language.
- *Composition* - how they are composed or assembled.

The authors summarize that one needs at least three different language classes, i.e., an *implementation*, a *definition*, and a *composition*, to construct component-based software application. They also state that the composition can take place during the design, the deployment, and the run-time stages of the development.

In the software engineering community most of the research on composition is on structural aspects. That is, they focus on the types of component, port, and connector models and implementations. In embedded systems which includes also robotics, the situation is different. Here, the component behavior or coordination, aka *model of computation* (MOC), shift into the focus in addition to structural aspects [Eker et al., 2003a]. This domain researches how the components with different models of computation interact and are composed in a deterministic way to create a compositional system [Goessler and Sifakis, 2005, de Alfaro and Henzinger, 2001]. The research on composition for the embedded systems tries to solve a number of questions related to the development of software for discrete and continuous control systems. For instance, in relation to the control block diagram model depicted in Figure 1.9, the following research questions need to be addressed by the composition mechanism:

- What is the order of the initialization, start and execution of the software components that implement the blocks in this control diagram?
- Should the disturbances g and f on the controller C and plant P models be processed before their respective inputs, y_d and u?
- Should the sensor measurements y be estimated before or after the desired inputs y_d?

- How does an external decision block, e.g., planner, will interact with the components in the control loop?

Many of the above listed questions are modelled and solved based on *actor* paradigm to describe the composition mechanism [Lee et al., 2003, Eker et al., 2003b, Hewitt and Baker, 1978]. In an actor-oriented approach time and concurrency become key parts of the programming model. In [Lee, 2004, Lee, 2006], the author discusses a family of actor-oriented DSLs and tool-chains for the engineering systems. He discusses the composition mechanisms, type systems, and MOCs they support. He also discusses how actor-oriented DSLs are implemented using conventional programming languages. Some examples of such actor oriented composition languages that have recently been developed include Ptalon [Cataldo, 2006], Giotto [Matic, 2008], and CAL [Eker and Janneck, 2003].

Some of the language design decisions in Section 2 and the implementations of the control architectures of the example applications in Section 5 have been influenced by the state of art presented in this section.

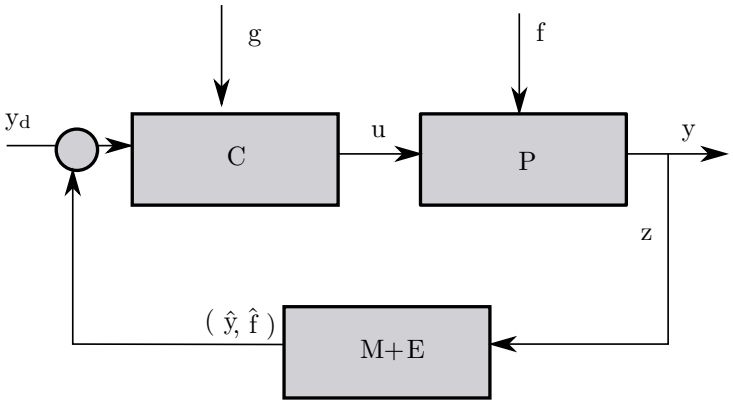


Figure 1.9: A control block diagram implementation of some task. Here the blocks and arrows represent models of a robot/plant P, a controller C, model update and estimation $M + E$, control input u, desired controlled variable y_d, measured controlled variable y, observed system state z, disturbances on the controller g , disturbances on the robot f, estimates of the controlled variables and disturbances $\hat{y}$ and $\hat{f}$, respectively.

1.5.3 Design and development of DSLs in robotics

A concept of *Domain Specific Languages* has been well known and researched for quite some time already in the software engineering and the programming language communities [Hudak, 1996, van Deursen et al., 2000, Bentley, 1986, Scott, 2009]. Recently, the introduction of the Model Driven Architecture (MDA) [Kleppe et al., 2003] and MDE [Kent, 2002] methodologies reinforced the attention for the DSLs again. Most of this research often focused on tool-chains and implementation methodologies of the DSLs. There is good amount of the published research that performs an exhaustive and comparative analysis of the features and shortcomings of these DSL tool-chains and the way they are implemented [Fowler, 2005, Merkle, 2010, Voelter and Groher, 2007, Ghosh, 2010, Parr, 2007].

As in general purpose programming languages, the DSLs can have their own syntax and semantics, but unlike the general languages the semantics of the DSLs are based on the abstractions of a concrete domain they are developed for. These domains can vary from the text processing to the modelling of complex dynamical systems. In general programming languages and their tool-chains, the knowledge of the operating system and the hardware platform defines what the domain knowledge is and based on it a machine-optimized specific code is generated. This is the case, regardless whether the language is compiled or interpreted. In the latter case there is another level of abstraction in the form of virtual machines that themselves are operating system and hardware specific. The same principle of the knowledge abstraction holds for the DSLs with the exception for the additional domain specific knowledge. This concrete knowledge of the specific models is added by the DSL compiler tools, which instantiate generic DSL programs in the form of the concrete programming language code. This process can be achieved in two ways: as an external DSL or an internal DSL [Fowler, 2010, Ghosh, 2010].

The **internal DSL** is built on top of the host programming language that the application is implemented in. The internal DSLs usually rely on already existing features of the host language and its tool-chain infrastructure. Therefore, one can consider the internal DSL to be designed as a library of generic language primitives. The DSL code can be then instantiated to a specific output by the host language's infrastructure [Ghosh, 2010]. Many internal DSLs use interpreted host languages such as Ruby, Lua, CLisp, etc. In [Ghosh, 2010], the author classifies the internal DSLs based on their implementation patterns into two main classes: *generative* and *embedded*. In the generative internal DSLs domain specific abstractions are transformed to generate concrete code through some form of metaprogramming, e.g., through macros, preprocessors and other metaprogramming mechanisms. In the embedded internal DSLs,

the domain abstractions that are part of the DSL type system are embedded within the type system of the host language. Here the author distinguishes a number of implementation patterns such as method chaining also known as fluent APIs [Fowler, 2010], typed embedding, reflective metaprogramming, etc. This classification is particularly interesting with respect to the design of the functional APIs and libraries. Since both domain specific functional libraries as well as embedded DSLs rely mainly on the type system of the host language to express the domain specific abstractions, the differences and advantages of one over the other become ambiguous.

Unlike the internal DSLs, the **external DSLs** are designed ground-up as stand-alone languages with their own tool-chain infrastructures. They require a lexical and a semantic analysis and a necessary lexer and a parser need to be implemented. A few years ago that would have been a big undertaking and one would have used such tools as flex/lex [Levine, 2009] and bison/yacc [Levine, 2009]. Recently, efforts to implement a programming language, the external DSL in particular, have been minimized with the help of the tool-chains known as Language Workbenches [Fowler, 2005, Merkle, 2010, Erdweg et al., 2013]. Some of the well known workbenches are Xtext [Foundation, 2015], IntelliJ MPS meta-programming system [JetBRAINS, 2015], ANTLR [Parr, 2007]. In [Ghosh, 2010] the author provides a classification of the external DSLs based on their implementation patterns. Here, he recognizes DSL workbenches, DSLs with embedded foreign code, DSLs based on context-based string manipulation and parser combinators. But regardless of the implementation pattern used, the principle always stems from the fact that the domain specific abstractions, i.e., domain's semantic model, are transformed into the concrete code. This multitude of the implementation methods gives flexibility to a DSL developer. That is, as long as one has a valid semantic model of the domain and an appropriate Abstract Syntax Tree (AST) that represents this model, the DSL of any syntactic flavor can be implemented.

In robotics, most of the DSL research was related to such sub-domains as planning, reasoning, coordination and perception. In [Nordmann et al., 2014] the authors provide a good overview of the DSL trend in robotics, they analyze existing resources on DSLs based on the DSLs' application domains. They identify eight robotics relevant sub-domains in which DSL developments have been going on: coordination, perception, planning, manipulation, etc. In [Dantam and Stilman, 2012, Dantam et al., 2010], the authors develop a Motion Grammar tool, which can be used to design and to analyze robot controllers using a formal language. The authors claim that the research results from the language and automata theory and can directly be applied on an application to prove its correctness and completeness. In [Ragan-Kelley et al., 2013], the authors develop a new programming language,

Halide, for image processing and computational photography. Another domain that is relevant to the robot task programming is optimization. There is a lot of research to express optimization-based problems using the DSLs. A good overview of the existing modelling languages and tool-chains in this domain is given in [Kallrath, 2004, Fragniere and Gondzio, 2002, Andersson, 2013].

1.5.4 Robot Control and Optimization

So far, the discussions focused on different modelling and implementation approaches for the software components and their architectural variations that influence the development of task control frameworks. Functionally, these software components and architectures implement some form of the (online) constrained optimization problem. In particular, Optimal Control Problem (OCP) [Betts, 2010] and Model Predictive Control (MPC) [Richalet et al., 1978] approaches that are based on the constrained optimization methods have recently been gaining popularity in robotics. Mathematically, OCP and MPC solve the equations as in

$$
\begin{aligned}
&\underset{x(\bullet),u(\bullet),T}{min} \int_{\mathbf{0}}^{\boldsymbol{T}} \boldsymbol{L}(\boldsymbol{x},\boldsymbol{u})\mathrm{d}\boldsymbol{t} + \bar{\boldsymbol{E}}(\boldsymbol{x}(\boldsymbol{T})) \\
&\qquad \text{subject to} \\
&\boldsymbol{x}(\mathbf{0}) - \boldsymbol{x_0} = \mathbf{0}, \quad \boldsymbol{r}(\boldsymbol{x}(\boldsymbol{T})) = \mathbf{0} \\
&\dot{\boldsymbol{x}}(\boldsymbol{t}) - \bar{\boldsymbol{f}}(\boldsymbol{x},\boldsymbol{u}) = \mathbf{0}, \quad t \in [0,T] \\
&\bar{\boldsymbol{h}}(\boldsymbol{x},\boldsymbol{u}) \geq \mathbf{0}, \quad t \in [0,T].
\end{aligned}
\tag{1.1}
$$

In the context of robot motion task control, the integral term is a cost function that represents some physical quantities that need to be optimized, $\boldsymbol{x}_0$ and $\boldsymbol{r}(\boldsymbol{x}(\boldsymbol{T}))$ are initial and terminal constraints (robot states), $\dot{\boldsymbol{x}}(\boldsymbol{t}) - \boldsymbol{f}(\boldsymbol{x},\boldsymbol{u}) = \mathbf{0}$ is the differential equation (DE) model of the robot motion, i.e., forward/inverse kinematic or dynamics, and $\boldsymbol{h}$ is often a geometric path or a trajectory constraint in joint or Cartesian spaces [Diehl et al., 2006].

Even though most optimization-based control approaches solve for the same formulation in Equation (1.1), they differ in the concrete numerical techniques they use. Furthermore, the choice of the numerical technique is influenced by the form of the cost function. The former also defines the type of the control problem, i.e., linear quadratic, time optimal, etc. In [Diehl et al., 2006,

Diehl, 2001, Andersson, 2013], the authors identify three groups of methods to solve optimal control problems:

- Dynamic programming (DP) [Bellman, 1957] - uses a principle of optimality to recursively compute the solution of the optimal control problem. Here, either Bellman's equation for discrete time models or Hamilton-Jacobi-Bellman's equation for continuous time models need to be solved. Their outcome defines a necessary (and sufficient for the global optimum) condition of the optimality. DP suffers from the curse of dimensionality, thus restricted to small search spaces.
- Indirect methods [Diehl, 2001] - formulate the optimal control problem as a boundary value problem (BVP) in ordinary differential equations (ODE). Then, these ODEs are solved using numerical integration methods. The indirect methods are also referred to as *first optimize and then discretize*. Common BVP problems that are relevant in robot control are initial value problem (IVP) and two-point boundary value problems. For example, forward dynamics control problem is an IVP problem that can be solved using recursive domain specific Newton-Euler method.
- Direct methods [Diehl, 2001, Bock and Plitt, 1984] - are the converse of the indirect methods. Because of this they are also referred to as *first discretize and then optimize* method. Here BVP is converted to a nonlinear programming problem (NLP). The NLP can then be solved using one of the root searching methods [Betts, 2010].

In robotics, in addition to the least square optimization (Jacobian pseudo inverse methods), summarized in Section 3.2, there has been a considerable amount of research on the optimal path control problems. [Verscheure, 2009] provides an excellent overview of the state of the art on optimization-based path planning and control. The research in [Verscheure, 2009] is further extended to task programming in [Decré et al., 2010], and [Debrouwere et al., 2013] introduces the sequential quadratic programming approach to solve optimal path control problem with acceleration and jerk constraints. [Martin and Bobrow, 1995] and [Sohl and Bobrow, 2001] apply optimal control to minimize actuator efforts for serial kinematic chain motions with end-effector constraints. In [Erez and Todorov, 2012], the authors extend this to trajectory optimization for domains with contacts.

In MPC, the model of the process is used to predict future process behavior, outputs and inputs w.r.t to some cost function [Diehl, 2001, Ferreau et al., 2008]. MPC can be viewed as the OCP being solved during some period of time $[\boldsymbol{t_0}, \boldsymbol{t_0} + \boldsymbol{T}]$ with $\boldsymbol{T}$ the prediction period. The outcome of the OCP problem,

i.e., optimal control inputs for that prediction period, is applied for some short time $\boldsymbol{\delta}$ to this system. Then at the interval $[\boldsymbol{t_0 + \delta, t_0 + \delta + T}]$ a new OCP is solved. This is repeated continuously until the terminal state is reached. Because of this forward moving nature of computations, MPC is also known as the *receding horizon control*. Most of the MPC realizations use direct simultaneous methods like multiple shooting or collocation. Some of the applications of the MPC in robotics are presented in [Erez et al., 2011, Erez et al., 2013], and [Van den Broeck et al., 2011].

In general, a software realization of one of the numerical techniques that is used to solve OCP is referred to as a *solver*. A review of the state of art shows that the solvers can be distinguished into two categories: those that use domain specific knowledge and are only applicable in that context and those that use mathematical optimization techniques and are applicable in any domain. The advantage of having a domain specific solver is that it can utilize the structural, the behavioral models and patterns in the domain. Example, Newton-Euler inverse dynamics solver applies Newton's second law recursively or iteratively to a kinematic chain. The recursion is defined by the topology of the kinematic chain. Such domain knowledge can allow on one hand, efficient implementations and on the other hand, the semantic validation of the task control problem. The latter is virtually non-existent for mathematical optimization solvers. In domain independent solvers the OCP problem is solved in batch using the numerical integration and differentiation techniques. This is unlike in the domain specific solvers, where each component of the control loop is usually treated separately. OCP for the motion tasks are defined either on the kinematic level, position and velocity-resolved schema, or on the dynamic level, acceleration-resolved scheme. This choice of the constraint resolution scheme determines, the differential equation model of the robot's motion. For instance, if the task control uses the velocity-resolved scheme, the motion model becomes

$$\boldsymbol{J_r(q)\dot{q} = \dot{X}}. \tag{1.2}$$

Here, $\boldsymbol{J_r(q)}$ is a robot Jacobian that defines a mapping between joint space velocities $\boldsymbol{\dot{q}}$ and Cartesian space velocities $\boldsymbol{\dot{X}}$. The type of the kinematic solver used in the task control loop and a respective desired input set-point constraint are defined depending on whether an inverse or a forward velocity kinematics problem is being solved. For instance, if the desired input set-point is the velocity twist of the end-effector, then it is the inverse problem and task control loop is built around inverse kinematics solver and computed joint space velocity vector is sent to the robot.

In case the task control uses the acceleration-resolved scheme and involves robot dynamics, the motion model takes the form of

$$\boldsymbol{M(q)\ddot{q} = \tau(q, \dot{q})}. \tag{1.3}$$

In this acceleration-resolved scheme $\boldsymbol{M(q)}$ is an inertia matrix that defines a mapping between joint space accelerations $\ddot{\boldsymbol{q}}$ and the forces $\boldsymbol{\tau(q, \dot{q})}$. The complete specification of the constrained motion using the expanded dynamic motion model of the robot is given as in

$$\begin{aligned} \boldsymbol{M(q)\ddot{q}} &= \underbrace{\boldsymbol{\tau_a(q) - \tau_c(q) - C(q, \dot{q})}}_{\boldsymbol{\tau(q,\dot{q})}}, \\ \boldsymbol{h(q)} &= \boldsymbol{0}, \end{aligned} \tag{1.4}$$

with $\boldsymbol{h(q) = 0}$, $\boldsymbol{\tau_c(q)}$, $\boldsymbol{\tau_a(q)}$, $\boldsymbol{C(q, \dot{q})}$ being the holonomic position constraints, constraint forces, input forces and bias forces, respectively. Note that the solution of the constrained system in Equation (1.4) is neither optimal nor stable (controlled) as OCP formulation implies. These components need to be addressed, accordingly.

Again, as in the case of the kinematics problem one can define the inverse and forward dynamics problems. In the inverse problem the desired input set-point constraint to the task control loop is joint space acceleration vector and the control loop is built around inverse dynamics solver and computed joint space torque vector is sent to the robot. For the forward problem the input is joint space torque vector.

Chapter 3 introduces a domain specific constrained hybrid dynamics solver that computes the solution of Equation (1.4) using domain specific computational sweeps on the kinematic chain of a robot. Additionally, the chapter will discuss where a constrained optimization solver fits in a bigger picture of the constrained robot motion task control and programming framework.

1.6 Overview of the Thesis

In addition to introductory material in Chapter 1 the rest of the thesis is organized as follows.

Chapter 2 introduces semantic models to describe robot kinematic chains and operations on geometric primitives associated with the chains. The semantics of the models is based on the semantics of the geometric relations for rigid bodies as defined in [De Laet et al., 2013b]. The models are formulated as the composition of the models of the geometric primitives and take into account constraints imposed by the articulated rigid bodies. The chapter also covers a design and an implementation of the DSL using the defined semantic models. It discusses the state of the art on DSL design approaches and how the concept of functional composition is used to implement composable semantic models.

It provides some examples on how the DSL is used to construct and validate kinematic chains and algorithms. The chapter concludes with discussions on future possible extensions of the DSL and how the approach can be applied to other components of the task control application, e.g., specification of controllers, constraints, etc.

Chapter 3 introduces the linear-time constrained hybrid dynamics algorithm. This algorithm was developed by Russian scientists in the 70's but was never got prominence outside Russia [Popov, 1974, Popov et al., 1978, Vereshchagin, 1989]. The chapter explains the details of the algorithm and discusses it in retrospect to other robot dynamics algorithms. The chapter also presents the extensions of the algorithm to task prioritization and constraint weighting. Furthermore, it discusses some numerical considerations that need to be accounted for during the computation of constraint forces.

Chapter 4 presents the results of the validation of the algorithm from Chapter 3 on a number of simulation setups with multiple existing robot kinematic chains, e.g., Baxter [Rethink Robotics, 2012], KUKA youBot[KUKA, 2010] and virtual planar mechanisms. The chapter also discusses the role of the solver in control applications and how a different set of constraints can be stabilized using existing control approaches. The simulations are conducted with a different set of constraints both in Cartesian space and joint space. The setups involved over-constrained constraint configurations with the conflicting Cartesian and joint space constraints, as well as non-conflicting configurations.

Chapter 5 discusses how the contributions from the previous chapters are integrated to implement various robot skills in the context of WBCA. The chapter discusses that most constrained task specification frameworks comply to the same layered programming model and implement WBCA. In WBCA, the degrees of freedom available to the system are allocated to resolve the constraints of the task. The constraints are imposed in any space and need to be controlled in that space. The chapter presents a number of application setups that use different kinematic chains, e.g., a fixed serial robot or a mobile manipulator, and constraint controllers. The purpose of these setups is to show that as long as variabilities of the control architecture model are identified, any combination of constraints, controllers and solvers can be deployed. One of the setups uses a combination of the linear-time constrained hybrid dynamics solver with a receding horizon gain prediction algorithm. The algorithm does not generate optimal control inputs to the solver but modifies controller gains according to some objective function. The chapter analyzes and shows how such an architecture is implemented and deployed on a software architecture using the OROCOS RTT framework.

Chapter 6 summarizes the contributions and the results of the research

presented in the thesis. It discusses what further extensions and developments are required to improve robot task control and programming.

Chapter 2

Semantic Modelling for Composable DSLs *

2.1 Introduction

In Chapter 1 it was shown that a task specification has commonly four constituents: objects (including robots), constraints on object relations, controlled actions and a schedule. Each of them can be implemented as a DSL, in order to facilitate the task specification problem. These DSLs should comply with the semantic models, as in Definition 3, of each task component, thus making them invariant to a concrete (coordinate) representation used in an application. The invariance property of task primitives also fosters model and implementation re-use across different applications and task control frameworks, thus, making it a necessary condition for the re-usability.

Definition 5. *A semantic model of a geometric relation is a set of minimum geometric primitives, i.e., point, frame, body, their operations, and the geometric constraints they need to satisfy, that unambiguously define this relation [De Laet et al., 2013b].*

This chapter develops the semantic model and a compliant DSL to describe robot kinematic structures and robot motion models, i.e., kinematics and dynamics algorithms (Fig. 2.1). The semantic model and its implementation allow one to specify and validate a construction of robot kinematic structures and motion

*The content of this chapter is based on the research presented in [Shakhimardanov and Bruyninckx, 2015]

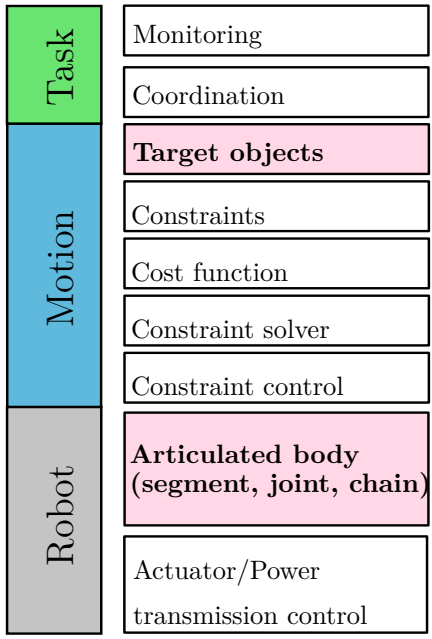


Figure 2.1: Target object and robot specification DSLs and their position in the task programming stack.

algorithms. The primitives of the models and the DSL are designed in such a way that their compositions define other complex primitives. These semantic models are based on coordinate invariant geometric relation semantics presented by De Laet et al. in [De Laet et al., 2013b] and defined as in Definition 5.

Before diving into the design and the development of the models and their computer language implementation, A concrete example motivates the need for the representation invariant models and programming tool-chains. The example is based on the tutorials by Roy Featherstone presented in [Featherstone, 2010a, Featherstone, 2010b].

2.2 Example

Figure 2.2 depicts an articulated system consisting of two rigid bodies. Both Figure 2.2(a) and Figure 2.2(b) show the same system with both bodies initially

at rest. An effort is applied only on the left body B_1. This effort contributes to the motions of both bodies. In Figure (a) the effort on the B_1 and its resultant motions use two 3D vectors, i.e., a net linear force $\boldsymbol{f}$ and a net moment $\boldsymbol{n}$ for the effort and a linear acceleration $\boldsymbol{a}_i$ and an angular acceleration $\dot{\boldsymbol{\omega}}_i$ for the motions. In Figure (b) the net effort on B_1 is the single 6D spatial force vector $\boldsymbol{F}$ and the resultant motion is the single 6D spatial acceleration $\ddot{\boldsymbol{X}}$. The goal is to express each body's motion in terms of the net effort on B_1, the masses and the inertias of the bodies.

The summary of some of the resultant equations is given in Table 2.1. In particular, one is to note the number of the equations and the constraints each representation prescribes. In 3D vector representation, each quantity should explicitly indicate the coordinate frame it is expressed in, before performing any computations. It also requires careful analysis of the lines of action of the linear forces to identify the components that contribute to the motions.

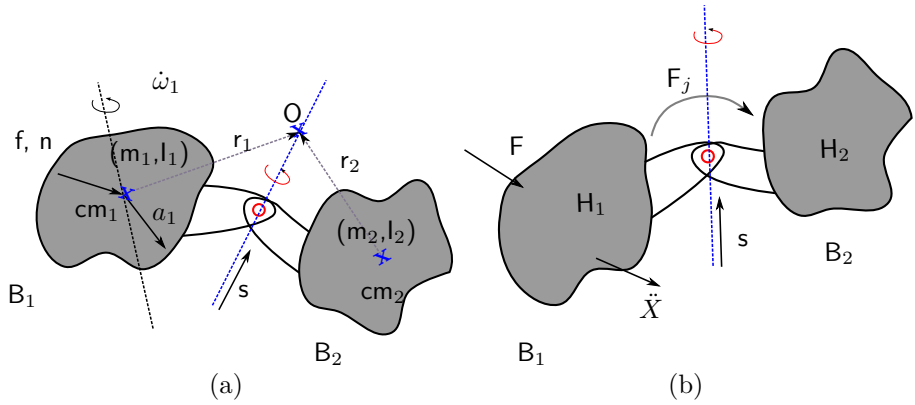


Figure 2.2: The motion of two-body articulated system can be described using 3D vectors for the linear and angular motions and efforts as in (a) or using 6D spatial vectors as in (b). Even though these two representations describe the same physical motion, the constraints they impose are different. This leads to two different ways of computing the same quantity. In (a) cm_i, m_i, I_i, a_i, $\dot{\omega}_i$, f_i, n_i, r_i, s, O are a center of mass, a mass, a rotational inertia about the center of mass, a linear acceleration of the center of mass point, an angular acceleration of the body, a linear force with the line of action passing through the center of mass point, a moment on each body B_i at the center of mass, a direction of the axis of rotation and a point through which the axis passes, respectively. In (b) F, $\ddot{X}$, H_i, F_j are a spatial force, a spatial acceleration, a spatial inertia of one of the bodies and the portion of the spatial force transmitted over the joint connecting the bodies, respectively.

3D vector formulation	6D vector formulation
${}^{O}a_1 = {}^{cm_1}a_1 - r_1 \times {}^{cm_1}\dot{\omega}_1$	$F - F_j = H_1 \cdot \ddot{X}_1$
${}^{O}a_2 = {}^{cm_1}a_2 - r_2 \times {}^{cm_2}\dot{\omega}_2$	$F_j = H_2 \cdot \ddot{X}_2$
${}^{cm_1}f_2 = {}^{O}f_2 = {}^{cm_1}f_2$, ${}^{cm_1}f_1 = f - {}^{cm_1}f_2$	$\ddot{X}_2 = \ddot{X}_1 + s\alpha$
${}^{cm_1}n_2 = {}^{cm_2}n_2 + (r_1 - r_2) \times {}^{cm_2}f_2$	$s^T(F - F_j) = 0$
${}^{O}n_2 = {}^{cm_2}n_2 - r_2 \times {}^{cm_2}f_2$	
${}^{cm_1}n_1 = n - {}^{cm_1}n_2$	
$s^T \cdot {}^{O}n_2 = 0$	

Table 2.1: Summary of the motion and effort vector relations using two 3D vectors and one 6D vector representations [Featherstone, 2010a]. Here, the last equations in each column are the constraints prescribed by the joint coupling. Superscripts O, cm_1, cm_2 denote the origin points of the coordinate frames in which the quantities are expressed in. Action lines of the linear forces on each body pass through their centers of mass.

The component-wise, for the linear and the angular components separately, Newton-Euler formulation increases the number of involved equations. In 6D vector formulation neither of the above is the case. Furthermore, spatial (one 6D vector) and conventional (two 3D vectors) acceleration describe the same physical phenomenon, but in different coordinate systems! The former represents the change in the velocity vector of the vector field at a point that coincides with the reference point which is fixed in space [Featherstone, 2001].

The example shows the complexity and the semantic differences involved, when using different representations for the same phenomenon even in a simple mechanism. This complexity increases manifold with the number of degrees of the robot. This may inevitably lead to an incorrect result, or worse an invalid computation, if not done carefully. If this happens it is very difficult to track such bugs, because of their snowball effect on the whole system.

In order to resolve this problem, one of the aspects to consider is to include explicit semantics to all the primitives and operations involved in the specification process. It can be achieved by identifying and decoupling general semantics of the coordinate specific geometric models and by encoding the semantics in DSL. This DSL can then be used to instantiate the same robot structural and computational models using concrete coordinate representations. Additionally, such an approach ensures correct-by-construction implementations and helps to avoid code duplication when implementing new algorithms.

```
1  //vector v_base_AB is expressed in the same base as the twist
2  //vector v_base_AB is a vector from the old to the new point
3  joint.twist.RefPoint(const Vector& v_base_AB) = Twist(vel+rot*
       v_base_AB,rot)
4  segment.twist = joint.twist(qdot).RefPoint(joint.pose(q).M *
       f_tip.p)
```

Listing 2.1: A common form of representation in conventional kinematics and dynamics libraries. It requires a lot of effort to check semantic validity and the results of such expressions [Smits et al., 2001].

Contributions. The goal of this chapter is to show how the semantic models can be used to describe kinematic chain structures and their operations. Specifically, the chapter develops **(i)** semantic models and constraints to specify: *(a) structural primitives* (Section 2.4), e.g., a segment, a joint, a kinematic chain and *(b) operational primitives* (Section 2.4) that work on the structural semantic models above, e.g., pose, twist or wrench operations on the segments; **(ii)** a C++ library that can be used to *construct* and *check* semantic validity of the kinematic structures and the algorithms (Section 2.6). For instance, it can be used to construct, compute and check inverse dynamics algorithm for a tree structured robot with a set of named tool-tips.

2.3 Related work

Despite being used in robotics for the past several decades, with the exception of the recent research by De Laet et al. [De Laet et al., 2013b] and to some extent by Featherstone [Featherstone, 2008], there have been few efforts to standardize the semantics of the geometric primitives, the kinematic models and computations they are used in. As a result, there have been a proliferation of various kinematics and dynamics libraries [Rickert, 2015, Smits et al., 2001, Felis, 2011], which rely on different sets of representations and implicit constraints. This rendered them incompatible, as well as tedious to debug. This problem can be well observed in Listing 2.1, which encodes a lot of such implicit knowledge. The listing shows a computation of the segment twist from the joint twist in a kinematic chain. The issue here is that there is no explicit information on the points at which the twist is measured or on the coordinates in which the measured twists are expressed. This knowledge should be deduced based on the limited code documentation or even worse by implementing a test application and analyzing its numerical output. This state of matters is common to many existing libraries and frameworks. The community has implicitly been aware of such problems and realized that they can be addressed by decoupling general domain semantics from the concrete implementations. Recently, it led to

the proliferation of various DSL and tool-chains. But before the DSLs became popular, this problem was usually addressed by developing class hierarchies using public inheritance (inherit functionalities or interfaces which are exposed for users of the class). In this approach, the most generic(base) class is chosen as the root parent of the hierarchy and represents the general semantics of the domain [Shakhimardanov et al., 2010]. The children of the base class would then specialize concrete situations with additional constraints. But many robotics software implementations *misused* the inheritance-based approaches to create unmanageable hierarchies of classes [Shakhimardanov et al., 2010]. The utilization of the DSLs does not guarantee that the similar problems can be avoided, on the contrary the misuse of DSLs can lead to the problem of DSL cacophony [Fowler, 2010]. In this case, many DSLs are created to model every tiny bit of the problem, which is not necessarily a productive approach and very similar to the issue of the unmanageable inheritance hierarchies.

The advantage of the DSLs and tool-chains over that of the public inheritance[†]-based approach is that the concrete domain knowledge is modeled as the specific transformations from the generic semantic model into the specific instances of that model. These transformations take into account all the constraints of the specific models, are validated and are not meant to be meddled with.

Few resources are available with respect to semantic models and DSLs in the context of rigid body applications in robotics. Most content in this context is developed in computer graphics and simulation applications. In these domains, most of the DSLs are based on XML language and are limited to the validation approach prescribed by XML infrastructure, e.g., DTD, XML Schema-based validation. Some of the examples include AutomationML [AutomationML, 2008], COLLADA [Barnes and Finch, 2008], X3D [Web3D Consortium, 2008] and SRDF [Kunze et al., 2011], URDF [Willow Garage, 2009] in robotics.

In other sources the models for relations of the geometric primitives in the form of scene graphs [Strauss and Carey, 1992, Foote et al., 2011a, Foote et al., 2011b] have been widely discussed. But their focus did not lie on the semantics and constraints of connected nodes. [Featherstone, 2008] provides a guideline on modeling rigid-body systems. It describes the construction process of complex kinematic structures and algorithms starting with the geometric model of the primitives, e.g., a joint and a link. It also introduces multiple frames to describe the locations of the joints on each body. These are analogues to the concept of joint attachment pose frames presented in this chapter with the exception for not being formalized in terms of the geometric primitives of a point, a body and a frame. In [Featherstone, 2010a] and [Featherstone, 2006], the author explains differences between two 3D Euclidean and 6D screw or

[†]This is the form of the inheritance usually used in robotics software.

spatial vector representations of the motions (as twist screws) and the efforts (as wrench screws). He propounds that 6D spatial vectors convey complete geometric information of the rigid body motions and efforts in coordinate-free form and states that there is no need for a reference point to define them. A similar discussion with the focus on the acceleration twists is presented in [Featherstone, 2001]. On the other hand, he uses Plücker coordinates to ground the ideas, which refutes the statement on the coordinate-free semantics of the motions and efforts.

In [Frigerio et al., 2011, Frigerio et al., 2012] Frigerio et al. define a DSL to describe the kinematic structures. They express the semantic model that the language uses in the form of a UML class diagram. Though their method is promising, it also has some shortcomings: first, the limited expressiveness of the UML which can not convey semantic constraints in the model completely; second, the model binds itself to implicit semantic constraints such as the number of motion DoFs of the joints, the co-location of joint reference and target frame origins to name some.

The content developed in this chapter makes use of the existing work by De Laet et al. presented in [De Laet et al., 2013b, De Laet et al., 2013a]. The research provides complete semantics of the geometric primitives relations, including pose, twist and wrench, and their coordinate semantics. They expose semantic constraints imposed by the specific coordinate representations used with these relations, as well as common errors made during the specification process. In [De Laet et al., 2013c], the approach is extended to the specification of rigid body pose and twist scene graphs. This enables one to query various geometric quantities associated with rigid bodies without relying on the specific coordinate representation and resolve the necessary semantic constraints. Table 2.2 summarizes how the different semantics and coordinates contribute to the complete specification of the geometric relations. Based on the information provided, one can conclude that the concrete and the complete specification of the geometric relation is obtained by composing its coordinate invariant semantics and its context specific coordinate representation, i.e., numerical aspect.

Notation. In the following discussions on the models, most of the examples will be using geometric semantics notation given in Table 2.2. Here, lower case letters f, l represent point names, upper case letters in square brackets $[F]$, $[L]$ represent orientation frame names, lower case letters in curly braces $\{f\}$, $\{l\}$ represent **pose frame** names (frame origin point and orientation frame), upper case letters in boldface $\mathbf{A}$, $\mathbf{C}$ represent body names. In order not to overload the figures, sometimes only one of the above is used. Also, in order to simplify

Geometric relation semantics	Geometric relation coordinate semantics	Composition of semantics and coordinates	Geometric primitives
PoseSem$((l,[A])\|\mathbf{C},(f,[B])\|\mathbf{D})$	PoseCoordSem(PoseSem, $[r]$)	Pose(PoseCoordSem, Coordinate)	point l orientation frame A body C
PoseSem$(\{l\}\|\mathbf{C},\{f\}\|\mathbf{D})$	PoseCoordSem(PoseSem, $[r]$)	Pose(PoseCoordSem, Coordinate)	point f orientation frame B body D coordinate frame r
TwistSem$(l\|\mathbf{C},\mathbf{D})$	TwistCoordSem(TwistSem, $[r]$)	Twist(TwistCoordSem, Coordinate)	point l body C body D coordinate frame r
WrenchSem$(l\|\mathbf{C},\mathbf{D})$	WrenchCoordSem(WrenchSem, $[r]$)	Wrench(WrenchCoordSem, Coordinate)	point l body C body D coordinate frame r

Table 2.2: Geometric relations semantics are defined in terms of a minimal number of geometric primitives. The coordinate semantics is defined by expressing the geometric relation in a specific coordinate frame. The complete specification of the geometric relation is given by the composition of its coordinate semantics and a specific coordinate representation. For the complete list refer to [De Laet et al., 2013b]

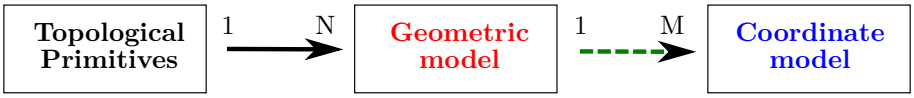


Figure 2.3: There is a one-to-many relationship between a physical primitive and its representation using a specific geometric model and in a specific coordinate representation.

the written form of the geometric expressions of Pose, Twist, Acceleration Twist and Wrench, the symbols X, $\dot{X}$, $\ddot{X}$ and F are used whenever the semantics is involved and their boldface versions $\boldsymbol{X}$, $\dot{\boldsymbol{X}}$, $\ddot{\boldsymbol{X}}$ and $\boldsymbol{F}$ are used whenever both the semantics and coordinates are represented in the expressions.

2.4 Semantic models for the kinematic structures

In order to define the specification semantics for the domain, first, one is required to identify the smallest conceptual primitives of the domain that are physically valid, and then formalize their meaning and possible semantic constraints. It allows to formally reason on the models and to check whether all semantic constraints of the domain are satisfied in a concrete combination of the primitives, as well as, the geometric and the coordinate models that are chosen for an application.

The Figure 2.3 shows a simplified relationship between different modeling classes (representations). The models discussed in this chapter build upon the geometric and the coordinate models that have been presented in [De Laet et al., 2013b]. Therefore, the main effort in defining the primitives for the kinematic chain models is to establish relationships among these three model classes. This mapping process is not achieved by a one-to-one relation among the primitives. It requires an effort into the complete formal specification of the constraints that need to be resolved across the chosen models. This is partially depicted in the Figure 2.3 as a one-to-many relationship. A more elaborate example of this matter is given in the Figure 2.4. Here, the structural primitive, `Segment` is defined in two ways, both of which are semantically valid. On the top, the pose frame of the proximal end is defined with respect to the pose frame of the distal end, which is then defined with respect to some reference body and a reference pose frame attached to it. Whereas, in the bottom, the pose frames of the proximal end and the distal end of the link body are defined with respect to the same reference body and a pose frame attached to it. Also, geometrically the pose frames can be expressed either as a position and an orientation or as a pose. The view of the coordinate models gives a rough idea on how each

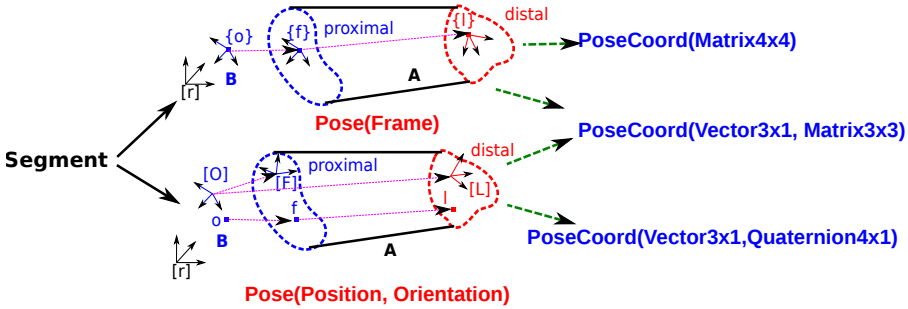


Figure 2.4: There is a one-to-many relationship between a physical primitive (both a structural and a operational) and its representation using a specific geometric model and in a specific coordinate representation.

of the mappings can be configured using a specific coordinate representation. Each configuration has its own constraints, the valid resolution of which may depend on the concrete application context. In general, a complete semantics of the modeling primitive is defined through its *structural* representation (i.e., data) and *operations* that work with/on this structure. In order to create a semantically valid model, both of the aspects will have the constraints that need to be satisfied. Here, it is necessary to emphasize that the semantics of the kinematics (and dynamics) modeling primitives are *not* new definitions, but are built by composing the existing semantics in geometric and coordinate models, but with *additional constraints*. Therefore, the full specification requires composition rules to create complex model primitives from the combination of the structural representations and their operations. The following sections introduce the details of the semantics and constraints associated with the modeling primitives.

2.4.1 Models of the structural primitives

While defining the basic structural primitives to describe kinematic mechanisms, one can distinguish between two types of properties they may possess: geometric or kinematic and dynamic properties. The former includes geometric and kinematic relations between points, frames and bodies, whereas the latter address physical properties as a mass, an inertia, an elasticity, a damping and so on of the body. For instance, in the context of a kinematic chain a segment is defined as a geometric constraint between two pose frames. On another hand, an ideal rigid body model (in this text an equivalent term is link), which is a part of the segment has a mass, an inertia and a shape. The following paragraphs describe each primitive and give examples on how to use them.

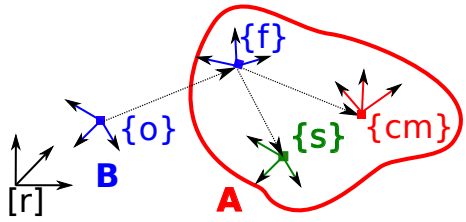


Figure 2.5: From a geometrical point of view, a minimal semantics of a link is given as an ideal rigid body with any fixed pose frame, $\{f\}$ on it. In case the dynamics and shape geometry are required, additional two pose frames are defined, $\{cm\}$ at which the center of mass and moment of inertia are defined and $\{s\}$ where the shape is attached. In the nominal case the inertia attachment pose frame is assumed to coincide with $\{cm\}$. In a general case, the inertia can be defined in any other pose frame.

Link is an ideal rigid body, Figure 2.5. Geometrically, it is defined as the body with an attached pose frame, $\{f\}$. This is a minimal semantics of the link and can be used in the contexts where only the knowledge on the geometric relations is required, e.g., in kinematics computations. For the contexts, where also dynamic and shape properties are required the knowledge on the mass, inertia and shape of the link is added. In order to define these, the link requires two additional pose frames: a shape attachment pose frame, $\{s\}$ and $\{cm\}$ pose frame where the center of the mass and the moment of inertia, I_{cm} are defined[‡]. Furthermore, if the moment of inertia of the body is measured at any other pose frame with the reference point at its origin, then that pose frame is an inertia attachment pose frame. In order to uniquely distinguish among multiple pose frames on the link, each pose frame is given an *attachment semantics*. In addition to the inertia and shape attachment semantics, there are pose frames with the feature attachment semantics that are used to specify locations of sensors, tool-tips or features relevant to the task computations. The attachment semantics is a semantic tag added to the pose description.

$$\text{AttachmentSemantics} \in \{\text{Type::Inertia, Type::Shape, Type::Feature}\}$$

$$\begin{aligned}\text{AttachmentPoseFrame} &\equiv (\text{Pose, AttachmentSemantics}) \\ &\in \{\text{Pose}_{\text{feature}}, \text{Pose}_{\text{inertia}}, \text{Pose}_{\text{shape}}\}\end{aligned}$$

[‡]The semantics of the inertia is defined in Section 2.4.2.

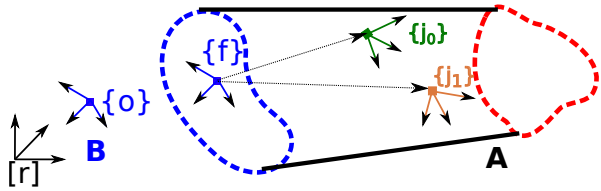


Figure 2.6: Segment is a Link with additional attachment pose frames for joints (frames $\{\mathbf{j}_0\}$, $\{\mathbf{j}_1\}$). Here $\{\mathbf{f}\}$ is the feature pose frame that is available through the specification of the link.

$$\text{Link} \equiv \Big(X_{\text{feature}}(\{f\}|\mathbf{A}, \{o\}|\mathbf{B}), X_{\text{inertia}}(\{cm\}|\mathbf{A}, \{f\}|\mathbf{A}),$$

$$X_{\text{shape}}(\{s\}|\mathbf{A}, \{f\}|\mathbf{A}), \text{mass}, \text{moment of inertia}\Big)$$

Constraints: ***(i)*** every attachment pose frame should have attachment semantics and be uniquely identifiable, ***(ii)*** the link can have one shape and one center of mass attachment pose frames and more than one feature attachment pose frames, ***(iii)*** one body fixed feature pose frame is chosen as a reference and all the other attachment pose frames are defined with respect to this pose frame, $\{f\}$ in Figure 2.5.

Segment is a link with additional attachment pose frames in the context of the kinematic chain construction, Figure 2.6. There is one additional type of attachment semantics: *joint attachment*. The pose frames that are used to specify relative constrained motion between two segments in the kinematic chain have joint attachment semantics.

$$\text{AttachmentSemantics} \in \{\text{Type::Joint}, \text{Type::Feature}\}$$

$$\text{AttachmentPoseFrame} \equiv (\text{Pose}, \text{AttachmentSemantics})$$

$$\in \{\text{Pose}_{\text{feature}}, \text{Pose}_{\text{joint}}\}$$

$$\text{Segment} \equiv \Big(\text{Link}, X_{\text{joint}}(\{j_0\}|\mathbf{A}, \{f\}|\mathbf{A}), X_{\text{joint}}(\{j_1\}|\mathbf{A}, \{f\}|\mathbf{A})\Big)$$

From the separately defined attachment semantics in the context of the link, only the feature attachment semantics is accessible to the segment, i.e., one cannot define the pose frames with the inertia or shape semantics in the context of the segment. These are available to the segment through the definition of

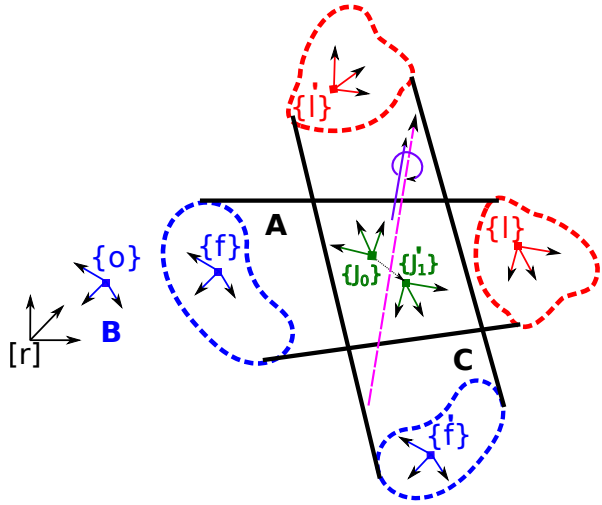


Figure 2.7: Joint is a kinematic constraint relation between two unique joint attachment pose frames of the segment. There can be more than one joint attachment pose frames on the segment. In case of the 1-DoF joint, the relative motion of the joint frames takes place about a directed line.

the link. If the link does not have any dynamic or shape properties, then the segment will also have only the geometric semantics. The addition of further feature pose frames for the sensors and tasks is possible.

Constraints: ***(i)*** the link is the part of the segment specification and its constraints are also valid in this context, ***(ii)*** all the joint pose frames are defined with respect to the reference feature pose frame defined on the link, $\{f\}$ in Figure 2.6, ***(iii)*** the pose relations between the reference feature pose frame and the joint attachment pose frames are constant, ***(iv)*** the segment can have more than one joint pose frames.

Joint is a kinematic primitive, Figure 2.7. It represents a relative *constrained* motion of two segments. In particular, the motion takes place between two pose frames with joint attachment semantics, which can be anywhere on the segments. The motion of the joint is constrained in the subspace $\hat{S}_M$ of 6-DoF motion space $\bar{M}^6$, $\hat{S}_M \subseteq \bar{M}^6$. The subspace $\hat{S}_M$ can be represented as a range of $6 \times n_f$ matrix $\boldsymbol{S}$ with $dim(\hat{S}_M) = n_f$ being number of free motion degrees, ranging from 1-DoF for prismatic and rotational joints to a 6-DoF for a free floating joint [Featherstone, 2008]. In the special case of the 1-DoF rotational and prismatic joints, $\boldsymbol{S}$ is 6×1 column vector with all the row elements except for one being zero and the motion takes place about a *directed line*. Depending

on the line representation used, the joint attachment pose frames need to satisfy a set of constraints imposed by these representations. For instance, a minimal Denavit-Hartenberg line representation (uses four parameters), when adapted to the frame notation, implies that the Z-axis of the frame can be freely chosen, but not the position of the frame's origin along the line, nor the orientation of the frame about the line [Denavit and Hartenberg, 1955]. Such minimal representations will always have singularity issues.

The joint also has a set of *properties* that help to further refine the relative constrained motion between the segments. The coordinate representation invariant geometric property is a *polarity*. The polarity of the joint specifies which of the two involved joint pose frames is a *reference* pose frame for the constrained motion. When the reference is set, the remaining pose frame becomes the *target*. The coordinate specific geometric properties include *direction of the motion*, *limits on the motion* and an *offset distance*.

$$\text{Joint} \equiv \Big(X_{\text{joint}}(\{j_0\}|\mathbf{A}, \{f\}|\mathbf{A}), X_{\text{joint}}(\{j_1'\}|\mathbf{C}, \{f'\}|\mathbf{C}), \hat{S}_M, \text{polarity} \Big)$$

$$\hat{S}_M \subseteq \bar{M}^6, \quad \text{polarity} \in \{\{j_0\}|\mathbf{A}, \{j_1'\}|\mathbf{C}\}$$

Constraints: ***(i)*** the joint can only be specified between two joint attachment pose frames, ***(ii)*** the joint attachment pose frames cannot be on the same segment, ***(iii)*** for the joint using a concrete coordinate representation, $\boldsymbol{S}_{6\times n_f}$ is defined in the coordinates of the target joint attachment pose frame.

Transmission represents all dynamic properties associated with the joint. Semantically, the joint is a kinematic primitive constraining the relative motion of the bodies and does not have dynamic properties such as *backlash*, *friction* or *impedance*, determined by the *inertia*, *stiffness* and *damping*. These properties are part of the transmission system that connects the joint and the motor that drives it [Albu-Schäffer et al., 2007b]. These properties are not considered in this research, except for the simple case used in the computation of the dynamics algorithm presented in Chapter 3.

Kinematic chain[§] is defined as an ordered set of joints, $\mathbf{L}_{\text{joints}}$, where the first element in the set is a *base/root* of the chain, Figure 2.8. Physically, any given set of joints can form a kinematic chain with multiple loops. In this research focuses on the kinematic chains with a spanning tree topology. The difference between the latter and the former is determined by the constraints on the

[§]The term kinematic chain does not imply that segments and joints are serially connected.

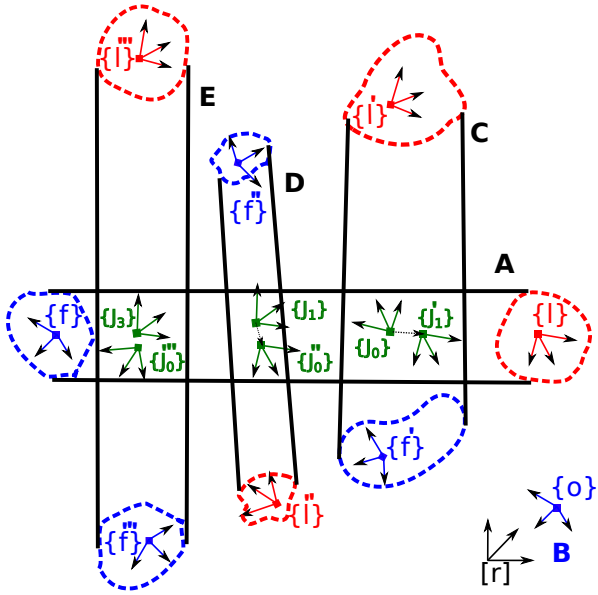


Figure 2.8: A KinematicChain is a kinematic primitive and defined as a collection of joints with additional constraints on this collection.

connectivity of the joint attachment pose frames and bodies.

$$\text{KinematicChain} \equiv \mathbf{L}_{\text{joints}}, \quad \text{where } \mathbf{L}_{\text{joints}} \subseteq \forall_{\text{joints}}$$

The constraints are also the cause that not every ordered set of the joints creates a semantically valid kinematic chain. That is, *the existence of a list of valid joints is a necessary, but not a sufficient condition.* The next paragraph describes a number of constraints that need to be satisfied in order to construct the semantically valid kinematic chain with the spanning tree topology from the existing set of the semantically valid joints.

Constraints: ***(i)*** a joint attachment pose frame on the segment that is used to define a joint connection cannot be reused to define another joint connection. That is, a single unique joint requires a single unique joint attachment on each segment it connects. This ensures that multiple joints are not accidentally co-located; ***(ii)*** if the joint has been set as the root joint of the kinematic chain, its reference joint attachment pose frame cannot be used as a target joint attachment pose frame in any other joint and its reference body becomes the base/root body of the chain; ***(iii)*** after the root joint of the kinematic tree has been chosen, the polarity property of all joints in the chain should hold.

Example, in Figure 2.8, let

$$\text{Joint}_1 \equiv \Big(X_{joint}(\{j_1^{'}\}|\mathbf{C}, \{f'\}|\mathbf{C}), X_{joint}(\{j_0\}|\mathbf{A}, \{f\}|\mathbf{A}), \hat{S}_{M1}, \overbrace{\{j_0\}|\mathbf{A}}^{\text{polarity}}\Big).$$

and be set as the root joint of the kinematic chain. Then

$$\text{Joint}_2 \equiv \Big(X_{joint}(\{j_0^{'''}\}|\mathbf{E}, \{f^{'''}\}|\mathbf{E}), X_{joint}(\{j_3\}|\mathbf{A}, \{f\}|\mathbf{A}), \hat{S}_{M2}, \overbrace{\{j_0^{'''}\}|\mathbf{E}}^{\text{polarity}}\Big).$$

cannot be part of this kinematic chain, because the constraint ***(ii)*** is not satisfied.

It is important to note that the structural model primitives do **not** carry completely **new** semantics, but rather are built upon semantics of the simpler primitives by composition of their semantics and extension of additional constraints. This also makes them self-contained. For instance, in order to specify a kinematic chain one requires only a list of joints, which already contain necessary data to construct the chain. But such an approach is only possible if all the constraints and their validity *contexts* are identified and made explicit. It should be achieved across all levels of modeling, starting from the specification of rigid body's pose semantics and coordinate representation and ending with the addition of the joints into the existing kinematic chain. The difficulty of the problem lies in identifying the scope of the context in which a set of the constraints can become (in)valid. Table 2.3 summarizes the semantic constraints on the structural models.

2.4.2 Models of the operational primitives

The goal of the operational primitives for the kinematics and dynamics is not to manipulate the structural models from Section 2.4.1 per se, but rather to operate on the *physical* (geometric, kinematic, dynamic) relations that are associated with them. As with the structural models, the semantics of and the constraints on the operations also build upon those associated with the (geometric) relations on points, frames and bodies. Table 2.4 shows a summary of the relevant operations. These operations are also valid with respect to the poses, twists and wrenches of the structural primitives of kinematic mechanisms. For instance, in Figure 2.13 one could query for the pose of the Segment_3 measured in terms of the pose frame $\{l_3\}$ with respect to the robot's base pose frame $\{f_0\}$ through the composition of the other intermediate pose relations. In another example also in Figure 2.13, one could query for the twist of the segment Segment_3 measured at the point l_3 and expressed in the coordinate frame $[L3]$ through the composition of all twists from the supporting joints.

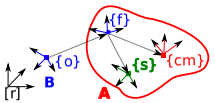
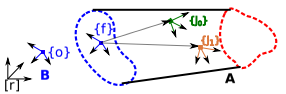
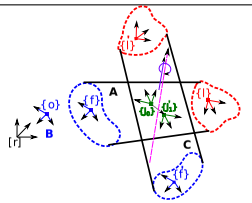
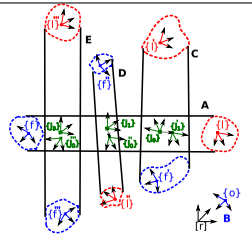
Name	Graphical depiction	Constraints
Link		***(i)*** every attachment pose frame should have attachment semantics and be uniquely identifiable, ***(ii)*** the link can have one shape and one center of mass attachment pose frames and more than one feature attachment pose frames, ***(iii)*** one body fixed feature pose frame is chosen as a reference and all the other attachment pose frames are defined with respect to this pose frame, $\{f\}$
Segment		***(i)*** the link is the part of the segment specification and its constraints are also valid in this context, ***(ii)*** all the joint pose frames are defined with respect to the reference feature pose frame defined on the link, $\{f\}$, ***(iii)*** the pose relations between the reference feature pose frame and the joint attachment pose frames are constant, ***(iv)*** the segment can have more than one joint pose frames.
Joint		***(i)*** the joint can only be specified between two joint attachment pose frames, ***(ii)*** the joint attachment frames cannot be on the same segment, ***(iii)*** **S** is defined in the coordinates of the target joint attachment pose frame.
KinematicChain		***(i)*** a joint attachment pose frame on the segment that is used to define a joint connection cannot be reused to define another joint connection. That is, a single unique joint requires a single unique joint attachment on each segment it connects; ***(ii)*** if the joint has been set as the root joint of the kinematic chain, its reference joint attachement pose frame cannot be used as a target joint attachment pose frame in any other joint and its reference body becomes the base/root body of the chain; ***(iii)*** after the root joint of the kinematic tree has been chosen, the polarity property of all joints in the chain should hold.

Table 2.3: Composition semantics of the structural models. The geometric primitives in the compositions can have the coordinate semantics as given in Table 2.2.

But unlike full six DoF geometric relations of *free* rigid bodies, the relations in kinematic mechanisms are constrained by the DoF of the joints determined by the motion subspace $\hat{S}_M \subseteq \bar{M}^6$ that can be represented as the range of the matrix $\boldsymbol{S}_{6\times n_f}$. These constraints need also be reflected in the operations.

Geometric relation	Semantic operations
Pose	change(Ref)Point change(Ref)OrientationFrame compose, inverse
Twist	change(Ref)Point compose, inverse
Wrench	change(Ref)Point compose, inverse

Table 2.4: The geometric relations and a list of semantically valid operations that can be applied. An excerpt from [De Laet et al., 2013b].

Pose operations: can consist of any combination of pose operations from Table 2.4 and are collectively referred to as *transformations*.

Constraints: no further constraints are present on the pose semantic operations in the context of the kinematic chains. The motion subspace matrix $\boldsymbol{S}$ introduces changes on the coordinate level, e.g., in 1-DoF prismatic joint all coordinate values in the constrained degrees of freedom remain constant, except for the DoF indicated in $\boldsymbol{S}$. That is why, predefining the complete pose specification with the concrete choice of the coordinate representation based on $\boldsymbol{S}$ simplifies the computations.

Velocity operations: can consist of any combination of twist operations from Table 2.4 are collectively referred to as *transformations*. The relative twist relation between two freely moving bodies is given as in

$$\dot{X}_{relative} = \dot{X}_{body_{i+1}} - \dot{X}_{body_i}, \text{with } \dot{X}_{(\cdot)} \in \bar{M}^6, i \in \mathbb{N}.$$

The same relationship holds for the relative motion of two segments in the kinematic chain, except for the 'free' motion of the segments is constrained by the connecting joint's $\hat{S}_M$. The joint is defined as the relationship between two unique joint attachment pose frames, Section 2.4.1, hence the relative motion

of two segments is given as the difference of the twists of two bodies that are measured at the points which are at the origins of the joint attachment pose frames. That is, in Figure 2.7 with the polarity set to $\{\mathbf{j}_0|\mathbf{A}\}$

$$\dot{X}_{relative}(j'|\mathbf{C},\mathbf{A}) = \Big(\dot{X}_{segment_{i+1}}(j'|\mathbf{C},\mathbf{B}) - \dot{X}_{segment_i}(j_0|\mathbf{A},\mathbf{B})\Big),$$

$$\text{with } \dot{X}_{(\cdot)} \in \hat{S}_M.$$

One can also express the relation above using the semantic operation on twists from Table 2.4

$$\dot{X}_{relative}(j'|\mathbf{C},\mathbf{A}) = \text{compose}\Big(\dot{X}_{segment_{i+1}}(j'|\mathbf{C},\mathbf{B}), \text{inverse}(\dot{X}_{segment_i}(j_0|\mathbf{A},\mathbf{B}))\Big).$$

Even though this semantic model is sufficient to describe the operations on the kinematic chains, many algorithms rely on the joint model that uses the concrete matrix representation of the motion subspace, $\boldsymbol{S}_{6\times n_f}$ and the vector of joint variables, $(\mathbf{q}_{n_f\times 1}, \dot{\mathbf{q}}_{n_f\times 1}) \in \mathbb{R}$ that are expressed in target joint attachment pose frame coordinates. Thus, for the example above an equivalent model using motion subspace matrix representation in concrete coordinates is given as

$$\dot{\boldsymbol{X}}_{relative}(j'|\mathbf{C},\mathbf{A},[J']) = \dot{\boldsymbol{X}}_{joint} = \boldsymbol{S}(\boldsymbol{q}) \cdot \dot{\mathbf{q}}.$$

There are also other time dependent terms in this relationship that are not considered in this context [Featherstone, 2008].

Constraints: ***(i)*** the twist operations require the pose operations be updated beforehand, i.e., the pose relation between target and reference joint attachment frames is required.

Acceleration operations: the acceleration twists have a lot in common with the velocity twists and share most of the semantic operations. Therefore, acceleration twist operations can consist of any combination of twist operations from Table 2.4. For Figure 2.7, semantics of the acceleration twist relations is given as

$$\ddot{X}_{relative}(j'|\mathbf{C},\mathbf{A}) = \Big(\ddot{X}_{segment_{i+1}}(j'|\mathbf{C},\mathbf{B}) - \ddot{X}_{segment_i}(j_0|\mathbf{A},\mathbf{B})\Big),$$

$$\ddot{X}_{relative}(j'|\mathbf{C},\mathbf{A}) = \text{compose}\Big(\ddot{X}_{segment_{i+1}}(j'|\mathbf{C},\mathbf{B}), \text{inverse}(\ddot{X}_{segment_i}(j_0|\mathbf{A},\mathbf{B}))\Big),$$

$$\text{with } \ddot{X}_{(\cdot)} \in \hat{S}_M.$$

As with the velocity twist using 6D spatial vector representation, the acceleration twist 6D vector is not part of the Euclidean vector space. These non-Euclidean

velocity and acceleration twists are often referred to as the spatial or motor twists. Euclidean (conventional) and the spatial twists describe the same motion of the body, albeit with different reference frames [Featherstone, 2001, Stramigioli and Bruyninckx, 2001]. The acceleration twist of the connected mechanism is also influenced by the joint's geometric model. As in the case of the coordinate specific velocity twists, the acceleration twist in concrete coordinates uses motion subspace matrix, $\boldsymbol{S}$ and is obtained by differentiating joint twist expression given in the previous paragraph, where $\ddot{\mathbf{q}}_{n_f \times 1} \in \mathbb{R}$:

$$\ddot{\boldsymbol{X}}_{relative}(j'|\mathbf{C}, \mathbf{A}, [J']) = \dot{\boldsymbol{X}}_{joint} = \boldsymbol{S} \cdot \ddot{\boldsymbol{q}} + \dot{\boldsymbol{S}} \cdot \dot{\boldsymbol{q}}.$$

Here, the derivative of the time dependent terms were dropped [Featherstone, 2008].

Constraints: ***(i)*** the acceleration twist operations require the velocity twist be updated before.

Wrench operations : The wrench, also known as the spatial force or screw wrench, is the dual of the spatial twists, therefore they have the same set of the base operations and operation semantics as given in Table 2.4. Assume there is a wrench $F_{segment_i}(j_0|\mathbf{A}, \mathbf{B})$ Because of the motion constraint imposed by the joint, only part of this wrench is transmitted across the joint to the next connected segment and is given as $F_{segment_{i+1}}(j'|\mathbf{C}, \mathbf{B})$. The transmitted wrench is determined by

$$\boldsymbol{\tau}(j'|\mathbf{C}, \mathbf{B}, [J']) = \boldsymbol{S}^{\mathrm{T}} \cdot \boldsymbol{F}(j'|\mathbf{C}, \mathbf{B}, [J']), \text{with } \boldsymbol{\tau}_{n_f \times 1} \in \mathbb{R}.$$

Here $\boldsymbol{\tau}$ is a joint force variable. The above expression also follows from the fact that all constraint forces F_c resulting from the constrained motion of the joint act in a subspace $\hat{S}_c$ and is a reciprocal complement of $\hat{S}_M$, $\hat{S}_c \perp \hat{S}_M$ [Duffy, 1990]. Thus, $\boldsymbol{S}^T \cdot \boldsymbol{F_c} = 0$

Constraints: ***(i)*** the wrench operations require pose and twists operations be updated beforehand.

Inertia operations : Spatial inertia, H of a rigid body defines a linear mapping relation between its velocity twist and spatial momentum. The spatial inertia depends on a point and an orientation frame on the rigid body. For Figure 2.5, the semantics of the spatial inertia expressed in terms of the geometric primitives is given as

$$p(cm|\mathbf{A}, \mathbf{B}) = H\Big(X(\{cm\}|\mathbf{A}, \{f\}|\mathbf{A})\Big) \cdot \dot{X}(cm|\mathbf{A}, \mathbf{B}), \text{with } p \in \bar{F}^6,\ \dot{X} \in \bar{M}^6.$$

The following are the semantic operations that are defined on the spatial inertia:

$$H\Big(X((cm', [CM])|\mathbf{A}, (f, [F])|\mathbf{A})\Big) = H\Big(X((cm, [CM])|\mathbf{A}, (f, [F]|\mathbf{A}))\Big).$$
$$.\text{changePoint}\Big(\text{Position}(cm', |\mathbf{A}, cm|\mathbf{A})\Big);$$

$$H\Big((cm, [CM'])|\mathbf{A}\Big) = H\Big((cm, [CM])|\mathbf{A}\Big).\text{changeOrientFrame}$$
$$\Big(\text{Orientation}([CM'], |\mathbf{A}, [CM]|\mathbf{A})\Big);$$

$$H(\{cm\}|\mathbf{A}) = \text{compose}\Big(H(\{cm'\}|\mathbf{A}'), H(\{cm''\}|\mathbf{A}'')\Big).$$

The coordinate semantics of the spatial inertia depends on how the coordinate semantics of the motion and the momentum are defined. For example, if the linear components are before the angular components in the representations of the motion, i.e., (v, ω) vs (ω, v) and momentum then the spatial inertia's matrix representation is given as

$$\boldsymbol{H} = \begin{pmatrix} \boldsymbol{m} \cdot \mathbf{1}_{3\times 3} & \mathbf{0}_{3\times 3} \\ \mathbf{0} & \boldsymbol{I_{cm}}_{3\times 3} \end{pmatrix}.$$

The coordinate transformation of the inertia follows from the fact that it is a linear mapping operator between the motion and force spaces:

$$\boldsymbol{p}_{\{cm\}} = \boldsymbol{H}_{\{cm\}} \cdot \dot{\boldsymbol{X}}_{\{cm\}} \text{ and } \boldsymbol{p}_{\{cm'\}} = \boldsymbol{H}_{\{cm'\}} \cdot \dot{\boldsymbol{X}}_{\{cm'\}};$$

$$\dot{\boldsymbol{X}}_{\{cm'\}} = \Big({}^{\{cm'\}}\boldsymbol{T}\boldsymbol{v}_{\{cm\}}\Big)\dot{\boldsymbol{X}}_{\{cm\}} \text{ and } \boldsymbol{p}_{\{cm'\}} = \Big(({}^{\{cm'\}}\boldsymbol{T}\boldsymbol{f}_{\{cm\}})\Big)\boldsymbol{p}_{\{cm\}};$$

$$\underbrace{\Big(({}^{\{cm'\}}\boldsymbol{T}\boldsymbol{v}_{\{cm\}})^{-T}\Big)}_{\Big(({}^{\{cm'\}}\boldsymbol{T}\boldsymbol{f}_{\{cm\}})\Big)}\boldsymbol{p}_{\{cm\}} = \boldsymbol{H}_{\{cm'\}}\Big({}^{\{cm'\}}\boldsymbol{T}\boldsymbol{v}_{\{cm\}}\Big)\dot{\boldsymbol{X}}_{\{cm\}} \Rightarrow$$

$$\boldsymbol{p}_{\{cm\}} = \underbrace{\Big(({}^{\{cm'\}}\boldsymbol{T}\boldsymbol{v}_{\{cm\}})^{T}\Big)\boldsymbol{H}_{\{cm'\}}\Big({}^{\{cm'\}}\boldsymbol{T}\boldsymbol{v}_{\{cm\}}\Big)}_{\boldsymbol{H}_{\{cm\}}}\dot{\boldsymbol{X}}_{\{cm\}}.$$

Here $\Big(({}^{\{cm'\}}\boldsymbol{T}\boldsymbol{f}_{\{cm\}})\Big)$ and $\Big(({}^{\{cm'\}}\boldsymbol{T}\boldsymbol{v}_{\{cm\}})\Big)$ are the coordinate transformations for the force and motion spaces.

Constraints: ***(i)*** the total spatial inertia of the two-body system is determined by how these rigid bodies are 'connected' [Featherstone, 2008]. This constraint is expressed by

$$\boldsymbol{H}_A = \boldsymbol{H}_{A'} + \boldsymbol{H}_{A''} - \boldsymbol{H}_{A''}\boldsymbol{S}\left(\boldsymbol{S}^T\boldsymbol{H}_{A''}\boldsymbol{S}\right)^{-1}\boldsymbol{S}^T\boldsymbol{H}_{A''}. \tag{2.1}$$

This equation is derived explicitly in Chapter 3, Equation (3.13). If the bodies are rigidly connected, $\boldsymbol{S} = \boldsymbol{0}_{6\times n_f}$, then the negative term in the equation disappears and the total spatial inertia of the two-body system is the sum of the inertias of the constituting bodies. If the rigid bodies are not connected to each other, i.e., 'free', $\boldsymbol{S} = \boldsymbol{1}_{6\times n_f}$, then the complete terms involving $\boldsymbol{H}_{A''}$ disappear which physically means that the total inertia of each body is equal to its spatial rigid body inertia.

2.5 Software design

This section reports on the details of the DSL implementation that uses the semantic models described above. The DSL enables programmatic construction of robot models and algorithms with semantic checking capabilities. The concrete knowledge of the specific coordinates is added by the compiler tools, which instantiate generic DSL program in the form of the specific programming language code. As discussed in Section 1.5.3, it can be achieved in two ways: as an external DSL or an internal DSL.

From a programming language point of view, the procedure to process the models and to generate a specific code is conceptually the same for both types of the DSLs. Figure 2.9 shows a simplified view on this model-to-code transformation process. The blocks in the enclosing red area are a part of the internal DSL processing phase. If one encodes the semantic models and their instances directly in a host programming language, here represented as the rectangle with the rounded corners in the red area, the language infrastructure takes care of the most of the validation process. Additionally, the constraint checking partially relies on the type checking, resolution of the scopes and the order of the function calls as implemented by the host language's infrastructure. For instance, in a **C++** template-metaprogramming-based implementation the model checking procedure relies on the type checking, partial instantiation and overload resolution features of the language infrastructure. Examples: to check whether a complex operations such as a forward velocity twist is correctly composed from the simpler operations of the pose and the twist; a semantically correct *compose* operation is invoked depending on the type of the inputs it receives, i.e., the composition of poses vs the composition

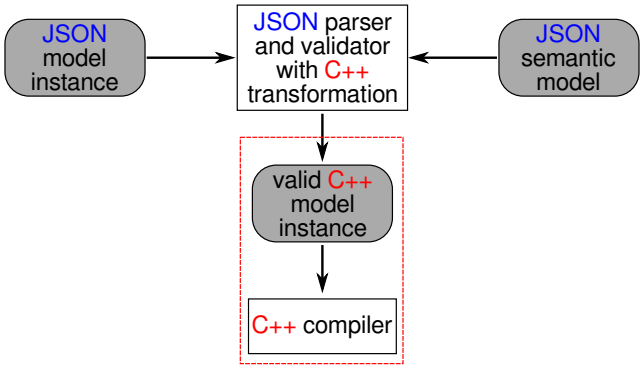


Figure 2.9: Model-to-code transformation process for models represented in JavaScript Object Notation(JSON) [Crockford, 2006, Zyp et al., 2013] and C++ languages. When the modeling language is implemented as the internal DSL, only the steps in the enclosing dashed area are carried out. The rectangles with the rounded corners represent data, i.e., the models to be processed and their concrete instances, the other blocks represent the transformation procedure performed by the compiler tool-chain.

of twists. The current implementation of the internal DSL is based on the template-metaprogramming features of the **C++** host language. This choice was partially influenced by the fact that the original geometric relations semantics library [De Laet et al., 2013a, De Laet and Bellens, 2012] is also implemented using **C++** templates. Furthermore, a template instantiation mechanism creates a new type with every new template parameter. This approach of the type enforcement and checking facilitates the implementation of some of the DSL validation mechanisms. When the modeling language is implemented as the external DSL, all necessary infrastructure is developed ground up as in any other programming language. Hence, it requires an implementation of both a front-end, including a lexical analysis, a definition of the syntax, a parser to transform the input into something meaningful based on the given syntax (a syntax analysis), and a semantic analysis and a back-end that is responsible for a target code generation [Scott, 2009]. In the context of this work, the external DSL uses JSON representation as the basis for its syntax, thus it relies on the lexical and the syntax analysis capabilities of the existing JSON parsers [Zyp et al., 2013]. Taking this into account, the model-to-code transformation process in Figure 2.9 for the external DSL using JSON representation includes the rest of the blocks in addition to the ones in the red area. Furthermore, the use of JSON allows to perform partial semantic analysis, where the semantics of the models, i.e., meta-models, are encoded in JSON schema. So, the schema together with the parser and validator software form a simplified front-end of the DSL tool-chain

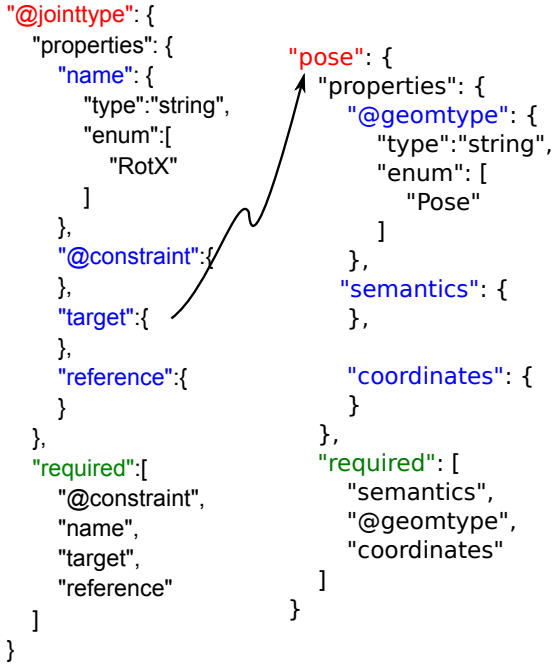


Figure 2.10: A model composition using JSON schema. A joint is defined as a constrained relative motion between two bodies. The relative constrained pose of the bodies is determined by the pose relationship of the pose frames (a frame and a point at its origin) fixed on the bodies: the target and the reference. Each pose frame specification in joint scheme (on the left) references to and complies with the pose scheme (on the right). Any modifications to the latter does not affect the rest of the former.

(the blocks outside of the red area). One of the main advantages of using JSON schema is that a scheme can reference any other already existing schema. This allows for the re-usability of the existing models in JSON and the resulting new schema are modular, i.e., any changes in the referenced schema are local and affect only the models in the referencing schema that need to comply with the referent. Figure 2.10 illustrates this situation on the basis of the JSON scheme for the semantic model of the joint. The scheme shows that the constrained motion of two bodies connected by the joint is determined by the constrained geometric relationships of two pose frames (a frame and a point at its origin) fixed on the bodies: a target and a reference. The complete model of these two pose frames is defined in another scheme somewhere else and is only being referenced. This complete semantic model of the pose, shown on the right side has two parts: a coordinate semantics and a concrete coordinate,

```
"posesemantics": {
  "type": "object",
  "properties": {
    "target": {
      "$ref": "#/definitions/pointframeonbody"
    },

    "reference": {
      "$ref": "#/definitions/pointframeonbody"
    }
  },
  "required":[
  "target",
  "reference"
  ]
}
```

Listing 2.2: Pose semantics in JSON representation as defined in terms of a point, a frame and a body modeling primitives.

where the coordinate semantics is the composition of the coordinate invariant semantics and a reference coordinate frame as given in Table 2.2. The JSON representation of the coordinate invariant semantics is given in Listing 2.2 and the list of the possible concrete coordinates that can be used with these semantics is given in Listing 2.3. As it can be seen from these listings, both of the models refer to other schema representing the semantics of the different required sub-models. During the parsing process these references are resolved to the complete schema representing respective semantic models, which are then used during the validation. Since any modifications are local to the schema, in Listing 2.3 one could add further concrete coordinate representations that need to be supported without the joint scheme being aware of these. An example showing an excerpt of the valid input model instance which compiles with the schema above is provided in Listing A.1 in Appendix A.

As stated, JSON schema enable partial semantic analysis, but a set of the semantic rules are still a part of the validator software and are often implemented in the same programming language as the target output of the code-generation process. For example, if one requires **C** code as the target output, it is often simpler to adopt the parser/validator implemented in the same language. This is a pragmatic approach to the external DSL implementation and requires a different parser/validator library for every new target. This research combines the efforts of developing the internal and the external DSLs by choosing **C++** as the target language. This allows to integrate the existing legacy code from the geometric relations semantics **C++** template library [De Laet et al., 2013a, De Laet and Bellens, 2012]. The semantics models presented in Section 2.4 are encoded both in JSON schema and **C++**. Furthermore, the semantic models of

```
"geometric_coordinates": {
  "type":"object",
  "items":
  {
    "oneOf":[
    {
      "$ref": "#/definitions/vector3DKDL"
    },

    {
      "$ref": "#/definitions/vector6DKDL"
    },

    {
      "$ref": "#/definitions/matrix4x4KDL"
    },

    {
      "$ref": "#/definitions/quaternionKDL"
    }
    ]
  }
}
```

Listing 2.3: A list of coordinate representations in JSON that can be used with a geometric relation, e.g `Pose` relation as in Listing 2.2.

geometric relations semantics need also be encoded in JSON schema as shown in the examples so far. As depicted in Figure 2.9, the **C++** implementation of the models and transformation rules are part of the parser/validator block that parses input JSON model instances and validates whether the input complies with the JSON schema. When the validation step is passed, a matching and a correct **C++** instance of the input model is generated. The correctness and validity of the **C++** models relies on the fact that the transformation rules are implemented correctly and that need to be ensured beforehand.

Figure 2.11 shows a simplified class diagram of the class library that models **C++** representation of the semantic models for the kinematic chains, which have been presented in Section 2.4. The implementation that complies with this diagram realizes the same set of semantic constraints encoded in JSON representation. A more interesting aspect of the **C++** implementation is depicted in Figure 2.12. From the computational point of view, any operation associated with the manipulation of the chain's computational state, e.g., poses, twists of the segments and joints, can be expressed as an iterative or a recursive operation on a connected directed graph, in this context a tree, and a set of operations that are carried out on each node or edge of the graph. In a graph terminology such an iteration is often called a *walk* on or a *traversal*

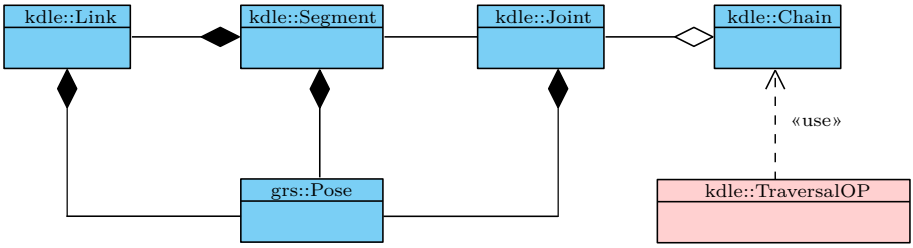


Figure 2.11: The class diagram describes the relationships between the structural model primitives. Here *kdle* and *grs* are namespaces. The Pose class carries part of the semantic information which will be used by the other structural classes. The complete semantic and structural information is carried by the Chain class. The Chain class is related to the operation models through the TraversalOP class.

of the graph. This traversal can be performed in multiple ways also referred to as a traversal policy, e.g., a depth first traversal/search (DFS), a breadth first traversal/search (BFS), that also affects the outcome of the traversal. The implementation of the operation classes in Figure 2.12 complies with these graph concepts. Each class in the diagram is realized as a functor class that has no or very little state of its own. The `kdle::TraversalOP` class is the top level class that brings all other classes together. The function of `kdle::TraversalOP` is to traverse a kinematic chain, `kdle::Chain` using some traversal policy, `kdle::TraversalPolicy`, while applying a set of operations, composition of Pose, Twist and Wrench operations `ComputationOP` to update the state, `kdle::StateGraph` associated with the chain. Each applied operation is checked for the semantic validity using the semantic data in the chain. The chain semantic data is defined in JSON models for the case of the external DSL or directly encoded using the classes in Figure 2.11 at the construction time for the case of the internal DSL using **C++**.

A simplified example that shows a use of the library as the internal DSL is given in Listing 2.4. The listing exempts some of the specifications, e.g., definitions of the pose semantics, coordinates and state graphs, and focuses on the definition of the kinematic chain primitives. Lines 1–13 show the specification of the pose frames with joint attachment semantics. The attachment semantics is determined by a tag that indicates the purpose of the pose frame. In this listing the tag used is `PoseFrameType::Joint`. There exist other tags such as `PoseFrameType::Inertia` or `PoseFrameType::Feature`. After the constructed joint pose frames are attached to two segments, one per segment on Lines 14–17, Line 20 defines the joint in terms of the joint attachment pose frames and joint properties that include the motion subspace, polarity, shaft inertia and joint

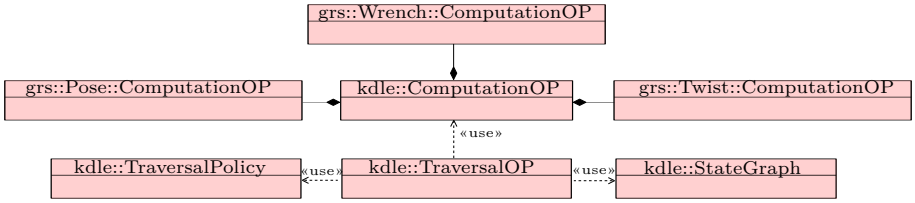


Figure 2.12: The class diagram describes the relationships between the operational model primitives. Here `OP` suffix stands for an operation. The operations in the context of the geometric relations represent the semantic operations associated with each relation. Examples include changeReferencePoint, changeCoordinate, compose and their combinations. The full list of the semantic operations is presented in [De Laet et al., 2013b].

limits, Lines 18–19. Note how the motion subspace matrix that represents motion subspace $\hat{S}_M$ of the joint is specified as a symbolic representation `JointTypes::REVOLUTE_Z`, which defines a constrained pose relation between the target and the reference joint attachment pose frames. The polarity of the joint is specified by the order of the arguments at the joint construction time, where the attachment pose frame specified at the latest is the reference. The current implementation provides only a place holder for the most of the joint properties as a tuple type. This will be made explicit in the future releases. On Lines 21–23 the specified joint becomes part of the joint list, which is used in the specification of the kinematic chain. Lines 25–26 specify a policy that defines how the constructed chain is traversed by the *traverseGraph* operation on Line 28. Algorithm 2.1 summarizes this procedure for the example presented in the next section.

2.6 Example

This section gives an example of how the models from Section 2.4 and their implementation can be used in an application. First, it describes the algorithmic procedure to create a kinematic chain and define the operations with semantic information. The kinematic chain to be constructed is depicted in Figure 2.13. In order to avoid overloading the figure with the frame notation, the following convention is assumed: each segment (except for the first and the last) in the figure has two joint attachment pose frames which are in use and coincide with the segment proximal end frames (a frame with a point at its origin). This allows the mechanism topology to be serial, which is a familiar topology of the majority of robot manipulators. Segment proximal end frames retain the index number of the segment in their names. That is, Segment0 has $\{f_0\}$ and $\{l_0\}$ frames,

```
1  //Segment1 JOINT AttachmentFrames
2  //JointFrame1
3  grs::Pose<KDL::Vector, KDL::Rotation> seg1Joint1FramePose(
       seg1Joint1FramePosition, seg1Joint1FrameOrientation);
4  //Add the JointFrame1 to the AttachmentFrames of the Segment
5  AttachmentFrames frameList1;
6  frameList1.push_back(kdle::createAttachmentFrame(
       seg1Joint1FramePose, kdle::PoseFrameType::JOINT));
7
8  //Segment2 JOINT AttachmentFrames
9  //JointFrame1
10 grs::Pose<KDL::Vector, KDL::Rotation> seg2Joint1FramePose(
       seg2Joint1FramePosition, seg2Joint1FrameOrientation);
11 AttachmentFrames frameList2;
12 frameList2.push_back(kdle::createAttachmentFrame(
       seg2Joint1FramePose, kdle::PoseFrameType::JOINT));
13
14 //Segment specification with two argument Pose
15 kdle::Segment< grs::Pose<KDL::Vector, KDL::Rotation> > segment1
       ("Segment1", link1, frameList1);
16 kdle::Segment< grs::Pose<KDL::Vector, KDL::Rotation> > segment2
       ("Segment2", link2, frameList2);
17 //Joint specification
18 kdle::JointProperties joint1_props;
19 joint1_props = std::make_tuple(kdle::JointTypes::REVOLUTE_Z,
       0.025, 145.0, -135.0);
20 kdle::Joint< grs::Pose<KDL::Vector, KDL::Rotation> > joint1("
       Joint1", segment2.getAttachmentFrames()[0], segment2,
       segment1.getAttachmentFrames()[0], segment1, joint1_props);
21 //Chain specification
22 std::vector< kdle::Joint< grs::Pose<KDL::Vector, KDL::Rotation>
        > > jointlist(joint1);
23 kdle::KinematicChain< grs::Pose<KDL::Vector, KDL::Rotation> >
       mychain("MyKinematicChain", jointlist);
24
25 //Traversal policy
26 kdle::DFSPolicy< kdle::KinematicChain< grs::Pose<KDL::Vector,
       KDL::Rotation> > > mypolicy;
27 //Chain traversal operation
28 kdle::traverseGraph(mychain, forwardPoseKinematics, mypolicy)(
       jstate, lstateIn, lstateOut);
```

Listing 2.4: Specification of a kinematic chain using the C++ template-based internal DSL. Here kdle::JointProperties are represented as a tuple type, whose elements contain the values for the motion subspace matrix, joint shaft inertia and joint limits in this particular example. The listing is associated with the setup in Figure 2.13.

Segment1 has $\{f_1\}$ and $\{l_1\}$ frames and so on. Joint attachment pose frames are numbered sequentially, starting with $\{j_0\}$ on Segment0, $\{j_1\}$ and $\{j_2\}$ on Segment1 and so on. Additionally, the chain has a camera and a gripper attached

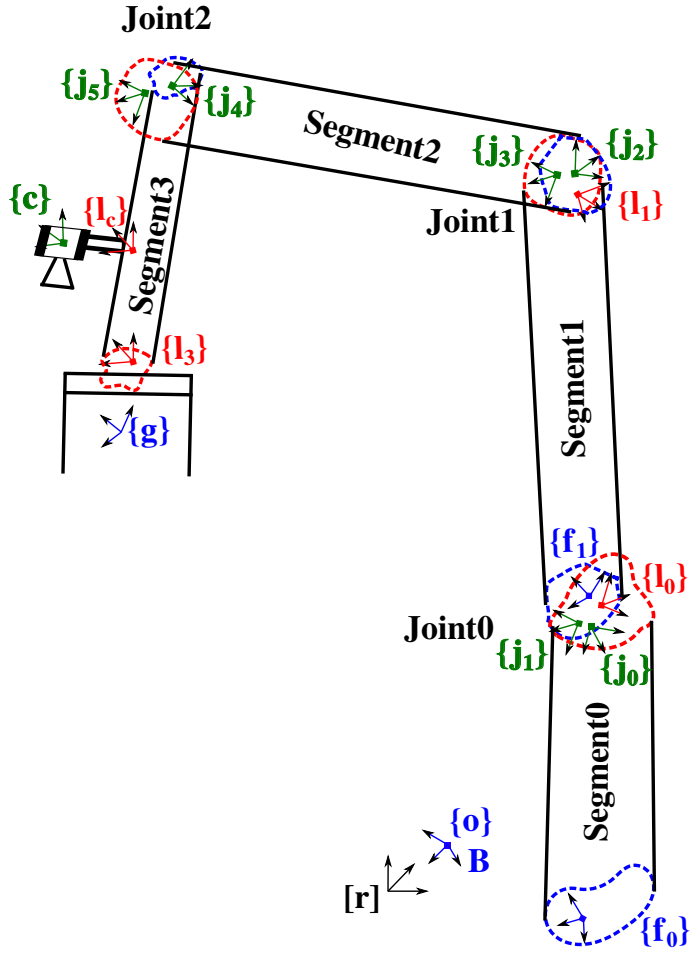


Figure 2.13: A serial kinematic mechanism and its segment and joint frames. $\text{Joint}_i(\text{Pose}(\{j_{2i+1}\}|\text{Segment}_{i+1}, \{j_{2i}\}|\text{Segment}_i), \hat{S}_{Mi}, \text{polarity}_i),\ i \in \{0, 1, 2\}$.

to it at pose frames $\{l_C\}$ and $\{l_3\}$ respectively. The Algorithm 2.1 shows a general procedure to construct a semantically valid chain and compute its kinematics or dynamics. The algorithm focuses on the steps to take to compute the kinematics with the semantic data, rather than the specification of the complete list of the arguments, their names and types that the operations at each step should take. A concrete application of this procedure using **C++** internal DSL is shown in Listing 2.4 that also includes type and name information.

The general procedure consists of the following steps: first, one is to create

the complete pose data of the frames of interest by choosing among available geometric and coordinate models, Lines 2-5 and 7-10. For instance, the pose can be described by a position of a point and an orientation frame. Then, the defined complete pose semantics are composed to specify the link primitive, Line 6 and get added extra attachment semantics to determine their other uses. That is, whether the poses are used to define attachment locations for the joints, inertia, task or any other feature, e.g., locations of the sensors, Line 11 defines joint attachment poses. The constructed link primitives and attachment pose frames with special purpose get composed into the segment primitives, Lines 12-13. They are then used to define a relative motion using the joint, Lines 14-15, which are collected into the kinematic chain, Line 16. The constructed chain is then traversed by applying operations at each joint and segment, to obtain the requested result, Lines 17-20.

Listing 2.5 shows an excerpt of the output of the operations on the serial chain. The output follows the same notational convention which is described in the Table 2.2. Here, to enable unique naming convention, the names of the geometric primitives are concatenated from the names of the primitives they are part of. For instance, the Twist primitive on the Line 17 relates the velocity twist of body Segment_2 relative to body Segment_1 measured at point l_2 and expressed in $[L_2]$ coordinate frame on body Segment_2. As can be seen from the listing, each geometric primitive which was involved in the operations carries semantic data with itself. When the program is executed it parses the semantic description of the robot structure and applies pose and twist operations where necessary. The semantic knowledge in operations is matched with that in the robot structure to check whether the outcome is valid. The output of the program can be redirected to other analysis tools. This allows to check whether the results of the algorithm are correct not only on the basis of the numerical data but also the semantic data during all the stages of the program. This explicit knowledge of each data primitive and action also facilitates understanding of the written code. This is unlike the example in Listing 2.1 with a lot of implicit assumptions.

In case the models constructed do not conform to all the semantic constraints in the DSL, the language and its toolchain infrastructure provides data on the source of the problems. Such "notifications" on the semantically invalid constructs are provided on two levels: on the geometric, e.g on Pose, Twist, Wrench transformations, and the kinematic level, e.g. on the construction of Joints and KinematicChains. Listing 2.6 shows an excerpt from the output of the computation of the relative velocity twist of the segment Segment_2 with respect to the previous segment Segment_1 measured at the point l_2 and expressed in the coordinate frame L_2. But the computation of `changeCoordinateFrame` returned with NOK, because the coordinate frames did not match. This level of the geometric diagnostics is provided by the geometric relations semantics

Algorithm 2.1: Definition of the kinematic chain with semantic data.

Input : N number of segments, M number of joints
Names and coordinates of points, frames and bodies

```
1  begin
2      for i ← 0 to 2N − 1 do
3          define PoseSemantics ;
4          define PoseCoordinates ;
5          compose Pose from semantics and coordinates;
6      compose Link from proximal and distal Pose data ;
7      for i ← 0 to 2M − 1 do
8          define PoseSemantics ;
9          define PoseCoordinates ;
10         compose Pose from semantics and coordinate;
11     define JointAttachmentFrame from Pose data;
12     for i ← 0 to N − 1 do
13         compose Segments from Links and JointAttachmentFrames ;
14     for i ← 0 to M − 1 do
15         compose Joints from
                                JointPose[i]|Segment[j]
                                JointPose[i + 1]|Segment[j+1]
                                JointProperties
16     compose KinematicChain from Joints ;
17     define PoseTransformation operation;
18     define TwistTransformation operation;
19     define Composite operation;
20     compose (TwistTransformation,PoseTransformation)
       traverse(KinematicChain,Composite)
```

library that the DSLs use [De Laet and Bellens, 2012]. The other problem indicated on Line 7 is that the relative joint frame pose can not be computed, because the joint constructured is invalid. From here one can deduce that there is a problem with the joint attachment frames. This kinematic diagnostics comes from the kinematics and dynamics DSL type system itself.

```
Composite operation is created
Traverse operation is created
3 argument traverse call
3 argument composition call
JOINT NAME Joint1

Inside Joint Pose Impl
Pose((j3.Segment2.Link2,[J3.Segment2.Link2])|Segment2.Link2,(j2
    .Segment1.Link1,[J2.Segment1.Link1])|Segment1.Link1,[J2.
    Segment1.Link1]) with position coordinates:[           0,
              0,           0] and orientation coordinates: [
    0.707107,    -0.707107,           0;
0.707107,    0.707107,           0;
0,           0,           1]
Inside Distal to Predecessor Joint Pose Impl
Pose((l2.Segment2.Link2,[L2.Segment2.Link2])|Segment2.Link2,(j2
    .Segment1.Link1,[J2.Segment1.Link1])|Segment1.Link1,[J2.
    Segment1.Link1]) with position coordinates:[    0.247487,
       0.247487,           0] and orientation coordinates: [
    0.707107,    -0.707107,           0;
0.707107,    0.707107,           0;
0,           0,           1]
Inside Joint Twist Impl
Twist(j3.Segment2.Link2|Segment2.Link2,Segment1.Link1,[J2.
    Segment1.Link1]) with linearVelocity coordinates:[
              0,           0,           0] and angularVelocity
    coordinates: [           0,           0,      0.8555]
Inside Joint DistalToRefJoint Twist Impl
Twist(l2.Segment2.Link2|Segment2.Link2,Segment1.Link1,[L2.
    Segment2.Link2]) with linearVelocity coordinates:[-3.87602e
    -18,    0.299425,           0] and angularVelocity
    coordinates: [           0,           0,      0.8555]
```

Listing 2.5: An excerpt from an output of a program in Listing 2.4 that shows the semantics of the geometric relations associated with the structural primitives, their numerical values and operations performed on them. The listing is associated with the setup in Figure 2.13.

2.7 Discussions and conclusions

This chapter presented the semantic models that describe kinematic chains in a coordinate-free form. These models make use of the semantic models of the geometric relations. Specifically, the structural models of the kinematic chain are built on the geometric modeling primitives of a point, a frame and a body and the pose relationship that uses these primitives. The distinguishing feature of the semantic models developed from the other similar efforts discussed in Section 2.3 is that the geometric primitives and their relationships are assigned different roles based on their functions in the kinematic chain. These roles are referred to as the *attachment semantics*. The addition of the attachment

```
JOINT NAME Joint1

Inside Joint Twist Impl
Twist(j3.Segment2.Link2|Segment2.Link2,Segment1.Link1,[J2.
    Segment1.Link1]) with linearVelocity coordinates:[
    0,           0,           0] and angularVelocity coordinates:
    [           0,           0,      0.8555]

Inside Joint DistalToDistal Twist Impl
LinearVelocity(l2.Segment2.Link2|Segment2.Link2,Segment1.Link1,[
    J2.Segment1.Link1]).changeCoordinateFrame(Orientation([L1.
    Segment1.Link1]|Segment1.Link1,[L2.Segment2.Link2]|Segment2.
    Link2,[L2.Segment2.Link2])) is NOK since:
* Coordinate Frame of LinearVelocity(l2.Segment2.Link2|Segment2.
    Link2,Segment1.Link1,[J2.Segment1.Link1]) != orientation frame
     of Orientation([L1.Segment1.Link1]|Segment1.Link1,[L2.
    Segment2.Link2]|Segment2.Link2,[L2.Segment2.Link2])

Twist(l2.Segment2.Link2|Segment2.Link2,Segment1.Link1,[J2.
    Segment1.Link1]) with linearVelocity coordinates:[-3.87602e
    -18,    0.299425,           0] and angularVelocity coordinates
    : [           0,           0,      0.8555]


Warning: can not return pose data. Check whether the joint is
    correctly constructed
```

Listing 2.6: An excerpt of error/warning output of the program in case constructed models are not semantically valid. In this case, on the geometric level the coordinate frames for the twist transformation do not much and on the kinematic level one of the joint construction constraints is not satisfied.

semantics to the available geometric relations semantics enables an extension of the semantic constraints that the geometric relations should satisfy in a specific application context, Table 2.3. Consequently, this enables creation of more concise models of the kinematic chains and better tailor them to the context of the applications they are used in. The chapter also introduces the semantics and coordinate semantics of inertia of the ideal rigid body. It shows that the inertia serves as a mapping operator between the motion quantities and the force quantities associated with the rigid body. Furthermore, it shows that the net inertial property of the connected rigid bodies is defined by the constraining joint model and is given as in Equation (2.1). The joint model also affects the amount of the force transmitted from one body to the other.

The chapter also presented semantic operations on the geometric relations that are associated with the kinematic chains. There are two points with respect to the constraints on the geometric relations that are to note here. First, geometric relations that describe the relative motions and dynamic interactions of the

segments are constrained to some motion subspace $\hat{S}$. This physical constraint needs to be accounted for in the respective operations of the geometric relations in the kinematic chain. Second, the fact that the segments are connected and part of the single kinematic structure makes the dependencies of the geometric relations on each other more obvious. That is, semantic operations on geometric relations need to be strictly ordered, e.g. the pose operations are performed before the twist operations.

The chapter also introduced the software implementation, which complies with the developed semantic models. The software is implemented both as the internal and the external DSL. Particularly, JSON-based external DSL implementation allows modular and extensible specifications. These are possible because of the modular organization of JSON representations. It allows to separately encode models of various semantic detail in the form of schema that can be composed with each other. Any modifications to the schema are local, the fact that enables updates to the functions of the parser that cope with those schema changes only. Another advantage that comes with the use of JSON representation is the possibility to integrate the semantic models of the other domains that are part of the robot application into a single linked data model. This is attained by the use of JSON-LD, JSON for the linked data representations [Sporny et al., 2013]. This lets to develop a graph of model of the robot application with explicit relations and constraints among coordinate-free semantics and concrete coordinates, their values and physical units. This graph can then be queried for specific quantities. Furthermore, these models can be developed independently and just reference each other as it is shown in example Listing A.2. This is a completely new approach, building on very recent developments in knowledge representation and the "semantic web", with potentially very high impact in robotics. Section 5.6 provides more details on the linked data modelling of the robot applications.

The necessity of the semantic models for the implementations is strengthened by the existence and use of the closely related concept of an AST in a programming language design [Scott, 2009]. The AST of the language is used during the semantic analysis of the program, i.e. to verify whether the primitives of the program are used correctly according to some set of predefined rules. This is also what happens in the implementations presented in this chapter. Based on the semantic models of the DSL, the compiler automatically resolves constraints imposed by the different geometric and coordinate models, when the robot structure and the operations are instantiated using specific representations. One can also state that there are two ASTs in this context: one for the encoding language (JSON, C++) and one for the domain specific semantic models. Each is responsible for its own semantic analysis. This also implies that one can design a DSL with the domain specific semantic models as

the single and main AST. This also allows implementation of the languages and toolchains of different syntactic flavor, for example using the language workbenches or the compiler-compiler tools such as JastAdd [Hedin, 2011]. The developed semantic models and their software implementations are available at the following locations [Shakhimardanov and Bruyninckx, 2014b, Shakhimardanov and Bruyninckx, 2014a]. The latest implementation can only generate C++ models and code from JSON semantic models that use KDL [Smits et al., 2001] coordinate representations. It also allows the coordinate specific implementations of the tree traversal policies, as well as composition of the operations that allow formulation of the computational sweeps in the dynamics algorithms. Furthermore, even though the physical quantities, their dimensions and units are part of the JSON models, their validation is not implemented yet.

Additionally, as it has been discussed in the paragraph on the semantics of the kinematic chain primitive in Section 2.4.1, the presented semantic models do not support the specification of the kinematic chains with the loop structures. On the other hand, this support can be attained by modifying the existing constraints on the connectivity of the joint attachment pose frames for the structural primitives. But for the operations, one is required to account for the additional constraints imposed by the loop joints on the motion, acceleration level constraints, and the wrench across the loop joint. A rough procedure on how to do this is presented in Chapter 8 in [Featherstone, 2008].

Chapter 3

Popov-Vereshchagin Hybrid Dynamics Solver

3.1 Introduction

This chapter introduces a *solver* component that builds upon the models and implementations presented in Chapter 2. The position of the solver component with respect to the other components in the constrained motion task stack is depicted in Figure 3.1. At the basis of the constrained motion task control is the determination of the control inputs that can *stabilize/control* the constraints on the geometric relations between the target objects. The functionality of the solver is to compute these stabilizing control inputs from the given set of target object constraints, the robot motion model (platform constraints) and the cost function (Figure 3.1). A general mathematical formulation of this problem and the techniques that can be used to realize it have been summarized in Section 1.5.4.

This chapter presents a constrained robot hybrid dynamics solver that computes the solution of

$$\begin{bmatrix} \boldsymbol{M}(\boldsymbol{q}) & \boldsymbol{J}_c^T(\boldsymbol{q}) \\ \boldsymbol{J}_c(\boldsymbol{q}) & \boldsymbol{0} \end{bmatrix} \begin{bmatrix} \ddot{\boldsymbol{q}} \\ \boldsymbol{\lambda} \end{bmatrix} = \begin{bmatrix} \boldsymbol{\tau}_a(\boldsymbol{q}) - \boldsymbol{C}(\boldsymbol{q}, \dot{\boldsymbol{q}}) \\ \hat{\boldsymbol{b}}(\boldsymbol{q}, \dot{\boldsymbol{q}}) \end{bmatrix} \tag{3.1}$$

This dynamics algorithm was first introduced by Popov et al. [Vereshchagin, 1989]. Equation (3.1) is the expanded matrix form of the formulation in Equation (1.4),

where $\boldsymbol{J_c}\ddot{\boldsymbol{q}} = \hat{\boldsymbol{b}}(\boldsymbol{q}, \dot{\boldsymbol{q}})$ is the second order derivative of the holonomic* path constraint $\boldsymbol{h}(\boldsymbol{q}) = \boldsymbol{0}$ in (1.4) with $\boldsymbol{J_c}$ being the constraint matrix, $\boldsymbol{\lambda}$ is the vector of Lagrange multipliers with the relation to the constraint forces being $\boldsymbol{\tau_c} = \boldsymbol{J_c^T}\boldsymbol{\lambda}$, $\boldsymbol{C}(\boldsymbol{q}, \dot{\boldsymbol{q}})$ and $\boldsymbol{\tau_a}$ are bias forces and joint space input forces, respectively [Featherstone, 2008]. The interesting feature of the solver is that it can cope with Cartesian space acceleration constraints during computational sweeps. The chapter also develops extensions to this solver that allow its implementation and integration with various types of controllers.

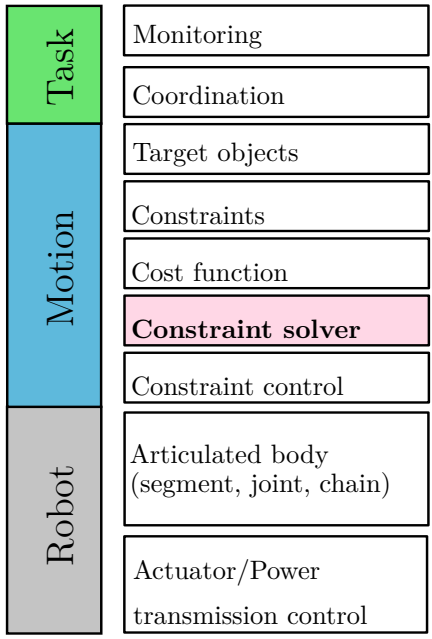


Figure 3.1: Solver specification DSL and its position in the (motion)task programming stack. For a domain specific solver, the robot (algorithm) specification DSL such as the one from the previous chapter can be used.

*There is no difference between holonomic and non-holonomic constraints on the acceleration level.

3.2 Related work

A hybrid dynamics algorithm solves the combined forward/inverse dynamics problem [Featherstone, 2008]: given some actual and desired joint motions and forces, some actual and desired segment motions and forces, as well as the mass distribution of each segment, find the resulting motion of the complete kinematic chain. Hybrid dynamics is the major algorithm for the *posture control* [Khatib et al., 1999] of mobile manipulators and humanoid robots, since such tasks typically require specifications of motion and/or force constraints on the end-effectors and other segments, but also specifications on the *posture* that keeps the robot close to a "most comfortable" configuration (for itself, its task, or its environment) via additional internal joint motions, forces or constraints, or (possibly virtual) external forces or constraints on intermediate segments.

The literature on robot dynamics is extremely rich, from the historical and general dynamics work by [Newton, 1871, Euler, 1776, Lagrange, 1867, Gauss, 1829] (the latter's *minimum principle* being further developed by [Gibbs, 1879, Hertz, 1894] and [Appell, 1899]), to general *multi-body dynamics* [Gantmacher, 1975, Roberson and Schwertassek, 1988, Wittenburg, 1977], and to robot-specific dynamics, first as "order N^3" ($\mathcal{O}(N^3)$) algorithms [Hooker and Margulies, 1965, Lilov and Wittenburg, 1977, Luh et al., 1980a, Roberson and Wittenburg, 1966], and later as $\mathcal{O}(N)$ ("order N") algorithms [Vereshchagin, 1974, Popov et al., 1978, Featherstone, 1983], where N is the number of degrees of freedom in the kinematic chain of the robot.

The Western robotics community has largely been unaware of the groundbreaking work by Popov et al. on linear-time dynamics algorithms for redundant serial robots with (partial) acceleration equality constraints [Vereshchagin, 1974, Popov et al., 1978]. The the linear-time hybrid dynamics algorithm by Popov et al. allows to deal with the motion constraints on the segments at the *acceleration* level, and the specification of each constraint can be only *partial*, i.e., allowing one or more degrees of constraint to be left unspecified [Vereshchagin, 1989]. It computes the magnitudes of the constraint forces that are caused by the imposed Cartesian acceleration constraints based on the Gauss principle of least action [Gauss, 1829] as the function(cost) to optimize.

The chapter first sets the terminology of recursive kinematics and dynamics algorithms and explains already existing inverse, forward and hybrid dynamics algorithms in terms of computational sweeps that use coordinate free form of the operation, Section 3.3. Section 3.4 then introduces Popov-Vereshchagin's original algorithm in this context and compares its differences to the other dynamics algorithms. Section 3.5 extends Popov-Vereshchagin's algorithm with (i) tree-structured chains, (ii) the weighting of (weighted and prioritized) acceleration

equality constraints on the end-effectors, (iii) taking maximum joint torque constraints into account.

Contributions. First, the constrained hybrid dynamics solver is explicitly *decomposed* into its structural and behavioral parts. These are subsequently *composed* in *various* configurable *computations*. This contribution does not lead to new functionalities, but it does offer new ways to integrate extra control approaches. This approach allows for all the extensions to be integrated, in any combination, into the linear-time recursions, without the need to construct an *explicit* Jacobian matrix, or to compute any Jacobian-based pseudo-inverses, *explicitly*. The latter methods are the cornerstone of all cited references, and they introduce, respectively, extra $\mathcal{O}(n^2)$ and $\mathcal{O}(n^3)$ computational complexity, which are avoided in the algorithm implementation presented in this chapter.

Second, chapter also presents extensions of the algorithm to cope with *resolution of over-constrained situations*, Section 3.5. In this case, a policy is needed to decide for the use of, either, priorities between, or weighing of, the conflicting constraints in the joint and Cartesian spaces. The resolution of the conflicting constraints through the prioritization is achieved through the configuration of the computational sweeps of the hybrid dynamics solver. The resolution of the conflicting constraints through the weighing is achieved through the decomposition of the acceleration energy generated during the motion. A subset of these extensions are already described by, for example, [Khatib et al., 2004a, Khatib et al., 2008, Nakanishi et al., 2008, Saab et al., 2013].

3.3 Segment-to-segment recursive dynamics

This Section contains no new material, but summarizes the well-known domain specific solvers for kinematic chains of generic types, for the sake of defining notations and terminology [Featherstone, 2008]. Those solvers have linear-time computational complexity, and use `outward` segment-to-segment recursion to propagate pose, velocity and acceleration from the root to the leafs, Section 3.3.1, or `inward` recursion for force and spatial inertia, Section 3.3.2. These recursive operations are also known as an outward and inward `computational sweeps` on the kinematic chain. A step-by-step introduction of these solvers also serves as a reference for the presentation of the constrained hybrid dynamics solver.

Tree-structured kinematic chains are only slightly more complex than serial chains, because trees are nothing but just a connected set of serial chains: only the bookkeeping of the kinematic and dynamic data structures requires more attention from the programmer, while the recursive segment-by-segment

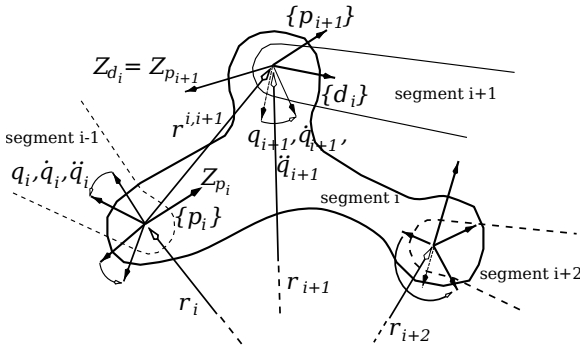


Figure 3.2: Reference attachment pose frames and notation for segments in a tree-structured kinematic chain. "p_i" is the *proximal* joint attachment pose frame of segment i, and d_i is the segment's distal joint attachment pose frame. The frames' Z axes lie along the (single degree of freedom) axes of the joints connected to the segment.

computations remain the same. Figure 3.2 shows the notational conventions of this chapter: (i) joint $i+1$ connects segments i and $i+1$, between the "distal" (i.e., more remote from the reference feature pose frame on a segment) joint attachment pose frame $\{d_i\}$ of segment i and the "proximal" joint attachment pose frame $\{p_{i+1}\}$ of segment $i+1$, i.e. $\{d_i\}$ is the reference and $\{p_{i+1}\}$ is the target; (ii) these pose frames $\{d_i\}$ and $\{p_{i+1}\}$ coincide when joint $i+1$ is in its zero position. Each joint's motion is constrained to motion subspace $\hat{S}$ which can be represented by the matrix $\mathbf{S}$. For a 1-DoF joint, this is equivalent to a directed line along some axis. e.g., Z, that can be represented by a unit vector $\boldsymbol{Z}$. The position vector $\boldsymbol{r}^{i,j}$ connects the origin of the joint attachment pose frame $\{i\}$ to the origin of the joint attachment pose frame $\{j\}$. All physical properties are expressed with respect to the reference point at the origin of the target joint attachment pose frame $\{p_i\}$ of the segment over which the recursion runs.

In the dynamic motion model of the robot in generalized coordinates as in

$$\boldsymbol{M}(\boldsymbol{q})\ddot{\boldsymbol{q}} + \boldsymbol{C}(\boldsymbol{q}, \dot{\boldsymbol{q}}) = \boldsymbol{\tau_a}(\boldsymbol{q}) \tag{3.2}$$

$$\boldsymbol{H}\ddot{\boldsymbol{X}} + \boldsymbol{F_{bias}} = \boldsymbol{F} \tag{3.3}$$

depending whether the motions, $\ddot{\boldsymbol{q}}$ are computed from the forces or the forces, $\boldsymbol{\tau_a}$ from the motions, one can distinguish between the forward and inverse dynamics, respectively. Here, there are no additional constraints on the model, except for those incurred by the joint connections whose net effective work

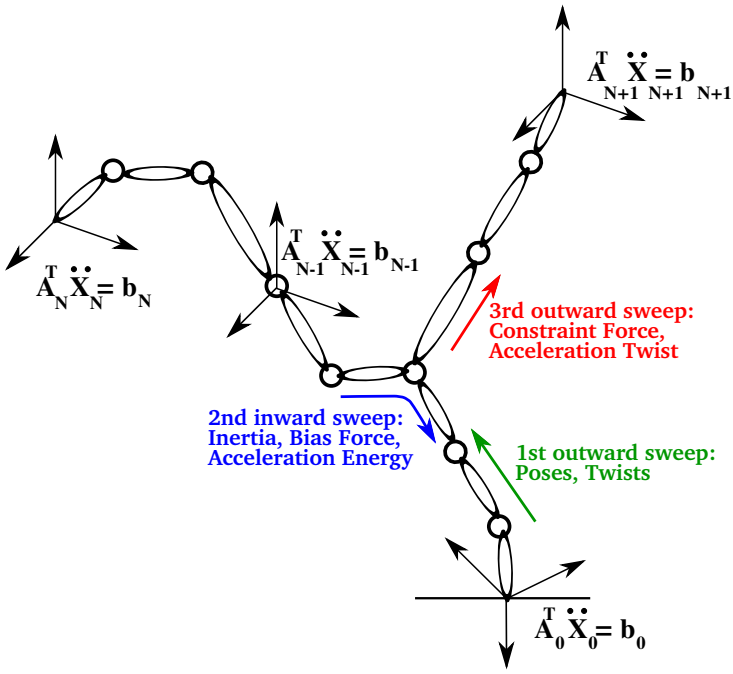


Figure 3.3: In a domain specific dynamics solver computations consist of the outward and the inward sweeps during which the relevant physical quantities of the kinematic tree are computed. In Popov-Vereshchagin's solver the sweeps also take into account the acceleration constraints on the segments. Here $\boldsymbol{A}_i$ is a constraint (unit force) matrix and $\boldsymbol{b}_i$ is a bias acceleration (energy) term.

is zero. These dynamics computations consist of a number of computational sweeps on the kinematic chain as depicted in Figure 3.3. In these sweeps, the variables $\boldsymbol{q}, \dot{\boldsymbol{q}}, \ddot{\boldsymbol{q}}, \boldsymbol{\tau_a}$ at each joint are associated with their spatial counterparts $\boldsymbol{X}, \dot{\boldsymbol{X}}, \ddot{\boldsymbol{X}}, \boldsymbol{F}$ at each segment. Regardless of the type of the dynamics computed the structure of the sweeps, i.e. the semantic operations performed, remains mostly the same. The differences in the computational sweeps will be discussed in the following sections. Without loss of generality and in order not to overload the notation, the semantic operations drop the complete specification of points, frames and bodies in the operation arguments whenever possible. Furthermore, since the algorithms mostly involve pose frames with the joint attachment semantics, these are called simply pose frames without the qualifying semantics.

3.3.1 Outward position, velocity and acceleration recursions

An outward recursion from segment i to segment $i+1$ involves the following operations:

1. segment i: a `change in reference point` from the proximal pose frame $\{p_i\}$ of segment i to this segment's distal pose frame $\{d_i\}$. This changes the vector representation of the physical entity being transformed (pose, velocity, acceleration, force), and the matrix representation of the segment's inertia.
2. joint $i{+}1$: the incorporation of the physical contribution (position, velocity, acceleration, force, joint axis inertia) at joint $\boldsymbol{q}_{i+1}$.

The pose of the proximal pose frame on segment $i{+}1$ with respect to the proximal pose frame on segment i is a function of the intermediate pose relations, where the pose relation between joint's target and reference attachment pose frames is dependent on joint position variable $\boldsymbol{q}_i$:

$$ {}^{p_{i+1}}_{p_i}\boldsymbol{X} = \text{compose}\Big({}^{d_i}_{p_i}\boldsymbol{X}\ {}^{p_{i+1}}_{d_i}\boldsymbol{X}(\boldsymbol{q}_i)\Big), \tag{3.4} $$

with ${}^{d_i}_{p_i}\boldsymbol{X}$ between the $\boldsymbol{i}$ th segment's distal and proximal frames being defined given the reference feature pose frame of the segment, Section 2.4.1. The branching that occurs in a tree-structured topology at segments from which two or more serial chains emerge, does not complicate the motion recursion at all: each outward recursion path remains exactly equivalent to a serial robot, since there is no interaction between two branches.

The velocity twist recursion finds $\dot{\boldsymbol{X}}_{i+1} = (\boldsymbol{\omega}_{i+1}^T, \boldsymbol{v}_{i+1}^T)^T$ of segment $i+1$ with the origin of the proximal frame $\{p_{i+1}\}$ as velocity reference point, given the twist $\dot{\boldsymbol{X}}_i$ of segment i with the origin of the proximal frame $\{p_i\}$ as velocity reference point, and given the joint twist contribution $\dot{\boldsymbol{X}}_j$ that depends on position $\boldsymbol{q}_{i+1}$ and the joint rate $\dot{\boldsymbol{q}}_{i+1}$. The coordinate-free expression for the twists does not differ from the one presented in in Section 2.4.2. The following expression uses twist transformation matrix $\boldsymbol{Tv}$ and the motion subspace matrix $\boldsymbol{S}$:

$$ \dot{\boldsymbol{X}}_{i+1} = ({}^{i+1}\boldsymbol{Tv}_i)\dot{\boldsymbol{X}}_i + \underbrace{\boldsymbol{S}_{i+1}\dot{\boldsymbol{q}}_{\boldsymbol{i+1}}}_{\dot{\boldsymbol{X}}_{\mathbf{joint}}}. \tag{3.5} $$

The acceleration recursion calculates the linear and angular components of acceleration $\ddot{\boldsymbol{X}}_{i+1}$ with the origin of the proximal frame $\{p_{i+1}\}$ as velocity reference point, given the linear and angular acceleration components of $\ddot{\boldsymbol{X}}_i$

with the origin of the proximal frame $\{p_i\}$ as velocity reference point), and given the joint angle $\boldsymbol{q}_{i+1}$, the joint angle speed $\dot{\boldsymbol{q}}_{i+1}$, and the joint acceleration $\ddot{\boldsymbol{q}}_{i+1}$. The result is similar to the velocity recursion, except for the gyroscopic *"bias acceleration"* $\ddot{\boldsymbol{X}}_{bias,i+1}$ of the moving frame $\{p_{i+1}\}$ due to the non-vanishing angular velocity $\boldsymbol{\omega}_i$ of segment i:

$$\ddot{\boldsymbol{X}}_{i+1} = ({}^{i+1}\boldsymbol{T}\boldsymbol{v}_i)\ddot{\boldsymbol{X}}_i + \underbrace{\boldsymbol{S}_{i+1}\ddot{\boldsymbol{q}}_{\boldsymbol{i+1}}}_{\ddot{\boldsymbol{X}}_{\mathbf{joint}}} + \underbrace{(\overbrace{\dot{\hat{\boldsymbol{S}}}_{i+1}}^{\text{time dependent often zero}} + \dot{\boldsymbol{X}}_{i+1} \times \boldsymbol{S}_{i+1})\dot{\boldsymbol{q}}_{\boldsymbol{i+1}}}_{\ddot{\boldsymbol{X}}_{\mathbf{bias,i+1}}} \quad (3.6)$$

This uses the property that $\dot{\boldsymbol{X}} \times \boldsymbol{S}$ is the time derivative of $\boldsymbol{S}$ in a coordinate moving with the body. $\dot{\boldsymbol{X}}\times$ is a spatial cross product operator and has the following 6×6 matrix representation:

$$\dot{\boldsymbol{X}}\times = \begin{pmatrix} \boldsymbol{\omega}\times & 0 \\ \boldsymbol{v}\times & \boldsymbol{\omega}\times \end{pmatrix}, \quad \boldsymbol{\omega}\times = \begin{pmatrix} 0 & -\omega_z & \omega_y \\ \omega_z & 0 & -\omega_x \\ -\omega_y & \omega_x & 0 \end{pmatrix}. \quad (3.7)$$

The computation of Equation (3.6) is possible, when a joint acceleration variable $\ddot{\boldsymbol{q}}$ is known. This is true for the inverse dynamics algorithm, where given the desired motions of the segments, one is to compute the desired forces that cause that motion, $\bar{M}^6 \mapsto \bar{F}^6$. But for the forward dynamics, the input-output causality is reversed and one is to compute the desired motion of the segments, when the desired forces are given, $\bar{F}^6 \mapsto \bar{M}^6$. Therefore, the outward sweep in forward dynamics computes only bias acceleration component of the complete acceleration twist. This difference is underlined in Algorithms 3.1 and 3.2.

3.3.2 Inward force and inertia recursions

The kinematic expression for the acceleration recursions in the paragraphs above deal with the calculation of the acceleration of segment $i+1$ if the pose, velocity and acceleration of the previous segment i are given, together with the acceleration generated at the joint between both segments. The acceleration recursion in the paragraphs below deals with *inertia*-dependent acceleration generated by torques on the joint axes, or external forces, represented by the ordered vector pair $\boldsymbol{F} = (\boldsymbol{\tau}, \boldsymbol{f})$, applied to the segments. Here, $\boldsymbol{\tau}$ and $\boldsymbol{f}$ are an angular moment and a linear force vectors. According to Hamiltonian mechanics, one can express the net force acting on the body as an instantaneous rate of

change of momentum

$$\boldsymbol{F}^b = \frac{d(\overbrace{\boldsymbol{H}\cdot\dot{\boldsymbol{X}}}^{\mathbf{p}})}{dt} = \boldsymbol{H}\ddot{\boldsymbol{X}} + \underbrace{\dot{\boldsymbol{X}}\times^*\boldsymbol{H}\dot{\boldsymbol{X}}}_{\boldsymbol{F}^{\mathbf{b}}_{\mathbf{bias}}} \text{ with } \dot{\boldsymbol{X}}\times^* = -(\dot{\boldsymbol{X}}\times)^T. \tag{3.8}$$

The left most expression arises because of the fact that the differentiation is done in a moving coordinate frame and $p \in \bar{F}^6$ and frequently referred to as a bias force, $\boldsymbol{F}^b_{bias}$. This Equation (3.8) computes the forces on each segment that result in accelerations $\ddot{\boldsymbol{X}}_{i+1}$ in Equation (3.6). This computation is often performed during outward sweep, since all required quantities are already available during each recursion. The equations so far defined relationships between the motion and force space quantities in a kinematic chain and do not account for the outside influences on the motion, such as external forces, $\boldsymbol{F}^{ext}$ and forces from the neighbouring connected bodies. Thus, the balance equation of the net forces on the segment is given as

$$\boldsymbol{F}_{i+1} + \underbrace{({}^{i+1}\boldsymbol{T}\boldsymbol{f}_0)\boldsymbol{F}^{ext}_0}_{\substack{\text{external}\\ \text{forces}}} = \boldsymbol{F}^b_{i+1} + \underbrace{\sum({}^{i+1}\boldsymbol{T}\boldsymbol{f}_k)\boldsymbol{F}_k}_{\substack{\text{forces transmitted}\\ \text{to children segments}}} \tag{3.9}$$

Equation (3.9) follows from the fact that a body in a kinematic chain has a single parent and may have multiple children segments. Because of the joint connections some of forces from these segments are transmitted over the joints to this body. The joint efforts are compensating the net force on each body, a fact that is used in defining the balance equation in inward sweep. Equations (3.8) and (3.9) are valid for the computations of $\bar{M}^6 \mapsto \bar{F}^6$. Furthermore, they follow from the fact that the instantaneous change in rigid body's momentum is caused by the net forces on this body and its spatial inertia matrix $\boldsymbol{H}$ defines the mapping between the motion domain quantity $\dot{\boldsymbol{X}}$ (spatial velocity) and force domain quantity $\boldsymbol{p}$ (spatial momentum). Forward dynamics computes $\bar{F}^6 \mapsto \bar{M}^6$. Since the body motions in the form $\ddot{\boldsymbol{X}}$ are not given, $\boldsymbol{F}^b_i$ cannot be computed. The following briefly describes how the motions are computed from the given segment forces. The complete derivation can be found in Chapter 7 of [Featherstone, 2008].

The formulas below assume that the acceleration starts from *standstill* of the segments. Both components of the acceleration have to be added to find the real physical acceleration of joints and segments. The indices "1" and "2" are used instead of "i" and "$i+1$", just for the sake of reducing the space required to write down some long equations.

The relationship between the acceleration $\ddot{\boldsymbol{X}}_1$ and the force $\boldsymbol{F}$ of segment 1 for an unconstrained segment 1 is given by the segment's spatial inertia matrix

$\boldsymbol{H}_1$. However, if segment 1 is connected to segment 2 by means of an ideal, *non-actuated* joint, the force $\boldsymbol{F}$ is not completely available to accelerate segment 1 because it has also to accelerate segment 2, through the connecting joint constraint. This is depicted in Figure 2.2(b) and Table 2.1. Here segment 1 is the reference or parent segment and segment 2 is the target or child segment. When $\boldsymbol{F}$ is applied on segment 1, part of it $\boldsymbol{F}_1$ accelerates segment 1 by an amount $\ddot{\boldsymbol{X}}_1$ and the rest is transmitted over the connecting joint to segment 2 and accelerates it by $\ddot{\boldsymbol{X}}_2$.

Semantically, a relative motion of two segments expressed in terms of their accelerations is as described in Section 2.4.2. But expressed using motion subspace matrix $\boldsymbol{S}$ and inertia of the segments is given by

$$\ddot{\boldsymbol{X}}_2 = \underbrace{\left(\mathbf{1} - \boldsymbol{S}\left(\boldsymbol{S}^T\boldsymbol{H}_2\boldsymbol{S}\right)^{-1}\boldsymbol{S}^T\boldsymbol{H}_2\right)}_{\boldsymbol{P}_2^T}\ddot{\boldsymbol{X}}_1 + \boldsymbol{S}_2\ddot{\boldsymbol{q}}_2 + \overbrace{\ddot{\boldsymbol{X}}_{bias,2}}^{\text{Equation (3.6)}}, \quad (3.10)$$

$$\text{with } \boldsymbol{P}_2 = \mathbf{1} - \boldsymbol{H}_2\boldsymbol{S}\left(\boldsymbol{S}^T\boldsymbol{H}_2\boldsymbol{S}\right)^{-1}\boldsymbol{S}^T, \quad (3.11)$$

and $\mathbf{1}$ the 6×6 unit matrix. Since $\boldsymbol{P}_2\boldsymbol{P}_2 = \boldsymbol{P}_2$, this matrix is a *projection*, and, hence, *positive semi-definite*. The forces on each segment are given as

$$\begin{aligned} &\boldsymbol{F}_1 = \boldsymbol{H}_1\ddot{\boldsymbol{X}}_1 + \boldsymbol{F}_{bias,1} \text{ and } \boldsymbol{F}_2 = \boldsymbol{H}_2\ddot{\boldsymbol{X}}_2 + \boldsymbol{F}_{bias,2} \\ &\boldsymbol{F} = \boldsymbol{F}_1 + \boldsymbol{F}_2 = \boldsymbol{H}^A\ddot{\boldsymbol{X}}_1 + \boldsymbol{F}^A_{bias} \end{aligned} \quad (3.12)$$

$$\boldsymbol{H}^A = \boldsymbol{H}_1 + \underbrace{\boldsymbol{H}_2 - \boldsymbol{H}_2\boldsymbol{S}\left(\boldsymbol{S}^T\boldsymbol{H}_2\boldsymbol{S}\right)^{-1}\boldsymbol{S}^T\boldsymbol{H}_2}_{\boldsymbol{H}^a_2 = \boldsymbol{P}_2\boldsymbol{H}_2}. \quad (3.13)$$

$\boldsymbol{H}^A$ is the so-called *articulated body inertia* and $\boldsymbol{H}^a_2$ is apparent inertia of a handle connected to segment 1 and supporting segment 2 [Featherstone, 2008]. The articulated body inertia is the same as spatial inertia, if the articulated body consists of a single rigid body. Thus, without loss of generality and considering that each segment is itself an articulated body Equation (3.13) can be re-written in body coordinates as

$$\boldsymbol{H}^a_{i+1} = \underbrace{\left(\boldsymbol{H}^A_{i+1} - \boldsymbol{H}^A_{i+1}\boldsymbol{S}_{i+1}\left(\boldsymbol{S}^T_{i+1}\boldsymbol{H}^A_{i+1}\boldsymbol{S}_{i+1}\right)^{-1}\boldsymbol{S}^T_{i+1}\boldsymbol{H}^A_{i+1}\right)}_{\boldsymbol{P}^A_{i+1}\boldsymbol{H}^A_{i+1}} \quad (3.14)$$

$$\boldsymbol{H}^A_i = \boldsymbol{H}^A_i + \underbrace{\sum\left({}^i\boldsymbol{T}\boldsymbol{v}_{i+1}\right)^T\boldsymbol{H}^a_{i+1}\left({}^i\boldsymbol{T}\boldsymbol{v}_{i+1}\right)}_{\substack{\text{apparent inertia}\\ \text{of child handles}}}. \quad (3.15)$$

Here, it is important to emphasize that

$$\text{Spatial inertia of a body: } (\dot{X} \in \bar{M}^6) \xmapsto{\boldsymbol{H}} (p \in \bar{F}^6)$$

$$\text{Articulated body inertia: } (\ddot{X} \in \bar{M}^6) \xmapsto{\boldsymbol{H^A}} (F \in \bar{F}^6)$$

The net bias force $\boldsymbol{F}_{bias}$ on the articulated system in terms of bias forces on each segment is:

$$\begin{aligned} \boldsymbol{F}^A_{bias} = \boldsymbol{F}_{bias,1} + \underbrace{\boldsymbol{F}_{bias,2} - \boldsymbol{H}_2\boldsymbol{S}(\boldsymbol{S}^T\boldsymbol{H}_2\boldsymbol{S})^{-1}\boldsymbol{S}^T\boldsymbol{F}_{bias,2}}_{\boldsymbol{P_2 F_{bias,2}}} \\ + \boldsymbol{H}_2\boldsymbol{S}(\boldsymbol{S}^T\boldsymbol{H}_2\boldsymbol{S})^{-1}\boldsymbol{\tau}_2 + \boldsymbol{H}^a_2\ddot{\boldsymbol{X}}_{bias,2} \end{aligned} \tag{3.16}$$

Here $\boldsymbol{F}_{bias,2} = \overbrace{\boldsymbol{F}^b_{bias,2}}^{\text{Equation (3.8)}} - \boldsymbol{F}^{ext}_2$. The term involving a joint torque variable $\boldsymbol{\tau}$ in Equation (3.16) follows from the fact that $\boldsymbol{\tau_2} = \boldsymbol{S_2^T F_2}$. So, by dropping 'bias' term in indices the total recursion of articulated body bias forces $\boldsymbol{F}^A_i \leftarrow \boldsymbol{F}^A_{i+1}$ in matrix form is defined by the balance equation for the net forces on the segment:

$$\boldsymbol{F}^a_{i+1} = \boldsymbol{P}^A_{i+1}\boldsymbol{F}^A_{i+1} + \boldsymbol{H}^A_{i+1}\boldsymbol{S}_{i+1}(\boldsymbol{S}^T_{i+1}\boldsymbol{H}^A_{i+1}\boldsymbol{S}_{i+1})^{-1}\boldsymbol{\tau}_{i+1} + \boldsymbol{H}^a_{i+1}\ddot{\boldsymbol{X}}_{\boldsymbol{bias,i+1}} \tag{3.17}$$

$$\boldsymbol{F}^A_i = \boldsymbol{F}^A_i + \underbrace{\sum({}^i\boldsymbol{T}\boldsymbol{f}_{i+1})\boldsymbol{F}^a_{i+i}}_{\substack{\text{apparent forces} \\ \text{from child handles}}} \tag{3.18}$$

with $\boldsymbol{F}^a_{i+1}$ apparent bias force applied by a handle connecting segment i with an articulated body/segment $i+1$ [Featherstone, 2008].

From a computational point of view, the matrix inverses $(\boldsymbol{S}^T\boldsymbol{H}\boldsymbol{S})^{-1}$ in Equations (3.11), (3.14) and (3.17) reduce, in practice, most often to the inversion of just a scalar α—at least under the assumption that every joint has only one degree of freedom—so that an appropriate choice of reference joint attachment pose frame $\{i+1\}$ results in the 6×1 vector $\boldsymbol{S}_{i+1}$ to have only one "1" and zeros everywhere else. Hence, also the matrix-vector products $\boldsymbol{H}\boldsymbol{S}\alpha^{-1}\boldsymbol{S}^T = \alpha^{-1}\boldsymbol{H}\boldsymbol{S}\boldsymbol{S}^T$ can then be found without computations, since it reduces to the selection and scaling of one number from the matrix $\boldsymbol{H}$.

3.3.3 Tree-structured chains

The outward recursions require no changes with respect to serial chains, since every segment and joint is on only one serial path from the root/base of the

tree. When segment i is a branching node in a tree, one must consider the sum over all apparent inertias $\boldsymbol{H}^a_{i+1}$ of the child articulated segments connected to segment i. This derives to Equation (3.15) that uses apparent inertias expressed in parent segment i coordinates (without the transformations the composition semantics of inertias is as presented in Section 2.4.2). The same procedure holds for the apparent bias forces Equation (3.17) that becomes Equation (3.18). By comparing Line 9 with substituted $\ddot{\boldsymbol{X}}$ from Line 6 in Algorithm 3.1 and Line 12 in Algorithm 3.2, it can be observed that their structure is the same, but for different inertias and missing components of $\ddot{\boldsymbol{X}}$ in Algorithm 3.2.

3.3.4 Outward recursion of control torques and segment accelerations

The relationship between the force $\boldsymbol{\tau}_{i+1}$ delivered at a joint and the acceleration $\ddot{\boldsymbol{q}}_{i+1}$ of that joint is given by:

$$\boldsymbol{\tau}_{i+1} = (\boldsymbol{S}^T_{i+1}\boldsymbol{H}^A_{i+1}\boldsymbol{S}_{i+1})\ddot{\boldsymbol{q}}_{i+1} + \boldsymbol{S}^T_{i+1}\Big(\boldsymbol{F}^A_{i+1} + \boldsymbol{H}^A_{i+1}((^{\boldsymbol{i+1}}\boldsymbol{T}\boldsymbol{v}_{\boldsymbol{i}})\ddot{\boldsymbol{X}}_i + \ddot{\boldsymbol{X}}_{\boldsymbol{bias,i+1}})\Big). \tag{3.19}$$

The factor in front of the joint acceleration $\ddot{\boldsymbol{q}}_{i+1}$ is the component on the joint axis of the articulated inertia $\boldsymbol{H}^A_{i+1}$ of all connected child articulated segments of $i+1$. The joint also feels (the component on the joint axis of) the external and Coriolis forces, articulated body bias forces $\boldsymbol{F}^A_{i+1}$ from these child segments, and the inertia force $\boldsymbol{H}^A_{i+1}\ddot{\boldsymbol{X}}_i$ of its own segment's motion. For the forward dynamics, the joint torques $\boldsymbol{\tau}_{i+1}$ are given, and the acceleration at joint $i+1$ follows straightforwardly from Equation ((3.19)):

$$\ddot{\boldsymbol{q}}_{i+1} = (\boldsymbol{S}^T_{i+1}\boldsymbol{H}^A_{i+1}\boldsymbol{S}_{i+1})^{-1}\Bigg(\boldsymbol{\tau}_{i+1} - \boldsymbol{S}^T_{i+1}\Big(\boldsymbol{F}^A_{i+1} + \boldsymbol{H}^A_{i+1}((^{i+1}\boldsymbol{T}\boldsymbol{v}_i)\ddot{\boldsymbol{X}}_i + \ddot{\boldsymbol{X}}_{\boldsymbol{bias,i+1}})\Big)\Bigg). \tag{3.20}$$

For the inverse dynamics, the joint accelerations $\ddot{\boldsymbol{q}}_{i+1}$ are calculated from the (previously computed) inward recursions of the (desired and actual) external forces at the end-effector ($\boldsymbol{F}_N$) and other segments ($\boldsymbol{F}_i$). Gravity is taken into account by initializing the recursion with the gravitational acceleration: $\ddot{\boldsymbol{X}}_0 = \boldsymbol{g}$. Once $\ddot{\boldsymbol{q}}_{i+1}$ is known, the total acceleration of the $i+1$th segment can be found:

$$\ddot{\boldsymbol{X}}_{i+1} = (^{i+1}\boldsymbol{T}\boldsymbol{v}_i)\ddot{\boldsymbol{X}}_i + \ddot{\boldsymbol{q}}_{i+1}\boldsymbol{S}_{i+1} + \ddot{\boldsymbol{X}}_{bias,i+1}, \tag{3.21}$$

with $\ddot{\boldsymbol{X}}_{bias,i+1}$ the bias acceleration due to the non-vanishing angular velocities of the proximal segment. Now, the joint acceleration calculation of the next

joint $\ddot{\boldsymbol{q}}_{i+1}$ can be done, so, the motion of all segments are calculated when this outward recursion reaches the end-effector segments.

Summaries of the algorithms that solve the inverse and forward problems of Equation (3.2) are listed in Algorithm 3.1 and 3.2, respectively. A line by line comparison of the computational sweeps in these listings shows which components of the motions (accelerations are underlined) and forces are computed in particular sweeps. Semantically, both algorithms use the same set of balance equations that define $(\bar{F}^6 \underset{\boldsymbol{H}}{\overset{\boldsymbol{H^{-1}}}{\rightleftharpoons}} \bar{M}^6)$, except for how the dynamic effects of joint constraints are accounted for. Once the inverse and forward problems are solved, a hybrid dynamics problem can be addressed. The hybrid dynamics problem is a general case of both inverse and forward problems and computes either motions or forces of a kinematic chain depending on the input-output causality at a particular joint constraint. It still computes a solution of Equation (3.2) and the kinematic chain is constrained only by the constraint forces internal to the system, i.e. at joint connection.

The next section presents the hybrid dynamics algorithm that can compute motions and forces of a kinematic chain that is under other external constraints in addition to the internal constraints at the joints, i.e. it computes Equation (3.1).

3.4 Popov-Vereshchagin's acceleration-contrained dynamics algorithm

This section introduces the extensions by Popov and Vereshchagin, to the previous computations of the dynamics of a kinematic chain, and applied to the case that *(equality)* constraints are imposed on the acceleration of one or more segments in the chain [Popov et al., 1978, Vereshchagin, 1989]. The constraints can be physical (e.g., contacts with the environment) or virtual (e.g., desired motion of (part of) a frame attached to a segment). The constraint can be partial, i.e., not fully constraining all six degrees of freedom of a segment; for example, only an acceleration constraint in one or two directions of a frame on the segment.

Assume that the segment "N" in a kinematic chain is not allowed to accelerate arbitrarily, but must satisfy the following linear constraints:

$$\boldsymbol{A}_N^T \ddot{\boldsymbol{X}}_N = \boldsymbol{b}_N. \tag{3.22}$$

If there are m constraints, $\boldsymbol{A}_N$ is a $6 \times m$ matrix, and each of its columns can be interpreted as *a direction of constraint force* that must act on

Algorithm 3.1: Inverse Dynamics

Input : Robot geometric, inertial data, $q_i, \dot{q}_i, \ddot{q}_i, \ddot{X}_0, F_i^{ext}$
Output : Robot force, F_i, τ_i

begin
  // Outward sweep of pose, twist and acceleration
  for $i \leftarrow 0$ **to** $N-1$ **do**
    ${}_{i}^{i+1}\boldsymbol{X} = \left({}_{i}^{d_i}\boldsymbol{X}\ {}_{d_i}^{i+1}\boldsymbol{X}(q_i)\right)$;
    $\dot{\boldsymbol{X}}_{i+1} = ({}^{i+1}\boldsymbol{T}\boldsymbol{v}_i)\dot{\boldsymbol{X}}_i + \boldsymbol{S}_{i+1}\dot{q}_{i+1}$;
    $\ddot{\boldsymbol{X}}_{bias,i+1} = \dot{\boldsymbol{X}}_{i+1} \times \boldsymbol{S}_{i+1}\dot{q}_{i+1}$;
    // full acceleration twist is computed
    $\underline{\ddot{\boldsymbol{X}}_{i+1} = ({}^{i+1}\boldsymbol{T}\boldsymbol{v}_i)\ddot{\boldsymbol{X}}_i + \boldsymbol{S}_{i+1}\ddot{q}_{i+1} + \ddot{\boldsymbol{X}}_{bias,i+1}}$;
    $\boldsymbol{F}^b_{bias,i+1} = \dot{\boldsymbol{X}}_{i+1} \times^* \boldsymbol{H}_{i+1}\dot{\boldsymbol{X}}_{i+1}$;
    $\bar{\boldsymbol{F}}^b_{bias,i+1} = \boldsymbol{F}^b_{bias,i+1} - ({}^{i+1}\boldsymbol{T}\boldsymbol{f}_0)^{i+1}\boldsymbol{F}^{ext}_0$;
    $\bar{\boldsymbol{F}}^b_{i+1} = \boldsymbol{H}_{i+1}\underline{\ddot{\boldsymbol{X}}_{i+1}} + \bar{\boldsymbol{F}}^b_{bias,i+1}$;
    $\boldsymbol{F_{i+1}} = \bar{\boldsymbol{F}}^b_{i+1}$;
  // Inward sweep of force
  for $i \leftarrow (N-1)$ **to** 0 **do**
    $\boldsymbol{\tau}_{i+1} = \boldsymbol{S}^T_{i+1}\boldsymbol{F_{i+1}}$;
    $\boldsymbol{F}_i = \boldsymbol{F}_i + \sum({}^i\boldsymbol{T}\boldsymbol{f}_{i+1})\boldsymbol{F}_{i+1}$;

the segment in order to make the chain dynamics satisfy the constraint. The right-hand side $m \times 1$ vector $\boldsymbol{b}_N$ represents the so-called "*acceleration energy*" [Saint Germain, 1900] generated in the constraint; $\boldsymbol{b}_N$ has the physical dimensions of force times acceleration. [Gauss, 1829] introduced this product of a force with an acceleration into classical mechanics, and it was applied by [Gibbs, 1879, Hertz, 1894], and [Appell, 1899] to derive the equations of motions of point masses; [Vereshchagin, 1974] extended it to serial kinematic chains of ideal revolute or prismatic joints. Virtual constraints can represent desired accelerations imposed on the segment by the human programmer; for example:

Algorithm 3.2: Forward Dynamics

Input : Robot geometric, inertial data, $q_i, \dot{q}_i, \tau_i, \ddot{X}_0, F_i^{ext}$
Output : Robot motion, $\ddot{q}_i, \dot{X}_i$

1 **begin**

// Outward sweep of pose, twist and bias component

2 **for** $i \leftarrow 0$ **to** $N-1$ **do**

3 $\quad {}_{i}^{i+1}\boldsymbol{X} = \left({}_{i}^{d_i}\boldsymbol{X}\ {}_{d_i}^{i+1}\boldsymbol{X}(q_i)\right)$;

4 $\quad \dot{\boldsymbol{X}}_{i+1} = ({}^{i+1}\boldsymbol{T}\boldsymbol{v}_i)\dot{\boldsymbol{X}}_i + \boldsymbol{S}_{i+1}\dot{q}_{i+1}$;

5 $\quad \ddot{\boldsymbol{X}}_{bias,i+1} = \dot{\boldsymbol{X}}_{i+1} \times \boldsymbol{S}_{i+1}\dot{q}_{i+1}$;

6 $\quad \boldsymbol{F}^b_{bias,i+1} = \dot{\boldsymbol{X}}_{i+1} \times^* \boldsymbol{H}_{i+1}\dot{\boldsymbol{X}}_{i+1}$;

7 $\quad \bar{\boldsymbol{F}}^b_{bias,i+1} = \boldsymbol{F}^b_{bias,i+1} - ({}^{i+1}\boldsymbol{T}\boldsymbol{f}_0)^{i+1}\boldsymbol{F}^{ext}_0$;

8 $\quad \boldsymbol{H}^A_{i+1} = \boldsymbol{H}_{i+1}$;

9 $\quad \boldsymbol{F}^{\boldsymbol{A}}_{\boldsymbol{i+1}} = \bar{\boldsymbol{F}}^b_{bias,i+1}$;

// Inward sweep of inertia and force

10 **for** $i \leftarrow (N-1)$ **to** 0 **do**

11 $\quad \boldsymbol{D}_{i+1} = \boldsymbol{S}^T_{i+1}\boldsymbol{H}^A_{i+1}\boldsymbol{S}_{i+1}; \quad \boldsymbol{P}^A_{i+1} = \boldsymbol{1} - \boldsymbol{H}^A_{i+1}\boldsymbol{S}_{i+1}\boldsymbol{D}^{-1}\boldsymbol{S}^T_{i+1}$;

12 $\quad \boldsymbol{H}^a_{i+1} = \boldsymbol{P}^A_{i+1}\boldsymbol{H}^A_{i+1}$;

13 $\quad \boldsymbol{H}^A_i = \boldsymbol{H}^A_i + \sum ({}^i\boldsymbol{T}\boldsymbol{v}_{i+1})^T\boldsymbol{H}^a_{i+1}({}^i\boldsymbol{T}\boldsymbol{v}_{i+1})$;

14 $\quad \boldsymbol{F}^a_{i+1} = \boldsymbol{P}^A_{i+1}\boldsymbol{F}^A_{i+1} + \boldsymbol{H}^A_{i+1}\boldsymbol{S}_{i+1}\boldsymbol{D}^{-1}\boldsymbol{\tau}_{i+1} + \boldsymbol{H}^a_{i+1}\ddot{\boldsymbol{X}}_{\boldsymbol{bias,i+1}}$;

15 $\quad \boldsymbol{F}^A_i = \boldsymbol{F}^A_i + \sum ({}^i\boldsymbol{T}\boldsymbol{f}_{i+1})\boldsymbol{F}^a_{i+i}$;

// Outward sweep of accelerations

16 **for** $i \leftarrow 0$ **to** $N-1$ **do**

17 $\quad \ddot{\boldsymbol{q}}_{i+1} =$
$\quad \boldsymbol{D}^{-1}\left\{\boldsymbol{\tau}_{i+1} - \boldsymbol{S}^T_{i+1}\left(\boldsymbol{F}_{i+1} + \boldsymbol{H}^A_{i+1}\left(({}^{i+1}\boldsymbol{T}\boldsymbol{v}_i)\ddot{\boldsymbol{X}}_i + \ddot{\boldsymbol{X}}_{\boldsymbol{bias,i+1}}\right)\right)\right\}$;

18 $\quad \underline{\ddot{\boldsymbol{X}}_{i+1} = ({}^{i+1}\boldsymbol{T}\boldsymbol{v}_i)\ddot{\boldsymbol{X}}_i + \ddot{\boldsymbol{q}}_{i+1}\boldsymbol{S}_{i+1} + \ddot{\boldsymbol{X}}_{bias,i+1}}$;

- To constrain the motion of a reference point on the segment partially in the vertical direction, the constraint matrices can be chosen as follows:

$$\boldsymbol{A}_N = \begin{pmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 1 & 0 \\ 0 & 1 \\ 0 & 0 \end{pmatrix}, \quad \boldsymbol{b}_N = \begin{pmatrix} 0 \\ 0 \end{pmatrix}. \tag{3.23}$$

The columns of $\boldsymbol{A}_N$ are the directions of constraint forces in the horizontal

X and Y directions, that must keep the acceleration in those directions to zero; the three rows of zeros on the top indicate the absence of acceleration constraints on the rotational degrees of freedom.

- To move the segment vertically without allowing rotations, the constraint matrices represent five constraints:

$$\boldsymbol{A}_N = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}, \quad \boldsymbol{b}_N = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}. \tag{3.24}$$

  That means that the constraining forces and moments are allowed to work in all directions, except the vertical Z direction.

- Here is the traditional case of giving the segment a desired acceleration $\ddot{\boldsymbol{X}}_N$ in full 6D:

$$\boldsymbol{A}_N = 1_{6\times 6}, \quad \boldsymbol{b}_N = \ddot{\boldsymbol{X}}_N. \tag{3.25}$$

Remember that the physical dimensions of $\boldsymbol{b}_N$ are not those of 6D acceleration, but of acceleration times force. In the last case, the constraint specification is non-conservative (i.e., the constraint forces do produce work (acceleration energy) against the constraints), by giving an acceleration energy setpoint via a non-zero $\boldsymbol{b}_N$ The m real constraint forces are not known in advance, but are computed by an extension to the hybrid dynamics algorithm. The formulation of the constrained dynamics problem that is computed by the constrained hybrid dynamics solver is given as

$$\begin{cases} \boldsymbol{H}\ddot{\boldsymbol{X}} + \boldsymbol{F_{bias}} + \boldsymbol{F_c} = \boldsymbol{F} \\ \boldsymbol{A}_N^T \ddot{\boldsymbol{X}}_N = \boldsymbol{b}_N, \\ \text{with } \boldsymbol{F}_c = \boldsymbol{A}_N \boldsymbol{\nu} \end{cases} \tag{3.26}$$

This is equivalent to Equation (3.1) expressed in generalized coordinates.

3.4.1 Integrating acceleration constraint into the dynamics recursions

From the acceleration constraint specification presented in the previous Section, the working directions of all constraint forces are known (these are the

"unit" forces contained in the matrices $\boldsymbol{A}_N$), but not their $m \times 1$ vector of magnitudes $\boldsymbol{\nu}$. So, the real constraint forces will be $\boldsymbol{A}_N\boldsymbol{\nu}$. The introduction of segment acceleration constraints leads to the Popov-Vereshchagin extensions to the traditional recursive formulas of Section 3.3.4, [Vereshchagin, 1989]. The approach is based on dynamic programming that is used to search for a local minimum of Lagrange function given as

$$\begin{aligned}\boldsymbol{\mathcal{Z}}_i(\ddot{\boldsymbol{X}}_i, \boldsymbol{\nu}) = \min_{\ddot{\boldsymbol{q}}_i} \Big\{ \sum_{i=0}^{N} \frac{1}{2} (\ddot{\boldsymbol{X}}_{\boldsymbol{i}} - \ddot{\ddot{\boldsymbol{X}}}_{\boldsymbol{i}})^T \boldsymbol{H}_i (\ddot{\boldsymbol{X}}_{\boldsymbol{i}} - \ddot{\ddot{\boldsymbol{X}}}_{\boldsymbol{i}}) + \\ + \sum_{i=1}^{N} \frac{1}{2} (\boldsymbol{d}_i \ddot{\boldsymbol{q}}_i^2) - \boldsymbol{\tau}_i \ddot{\boldsymbol{q}}_i + \boldsymbol{\nu}^T \boldsymbol{A}_N^T \ddot{\boldsymbol{X}}_N \Big\},\end{aligned} \tag{3.27}$$

where $\ddot{\ddot{\boldsymbol{X}}}$ is the acceleration of the system when it is not under constraint and $\boldsymbol{d}$ is inertia of a transmission unit connecting a joint to a motor, Section 2.4.2. The quantity $\boldsymbol{\mathcal{Z}}^{\dagger}$ is called *Zwang* or degree of constraint [Gauss, 1829]. According to Bellman's optimality principle [Bellman, 1957, Bertsekas, 1995], Equation (3.27) can be re-written as in

$$\begin{aligned}\boldsymbol{\mathcal{Z}}_{i-1}(\ddot{\boldsymbol{X}}_{i-1}, \boldsymbol{\nu}) = \min_{\ddot{\boldsymbol{q}}_i} \Big\{ \frac{1}{2} (\ddot{\boldsymbol{X}}_{\boldsymbol{i-1}} - \ddot{\ddot{\boldsymbol{X}}}_{\boldsymbol{i-1}})^T \boldsymbol{H}_{i-1} (\ddot{\boldsymbol{X}}_{\boldsymbol{i-1}} - \ddot{\ddot{\boldsymbol{X}}}_{\boldsymbol{i-1}}) + \\ + \frac{1}{2} (\boldsymbol{d}_i \ddot{\boldsymbol{q}}_i^2) - \boldsymbol{\tau}_i \ddot{\boldsymbol{q}}_i + \boldsymbol{\mathcal{Z}}_i(\ddot{\boldsymbol{X}}_i, \boldsymbol{\nu}) \Big\},\end{aligned} \tag{3.28}$$

where the last term $\boldsymbol{\mathcal{Z}}_i(\ddot{\boldsymbol{X}}_i, \boldsymbol{\nu})$ is searched in a form of a function that is quadratic in $\ddot{\boldsymbol{X}}_{\boldsymbol{i}}$ and $\boldsymbol{\nu}$ [Popov et al., 1978] and given as

$$\begin{aligned}\boldsymbol{\mathcal{Z}}_i(\ddot{\boldsymbol{X}}_i, \boldsymbol{\nu}) = \frac{1}{2} (\ddot{\boldsymbol{X}}_{\boldsymbol{i}} - \ddot{\ddot{\boldsymbol{X}}}_{\boldsymbol{i}})^T \boldsymbol{H}_i (\ddot{\boldsymbol{X}}_{\boldsymbol{i}} - \ddot{\ddot{\boldsymbol{X}}}_{\boldsymbol{i}}) + \frac{1}{2} (\boldsymbol{\nu}^T \boldsymbol{\mathcal{L}} \boldsymbol{\nu}) + \\ + \boldsymbol{\nu}^T \boldsymbol{A}_i^T (\ddot{\boldsymbol{X}}_{\boldsymbol{i}} - \ddot{\ddot{\boldsymbol{X}}}_{\boldsymbol{i}}) + \boldsymbol{F}_i^T (\ddot{\boldsymbol{X}}_{\boldsymbol{i}} - \ddot{\ddot{\boldsymbol{X}}}_{\boldsymbol{i}}) + \boldsymbol{\nu}^T \boldsymbol{U}_i + \bar{\boldsymbol{\alpha}}_i,\end{aligned} \tag{3.29}$$

where $\bar{\boldsymbol{\alpha}}$ is a scalar and $\bar{\boldsymbol{\alpha}}_N = 0$, $\boldsymbol{U}$ is a vector of size $m \times 1$ and $\boldsymbol{U}_N = \boldsymbol{A}_N^T \ddot{\ddot{\boldsymbol{X}}}_N$, $\boldsymbol{\mathcal{L}}$ is a symmetric matrix of size $m \times m$ and has units of $\boldsymbol{b}_N$ with $\boldsymbol{\mathcal{L}}_N = 0$. By first substituting Equation (3.29) for $\boldsymbol{\mathcal{Z}}_i(\ddot{\boldsymbol{X}}_i, \boldsymbol{\nu})$ in Equation (3.28) and minimizing the final expression with respect to $\ddot{\boldsymbol{q}}_i$ and solving for $\ddot{\boldsymbol{q}}_i$ gives the accelerations of constrained motion. This is given as

$$\ddot{\boldsymbol{q}}_i = \boldsymbol{D}_i^{-1} \left\{ \boldsymbol{\tau}_i - \boldsymbol{S}_i^T \left(\boldsymbol{F}_{bias,i} + \boldsymbol{H}_i (\ddot{\boldsymbol{X}}_{i-1} + \ddot{\boldsymbol{X}}_{\boldsymbol{bias,i}}) + \boldsymbol{A}_i \boldsymbol{\nu} \right) \right\}, \tag{3.30}$$

†"*Zwang*" is the German term for "constraint", as used in the early publications in the domain of constrained dynamics that appeared in the dominant German scientific literature of the early 20th century [Hertz, 1984], so the symbol $\boldsymbol{\mathcal{Z}}$ is commonly used in the international literature to denote degree of being constrained: The first English translation of this German and Russian literature introduced the term "acceleration energy", although strictly speaking it is not energy [Vereshchagin, 1974].

where $\boldsymbol{D}_i = \boldsymbol{d}_i + \boldsymbol{S}_i^T \boldsymbol{H}_i \boldsymbol{S}_i$. By substituting $\ddot{\boldsymbol{q}}_i$ back in Equation (3.27) and searching for $\boldsymbol{\mathcal{Z}}_i(\ddot{\boldsymbol{X}}_i, \boldsymbol{\nu})$ in the form of Equation (3.29) gives rise to the following recursive expressions for the coefficients in Equation (3.29):

$$\boldsymbol{H}_i = \boldsymbol{H}_i + ({}^i\boldsymbol{T}\boldsymbol{v}_{i+1})^T \Big(\boldsymbol{H}_{i+1} - \boldsymbol{H}_{i+1}\boldsymbol{S}_{i+1}\boldsymbol{D}^{-1}\boldsymbol{S}_{i+1}^T\boldsymbol{H}_{i+1}\Big)({}^i\boldsymbol{T}\boldsymbol{v}_{i+1}) \tag{3.31}$$

and $\boldsymbol{H}_N = \boldsymbol{H}_N$.

$$\boldsymbol{F}_i = ({}^i\boldsymbol{T}\boldsymbol{v}_{i+1}^T)\Big\{\boldsymbol{F}_{i+1} + \boldsymbol{H}_{i+1}\Big(\ddot{\boldsymbol{X}}_{\boldsymbol{bias,i+1}} + \boldsymbol{S}_{i+1}\boldsymbol{D}_{i+1}^{-1} \Big(\boldsymbol{\tau}_{i+1} - \boldsymbol{S}_{i+1}^T(\boldsymbol{F}_{i+1} + \boldsymbol{H}_{i+1}\ddot{\boldsymbol{X}}_{\boldsymbol{bias,i+1}})\Big)\Big)\Big\} \text{ and } \boldsymbol{F}_N = \boldsymbol{0}. \tag{3.32}$$

$$\boldsymbol{A}_i = ({}^i\boldsymbol{T}\boldsymbol{v}_{i+1}^T)\left(\boldsymbol{1} - \boldsymbol{H}_{i+1}\boldsymbol{S}_{i+1}\boldsymbol{D}_{i+1}^{-1}\boldsymbol{S}_{i+1}^T\right)\boldsymbol{A}_{i+1} \text{ and } \boldsymbol{A}_N = \boldsymbol{A}_N. \tag{3.33}$$

Note the striking resemblance of these recursive equations to those presented in Section 3.3.2 and stand for articulated body inertia, Equation (3.31) and articulated body bias forces, Equation (3.32). Furthermore, the structure of Equation (3.31) is analogous to that of the algebraic Riccati equation [Bertsekas, 1995].

When the solution to the minimization problem in Equation (3.28) is to be expressed in terms of the computational sweeps on a kinematic chain, these coefficient recursions are computed during the inward computational sweep as it is in the case of the forward dynamics problem. In addition to them, the inward sweep also contains the recursions of $\boldsymbol{U}_i$ and $\boldsymbol{\mathcal{L}}_i$ that have the following physical interpretations:

- The inward recursion keeps track of how much of the desired constraint acceleration energy $\boldsymbol{b}_N$ is already generated by the (external and inertial) forces, and by the applied joint torques. That amount of acceleration energy need, hence, not be generated anymore by the to-be-computed constraint forces. Its computation is recursively accumulated, from the end-effector down to segment i, in the acceleration energy $\boldsymbol{U}_i$:

$$\boldsymbol{U}_i = \boldsymbol{U}_{i+1} + \boldsymbol{A}_{i+1}^T\Big\{\ddot{\boldsymbol{X}}_{bias,i+1} + \boldsymbol{S}_i\boldsymbol{D}_{i+1}^{-1}\Big(\tau_{i+1} - \boldsymbol{S}_i^T(\boldsymbol{F}_{i+1} + \boldsymbol{H}_{i+1}^a\dot{\boldsymbol{X}}_{bias,i+1})\Big)\Big\}, \tag{3.34}$$

where $\boldsymbol{U}_N = 0$. The terms in the curly braces represent all accelerations that result from (i) the torque applied at joint i, and (ii) inertial forces applied more distally than this joint i.

- The inward recursion also keeps track of how much of the constraint acceleration energy $\boldsymbol{b}_N$ is already generated by each of the virtual "unit" constraint forces $\boldsymbol{A}_N$ themselves, via the $m \times m$ acceleration constraint *coupling matrix* $\boldsymbol{\mathcal{L}}_i$:

$$\boldsymbol{\mathcal{L}}_i = \boldsymbol{\mathcal{L}}_{i+1} - \boldsymbol{A}_{i+1}^T \boldsymbol{S}_{i+1} (\boldsymbol{S}_{i+1}^T \boldsymbol{H}_{i+1} \boldsymbol{S}_{i+1})^{-1} \boldsymbol{S}_{i+1}^T \boldsymbol{A}_{i+1}, \text{ and } \boldsymbol{\mathcal{L}}_N = 0. \tag{3.35}$$

  The jth row of $\boldsymbol{\mathcal{L}}_i$ contains the acceleration energy that the jth unit constraint force has generated (up to now in the recursion) against the accelerations generated by all constraint forces. $\boldsymbol{A}_{i+1}$ is what is felt of the unit constraint *forces*, more distal than the current joint i; the multiplication with $\boldsymbol{S}_{i+1}(\boldsymbol{S}_{i+1}^T \boldsymbol{H}_{i+1} \boldsymbol{S}_{i+1})^{-1} \boldsymbol{S}_{i+1}^T$ results in the acceleration generated at the current joint i by those constraint forces; and the multiplication with $\boldsymbol{A}_{i+1}^T$ yields the acceleration energy contributions at this joint.

Now, in order to find Lagrange multiplier $\boldsymbol{\nu}$, one is to minimize Lagrange function given in Equation (3.27) with respect to $\boldsymbol{\nu}$. As it can be observed the result of the minimization is equal to:

$$\begin{aligned} \boldsymbol{\mathcal{Z}}_i(\ddot{\boldsymbol{X}}_i, \boldsymbol{\nu}) = \min_{\boldsymbol{\nu}} \Big\{ \sum_{i=0}^{N} \frac{1}{2} (\ddot{\boldsymbol{X}}_{\boldsymbol{i}} - \ddot{\boldsymbol{X}}_{\boldsymbol{i}})^T \boldsymbol{H}_i (\ddot{\boldsymbol{X}}_{\boldsymbol{i}} - \ddot{\boldsymbol{X}}_{\boldsymbol{i}}) + \\ + \sum_{i=1}^{N} \frac{1}{2} (\boldsymbol{d}_i \ddot{\boldsymbol{q}}_i^2) - \boldsymbol{\tau}_i \ddot{\boldsymbol{q}}_i + \boldsymbol{\nu}^T \boldsymbol{A}_N^T \ddot{\boldsymbol{X}}_N \Big\} = \boldsymbol{b}_N. \end{aligned} \tag{3.36}$$

Searching for the solution of $\boldsymbol{\mathcal{Z}}_i(\ddot{\boldsymbol{X}}_i, \boldsymbol{\nu})$ in the form of Equation (3.29), the following expression is obtained:

$$\boldsymbol{\mathcal{L}}_i \boldsymbol{\nu} + \boldsymbol{A}_i^T (\ddot{\boldsymbol{X}}_{\boldsymbol{i}} - \ddot{\boldsymbol{X}}_{\boldsymbol{i}}) + \boldsymbol{U}_i = \boldsymbol{b}_N. \tag{3.37}$$

When the recursion arrives at the base ($i = 0$), one can solve for the still unknown constraint force magnitudes $\boldsymbol{\nu}$. For a robot with a rigidly fixed base, the acceleration $\ddot{\boldsymbol{X}}_0$ consists of gravity only and $\ddot{\boldsymbol{X}}_{\boldsymbol{0}} = 0$. If $\boldsymbol{\mathcal{L}}_i$ is not full rank, the imposed end-effector constraint cannot be completely satisfied and the inversion of the coupling matrix needs to be search using damped least squares approach, for instance. Another extreme case there are no any external constraints at all, then the whole computations reduce to Equations (3.31) and

(3.32). Finally, by substituting computed coefficients back in Equation (3.30), the acceleration of the constrained system can be computed. This remains similar to the unconstrained case of Equation (3.20), except that an extra joint torque $\boldsymbol{A}\boldsymbol{\nu}$ is added to generate the desired constraint forces. In summary, the hybrid dynamics algorithm of Popov and Vereshchagin has no formal changes in the equations with respect to the traditional case, Equation ((3.20)), except that, now, the joint torques τ_i are the sum of two contributions: (i) the "inverse dynamics" torque needed to satisfy the Cartesian constraints, and (ii) the "forward dynamics" torque needed for posture control, or any other feedforward objective. Figure 3.3 summarizes the number of sweeps, the quantities that are computed during each sweep and the constraints on the kinematic tree.

3.5 Extensions

This Section explains the various extensions that this research makes to the original Popov-Vereshchagin algorithm.

3.5.1 Multiple constraints in a tree

The recursions of the previous Sections apply to constraints on only one segment of a kinematic chain. Extending this to multiple constraints on (the same or different) segments in a tree-structured kinematic chain does not change the recursions in Equation (3.33) and Equation (3.34) of, respectively, the constraint force matrix $\boldsymbol{A}$ and the accumulated acceleration energy $\boldsymbol{b}$ of each individual constraint. But it involves some extensions for the acceleration energy coupling matrix $\boldsymbol{\mathcal{L}}$ and for the computation of the constraint force magnitudes $\boldsymbol{\nu}$. The following paragraphs explain the extensions for the case of the composition of two constraints, but the procedure generalizes to any number of constraints:

- When a constraint applies on a segment that is not a leaf segment, the recursions can start from that segment, because more distal segments will not contribute to the constraint satisfaction or violation.
- When the inward recursion of a k-dimensional constraint c reaches a segment where it must be joined with the recursion of a l-dimensional constraint d, both acceleration constraint coupling matrices $\boldsymbol{\mathcal{L}}^c$ and $\boldsymbol{\mathcal{L}}^d$, Equation (3.35), are fused into a $(k+l) \times (k+l)$-dimensional constraint matrix $\boldsymbol{\mathcal{L}}^{cd}$:

$$\boldsymbol{\mathcal{L}}^{cd} = \begin{pmatrix} \boldsymbol{\mathcal{L}}^c & 0 \\ 0 & \boldsymbol{\mathcal{L}}^d \end{pmatrix}. \tag{3.38}$$

Algorithm 3.3: Constrained Hybrid Dynamics

Input : Robot geometric, inertial data, $q_i, \dot{q}_i, \tau_i, \ddot{X}_0, F_i^{ext}, A_N, b_N$
Output : Robot motion, $\ddot{q}_i, \ddot{X}_i$

begin

// Outward sweep of pose, twist and bias component

for $i \leftarrow 0$ **to** $N-1$ **do**

$${}^{i+1}_{i}\boldsymbol{X} = \left({}^{d_i}_{i}\boldsymbol{X}\ {}^{i+1}_{d_i}\boldsymbol{X}(q_i)\right);$$

$$\dot{\boldsymbol{X}}_{i+1} = ({}^{i+1}\boldsymbol{T}\boldsymbol{v}_i)\dot{\boldsymbol{X}}_i + \boldsymbol{S}_{i+1}\dot{q}_{i+1};$$

$$\ddot{\boldsymbol{X}}_{bias,i+1} = \dot{\boldsymbol{X}}_{i+1} \times \boldsymbol{S}_{i+1}\dot{q}_{i+1};$$

$$\boldsymbol{F}^b_{bias,i+1} = \dot{\boldsymbol{X}}_{i+1} \times^* \boldsymbol{H}_{i+1}\dot{\boldsymbol{X}}_{i+1};$$

$$\bar{\boldsymbol{F}}^b_{bias,i+1} = \boldsymbol{F}^b_{bias,i+1} - ({}^{i+1}\boldsymbol{T}\boldsymbol{f}_0)^{i+1}\boldsymbol{F}^{ext}_0;$$

$$\boldsymbol{H}^A_{i+1} = \boldsymbol{H}_{i+1};$$

$$\boldsymbol{F}^{\boldsymbol{A}}_{\boldsymbol{i+1}} = \bar{\boldsymbol{F}}^b_{bias,i+1};$$

// Inward sweep of inertia and force

for $i \leftarrow (N-1)$ **to** 0 **do**

$$\boldsymbol{D}_{i+1} = \boldsymbol{d}_{i+1} + \boldsymbol{S}^T_{i+1}\boldsymbol{H}^A_{i+1}\boldsymbol{S}_{i+1}; \quad \boldsymbol{P}^A_{i+1} = \boldsymbol{1} - \boldsymbol{H}^A_{i+1}\boldsymbol{S}_{i+1}\boldsymbol{D}^{-1}_{i+1}\boldsymbol{S}^T_{i+1}\ ;$$

$$\boldsymbol{H}^a_{i+1} = \boldsymbol{P}^A_{i+1}\boldsymbol{H}^A_{i+1}; \quad \boldsymbol{H}^A_i = \boldsymbol{H}^A_i + \sum ({}^i\boldsymbol{T}\boldsymbol{v}_{i+1})^T\boldsymbol{H}^a_{i+1}({}^i\boldsymbol{T}\boldsymbol{v}_{i+1});$$

$$\boldsymbol{F}^a_{i+1} = \boldsymbol{P}^A_{i+1}\boldsymbol{F}^A_{i+1} + \boldsymbol{H}^A_{i+1}\boldsymbol{S}_{i+1}\boldsymbol{D}^{-1}_{i+1}\boldsymbol{\tau}_{i+1} + \boldsymbol{H}^a_{i+1}\ddot{\boldsymbol{X}}_{\boldsymbol{bias,i+1}};$$

$$\boldsymbol{F}^A_i = \boldsymbol{F}^A_i + \sum({}^i\boldsymbol{T}\boldsymbol{f}_{i+1})\boldsymbol{F}^a_{i+i}; \quad \boldsymbol{A}_i = ({}^i\boldsymbol{T}\boldsymbol{v}^T_{i+1})\boldsymbol{P}_{i+1}\boldsymbol{A}_{i+1}\ ;$$

$$\boldsymbol{U}_i = \boldsymbol{U}_{i+1} + \boldsymbol{A}^T_{i+1}\left\{\ddot{\boldsymbol{X}}_{bias,i+1} + \boldsymbol{S}_i D^{-1}\left(\tau_{i+1} - \boldsymbol{S}^T_i(\boldsymbol{F}_{i+1} + \boldsymbol{H}^a_{i+1}\ddot{\boldsymbol{X}}_{bias,i+1})\right)\right\};$$

$$\boldsymbol{\mathcal{L}}_i = \boldsymbol{\mathcal{L}}_{i+1} - \boldsymbol{A}^T_{i+1}\boldsymbol{S}_{i+1}D^{-1}_{i+1}\boldsymbol{S}^T_{i+1}\boldsymbol{A}_{i+1}, \quad \boldsymbol{\mathcal{L}}_N = 0\ ;$$

$$\boldsymbol{\mathcal{L}}_0\,\boldsymbol{\nu} + \boldsymbol{A}^T_0\ddot{\boldsymbol{X}}_0 + \boldsymbol{U}_0 = \boldsymbol{b}_N\ ;$$

// Outward sweep of accelerations

for $i \leftarrow 0$ **to** $N-1$ **do**

$$\ddot{\boldsymbol{q}}_{i+1} = \boldsymbol{D}^{-1}_{i+1}\left\{\boldsymbol{\tau}_{i+1} - \boldsymbol{S}^T_{i+1}\left(\boldsymbol{F}_{i+1} + \boldsymbol{H}^A_{i+1}(({}^{i+1}\boldsymbol{T}\boldsymbol{v}_i)\ddot{\boldsymbol{X}}_i + \ddot{\boldsymbol{X}}_{\boldsymbol{bias,i+1}}) + \boldsymbol{A}_{i+1}\,\boldsymbol{\nu}\right)\right\};$$

$$\ddot{\boldsymbol{X}}_{i+1} = ({}^{i+1}\boldsymbol{T}\boldsymbol{v}_i)\ddot{\boldsymbol{X}}_i + \ddot{\boldsymbol{q}}_{i+1}\boldsymbol{S}_{i+1} + \ddot{\boldsymbol{X}}_{bias,i+1};$$

This "fusion" is done before going over the first common more proximal joint, hence the recursion of $\boldsymbol{\mathcal{L}}^{cd}$ continues from there on as in Equation (3.35), but now with the $6 \times (k+l)$ constraint force matrix $\boldsymbol{A}^{cd}$:

$$\boldsymbol{A}^{cd} = \begin{pmatrix} \boldsymbol{A}^c & \boldsymbol{A}^d \end{pmatrix}. \tag{3.39}$$

The physical interpretation of $\boldsymbol{\mathcal{L}}^{cd}$ is clear: its jth row contains the acceleration energy that the jth unit constraint force has generated (up to now in the recursion) against the accelerations generated by all $k+l$ constraint forces. Indeed, the extra joint forces needed to realize the constraint forces in c also influence the constraint forces in d, and vice versa.

- At the base, the extended version of Equation (3.37) must be solved:

$$\boldsymbol{\mathcal{L}}_0^{cd} \begin{pmatrix} \boldsymbol{\nu}^c \\ \boldsymbol{\nu}^d \end{pmatrix} = \begin{pmatrix} \boldsymbol{b}_N^c \\ \boldsymbol{b}_N^d \end{pmatrix} - (\boldsymbol{A}_0^{cd})^T \ddot{\boldsymbol{X}}_0. \tag{3.40}$$

The constraints can be satisfied without conflicts, if $\boldsymbol{\mathcal{L}}_0^{cd}$ is of full rank, and hence it can be inverted. If some of the constraints are conflicting, Equation (3.40) must be solved via a traditional *weighted pseudo-inverse*.

3.5.2 Posture/Null space control

Similar to the concept of the velocity-level "null space" of a kinematic chain [Liegeois, 1977], the *acceleration null space* is defined to be the set of joint *forces* that generate a zero *acceleration* at the end-effector, $\ddot{\boldsymbol{X}}_N = 0$. This is a special case of the constrained end-effector in Equation (3.22):

$$\boldsymbol{A}_N = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}, \quad \boldsymbol{b}_N = 0. \tag{3.41}$$

Applications like *posture control* for a redundant or humanoid robot want to superimpose a null space motion to the outcome of the hybrid dynamics algorithm, to keep the whole robot close to a certain posture, for balancing or obstacle avoidance. Such posture control works by adding joint forces (or accelerations) and/or segment forces, to generate motions of the legs, torso and arms that do not change the desired end-effector motion constraints. The presented recursions deal with this problem as follows:

- If the null space motion is specified as desired accelerations $\ddot{q}_i$ of some of the joints, or *forces* τ_i at some of the joints, these are added to the recursion in Equation (3.34), possibly with the help of Equation (3.21).

- If the null space motion is specified as desired segment forces $\boldsymbol{F}_i$ on some of the segments, these are added as given forces to the recursion in Equation (3.17).

Both result in "bias accelerations" at the end effector, just like those generated by the Coriolis and centripetal forces. Hence, the later recursions then compute the joint forces that compensate for these bias accelerations, irrespective of their origin.

3.5.3 Novel linear-time solution for task prioritization and constraint weighting

The posture control procedure as outlined in the previous Section does, implicitly, a prioritization of the end-effector constraints over the posture control inputs: the constraints take precedence over the posture control inputs, because the compensating joint forces to realize the desired constraints (exactly, and irrespective of the other inputs) take place *after* the posture control forces are taken into account. (Of course, within the limits of (i) available actuator forces at the joints, and (ii) conflicts of the end-effector acceleration constraints.)

This insight allowed us to change the traditional implicit computational order, and to determine whether to prioritize constraints, joint forces, or external forces, just by switching the *order* of the recursions:

- The constraint vector $\boldsymbol{\nu}$ is computed with *zero* posture control forces (on joints and segments) in Equation (3.34) (but of course with all other external forces taken fully into account);
- During the final outward recursion Equation (3.30) the non-zero posture control forces are added (Figure 3.4).

Of course, variations on this prioritization approach exist by selecting whether or not to include the joint forces or the segment forces into the computational scheme (Figure 4.3).

Also the complementary approach to prioritization, that is weighting, of the various controls, can be achieved in the following ways by changing the traditional computatonal scheme:

- The actual bias acceleration $\ddot{\boldsymbol{X}}_{bias,i}$ at the end-effector that results from non-zero posture control forces at segments or joints is used to generate the right-hand side acceleration energy vector $\boldsymbol{b}_N^{posture}$ using

Equation (3.22) with the desired end-effector constraint matrix $\boldsymbol{A}_N$. The desired acceleration energy vector $\boldsymbol{b}_N^{ee}$ is *added* to $\boldsymbol{b}_N^{posture}$ as input to the constraint computation recursion (Figure 3.4). In this way, the weighting is achieved via the natural dynamics of the kinematic chain.

- An application-specific weighting can be introduced, by weighting the contributions of $\boldsymbol{b}_N^{ee}$ and $\boldsymbol{b}_N^{posture}$ before starting the recursion.

$$\boldsymbol{b}_{control} = w_{posture}\boldsymbol{b}_N^{posture} + w_{ee}\boldsymbol{b}_N^{ee}. \tag{3.42}$$

Finally, note that all the presented prioritization or weighting schema can still be achieved via the linear time recursions, without having to resort to the higher-order null space projection matrices as in, for example, [Khatib et al., 2008].

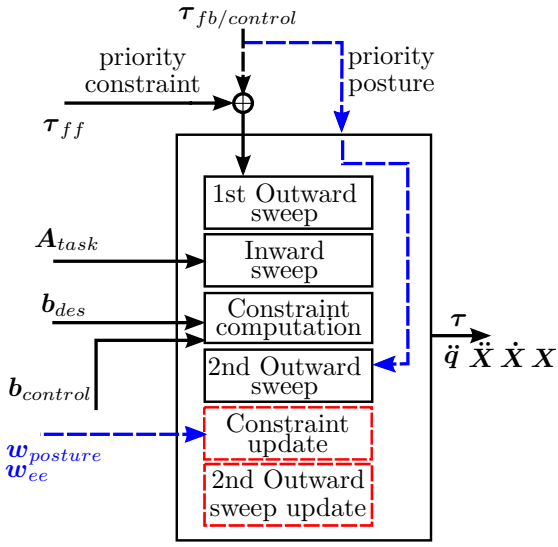


Figure 3.4: The figure depicts feedforward and feedback inputs to the recursions. Such decomposition of the torques and acceleration energy contributions allows implementation of priority and weighting-based task control.

3.5.4 Unilateral and maximum joint torque constraints

During the final outward recursion of Equation (3.30), one checks whether the requested total joint torque τ_i (that is, the sum of all the term inside the curly

braces) exceeds the maximum available torque $\tau_i^{\max}$. If that is the case, the algorithm is adapted in the following way:

- The total applied torque is set to $\tau_i^{\max}$ in Equation (3.30), and the recursion continues.
- When reaching the end of the recursion, one computes to what extent the local acceleration constraint is violated, via the acceleration energy formula for $\boldsymbol{b}_N^n$ in Equation (3.22).
- This new acceleration energy value $\boldsymbol{b}_N^n$ is used in Equation (3.37) to compute new constraint force magnitude values $\boldsymbol{\nu}^n$.
- Repeat the procedure until all specified constraint violation tolerances are satisfied, or no solution is possible.

The worst-case computational complexity occurs when (i) one wants to redo the computations to try to satisfy the constraints anyway despite the joint torque saturation, and (ii) the last joint N is the first one to saturate, and then joint $N-1$ in the next recursion, and soon. This involves a cost of $\mathcal{O}(N^2)$.

Chapter 5 presents a setup with the constraints on maximum joint torques. There the controller that stabilizes the constraints uses adaptive gains which are computed using a predictive greedy algorithm.

3.5.5 Numerical considerations

The matrix $\boldsymbol{\mathcal{L}}_0$ in Equation (3.37) is symmetric—as are all its predecessors $\boldsymbol{\mathcal{L}}_i$—as is apparent from Equation (3.35). Since each recursion in Equation (3.35) adds a *matrix of rank one*, and starts with a zero matrix, a minimum of m joint degrees of freedom are needed for the invertibility of $\boldsymbol{\mathcal{L}}_0$, and hence to generate the m constraint forces. Even then, $\boldsymbol{\mathcal{L}}_0$ can be *singular*, which is the mathematical indication that the kinematic chain is physically unable to generate the desired constraint of Equation (3.22). Another reason for singularity of $\boldsymbol{\mathcal{L}}_0$ is that the different constraints are not independent, sometimes also called "conflicting".

In both cases of singularity, a (weighted) pseudo-inverse solution can provide a set of joint forces that approximates the desired acceleration constraints, by weighting their relative importance. The weighting takes place in the space of the constraint magnitudes $\boldsymbol{\nu}$, or in the end-effector accelerations $\ddot{\boldsymbol{X}}_N$. This is different from the "inertia weighting" that takes place in the recursions on the mechanical dynamics, e.g., in Equation (3.11), [Whitney, 1969].

The numerical complexity of computing such weighted-inverse for $\boldsymbol{\mathcal{L}}_0$ is typically $\mathcal{O}(m^3)$, with m the number of constraints. But since the recursion in Equation (3.35) starts with a zero matrix, and each time adds a rank -1 symmetric update, linear time rank-revealing and decomposition solutions exist for this case, as described in [Sentana, 1996]. The latter solution is recursive, and fits perfectly to the use case of segment-by-segment recursions over 1-DOF joints. Hence, when arriving at the root segment, the presented numerical decomposition approach allows to immediately solve Equation (3.37) for the constraint force magnitudes $\boldsymbol{\nu}$, because the matrix $\boldsymbol{\mathcal{L}}_0$ being already available in a decomposed form that fits well to solving a linear set of equations with optimal numerical conditioning and efficiency.

3.6 Discussions and conclusions

This chapter brought the original work by Russian roboticists Popov and Vereshchagin [Popov et al., 1978, Vereshchagin, 1989] into the context of complex task specification and control for modern robotic systems. Popov-Vereshchagin algorithm provides a fully recursive, linear-time, inverse and forward dynamics algorithm with partial acceleration constraints. The original paper [Vereshchagin, 1989] also briefly mentions some further extensions, that were not included in this research: the inclusion of joint shaft model, the case of a free-floating base with general acceleration constraints on the base. Chapter introduced inverse, forward and hybrid constrained dynamics problems step-by-step. It has shown that the semantics of the motion and force computations is essentially the same and that the differences follow from the input-output causality defined on the mapping between motion and force space constraints.

The algorithms that solve the presented dynamics problems are described in terms of recursive computations, also known as computational sweeps, on a kinematic structure. During the sweeps either motion or force space variables associated with the kinematic mechanism are updated. It is shown that the inverse problem can be computed by the inverse dynamics algorithm, Algorithm 3.1 that consists of two computational sweeps: an outward sweep of motion space quantities and an inward sweep of force space quantities. The forward problem can be computed by the forward dynamics algorithm, Algorithm 3.2 that consists of three computational sweeps: an outward sweep of motion variables, an inward sweep of forces and inertias and the second outward sweep to compute the remaining motion variables that result from the forces. In all cases, it is important to emphasize the role of the rigid body inertia tensor that plays a role of a mapping operator between motion and force spaces. Depending on the type of the dynamics problem the mapping takes

place either between the momentum and the velocity in the inverse problem or between the force and the acceleration in the forward problem.

The chapter presented Popov-Vereshchagin algorithm in terms of the computational sweeps described in the context of the inverse and forward dynamics problems. Popov-Vereshchagin algorithm computes the motions and forces associated with the kinematic chain which is under external acceleration level equality constraints. It approaches the solution to this constrained problem by reformulating the original Equation 3.26 as a dynamic programming problem of minimization of Lagrangian, Equation 3.27. In mechanics, such a problem is known as Gauss's principle of least constraint that in its simplified form states that any motion of the constrained system deviates from its unconstrained motion as little as possible.

Furthermore, the chapter presented extensions to the algorithm to the case of tree-structured robots, with multiple "task frames", so that it becomes applicable to modern humanoid robots and mobile manipulators. The extensions are based on the insight that the computational sweeps are essentially a set of computations that are scheduled sequentially and that this schedule is determined by the structure of the kinematic chain whose dynamics is computed. This insight allows to compute many variants of the generic hybrid dynamics (forward, inverse, constraints, weighting, prioritization, torque limits,...) by simple changes in the scheduling of the computational sweeps and contributions of different quantities during each sweep. The result is a generic/meta algorithm that is simple to configure for one or more of the desired dynamics computations with a very limited extra cost of computations. This allows an implementation of different task control schema such as prioritization and weighting-based task control. The implementations do not involve conventional nullspace projection-based technique. It relies on the decoupling of the commanded inputs, e.g., torque or the acceleration energy contributions, and feeding them component-wise at different points of the computational sweeps. Such structural and behavioral re-organization of the algorithm allows one to combine it with other exotic control algorithms, such as the one discussed in detail in Section 5.4 that is used to predict and to adapt gains of the constraint controllers.

The implementation of the constrained dynamics solver is available as part of the OROCOS KDL [Smits et al., 2001] project. The latest tested version that also partially uses geometric relations semantics library [De Laet and Bellens, 2012] is available at [Shakhimardanov and Bruyninckx, 2014a]. The implementation follows the same guidelines presented in Section 2.5, i.e., the algorithms are expressed as the recursive sweeps that apply a set of composite operations on a kinematic chain according to some traversal policy. Most of the existing implementation is tested for serial kinematic chains and currently can only cope with the constraints on the end-effector segment. Therefore, all simulations and

examples in following chapters are performed on setups involving serial kinematic chains with the end-effector constraints. Some other proposed theoretical extensions such as the one in Section 3.5.4 are not implemented yet.

Chapter 4

Role of Popov-Vereshchagin Solver in Control

The goal of this chapter is twofold. First, it discusses how the constrained hybrid dynamics presented in Chapter 3 can be used in a robot control application. The examples and approaches presented in this context do not develop controllers that exhibit perfect behavior. The main focus lies in analyzing how the dynamics solver is integrated in a control loop and interacts with different existing control laws, and where a control specification itself fits in the motion programming stack, Figure 4.1.

The second complementary matter that is addressed by this chapter is a validation of the *implementation* of the algorithm and its extensions, Section 3.5.3, via extensive simulation results on some relevant use cases, Section 4.2.

4.1 Related work

Chapter 3 presented various domain specific solvers that can be used to compute dynamic motions and interactions of a robotic mechanism. First, it presented the forward and inverse dynamics solvers that compute an *unconstrained* dynamic motion of the robot given by Equation (3.2) either in generalized or spatial coordinates.

Then, the complexity of the problem increased by the introduction of the

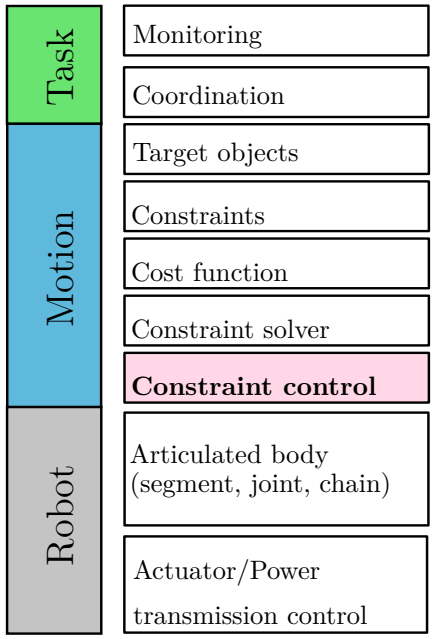


Figure 4.1: Controller specification and its position in the (motion)task programming stack.

external constraints on system's dynamics, and a resultant *constrained problem* had the form that could be expressed using either Equation (1.4) in generalized coordinates with a holonomic position level constraint, or Equation (3.1) in generalized coordinates with an acceleration level constraint, or Equation (3.26) in spatial coordinates with the special case of the acceleration level constraint on robot's end-effector.

Many texts in robotics and in multi-body dynamics do implicitly rely on some form of optimization, in order to solve the constrained dynamics problems above [Udwadia and Kalaba, 2002, Masarati, 2011, Peters et al., 2008]. This is especially observable in redundancy resolution approaches based on the pseudo-inverse of the robot Jacobian matrix. [Whitney, 1969, Liegeois, 1977] and [Hollerbach and Suh, 1987, Luh et al., 1980b] are some of the well-known texts for the velocity- and the acceleration-resolved schema, respectively that use techniques based on the pseudo-inverses. Additionally, many of those contributions do not always draw a line between the solution of the constrained problem and the design requirements for the controllers.

Many constrained kinematics and dynamics solvers, although claim to solve the problem described by Equation (3.1), essentially solve a *constrained optimization* problem as in

$$\begin{cases} \boldsymbol{\mathcal{I}} = \boldsymbol{u}^{\boldsymbol{T}} \cdot \boldsymbol{W}(\boldsymbol{q}, \boldsymbol{u}) \cdot \boldsymbol{u} \\ \boldsymbol{h}(\boldsymbol{q}, \boldsymbol{t}) = \boldsymbol{0} \\ \boldsymbol{M}(\boldsymbol{q})\ddot{\boldsymbol{q}} = \boldsymbol{u}(\boldsymbol{q}, \dot{\boldsymbol{q}}) - \boldsymbol{C}(\boldsymbol{q}, \dot{\boldsymbol{q}}) - \underbrace{\boldsymbol{\tau_c}(\boldsymbol{q})}_{\boldsymbol{J_c^T \lambda}}. \end{cases} \tag{4.1}$$

Here, $\boldsymbol{\mathcal{I}}, \boldsymbol{u}, \boldsymbol{h}, \boldsymbol{W}$ are a cost function, a control input, a holonomic position constraint function, and a positive semi-definite metric weighing matrix, respectively. In this formulation the metric weighing matrix $\boldsymbol{W}$ is often chosen based on the fundamental physical principle, e.g., D'Alembert, Hamiltonian, Gauss, Appell principles, that is used to analyze and solve the problem. For instance, as it was presented in Chapter 3, an equivalent of the formulation in Equation (4.1) is given as in

$$\begin{cases} \boldsymbol{\mathcal{I}}_i(\ddot{\boldsymbol{X}}_i) = \min\limits_{\ddot{\boldsymbol{q}}_i} \Big\{ \sum\limits_{i=0}^{N} \frac{1}{2} (\ddot{\boldsymbol{X}}_{\boldsymbol{i}} - \ddot{\boldsymbol{X}}_{\boldsymbol{i}})^T \boldsymbol{H}_i (\ddot{\boldsymbol{X}}_{\boldsymbol{i}} - \ddot{\boldsymbol{X}}_{\boldsymbol{i}}) \Big\} \\ \boldsymbol{A}_N^T \ddot{\boldsymbol{X}}_N = \boldsymbol{b}_N \\ \boldsymbol{H}\ddot{\boldsymbol{X}} = \boldsymbol{F} - \boldsymbol{F_{bias}} - \underbrace{\boldsymbol{F}_c}_{\boldsymbol{A\nu}} \end{cases} \tag{4.2}$$

and Popov-Vereshchagin constrained hybrid dynamics solver, Algorithm 3.3 computes the solution to this problem. Furthermore, the motion model used in these formulations do not have to involve dynamics, Equation 3.2, but can also rely on the kinematic model as in Equation 1.2. Then, the solution of these kinematic formulations is based on looking for the solution of the least-squares problem [Duffy, 1990, Doty et al., 1993], where the weighing or damping matrix $\boldsymbol{W}$ plays a central role [Smits, 2010].

In [Nakanishi et al., 2008, Peters et al., 2008], the authors present a framework for the robot control. They introduce the framework as an 'umbrella' to explain the relationships between the existing robot control approaches. The authors formulate their approach as an optimization problem that can be expressed as in Equation (4.1) using the principles of motion discussed in [Udwadia and Kalaba, 2002]. The analytic solution of this optimization problem is given as

$$\boldsymbol{u} = \boldsymbol{W^{-1/2}}(\boldsymbol{J_c M^{-1} W^{-1/2}})^{+}(\hat{\boldsymbol{b}} - \boldsymbol{J_c M^{-1} C}), \tag{4.3}$$

where $\boldsymbol{J_c\ddot{q} = \hat{b}(q, \dot{q})}$ is obtained by differentiating the holonomic constraint function $\boldsymbol{h(q, t) = 0}$ upto the acceleration level. Based on this formulation of the input $\boldsymbol{u}$, the authors can express different control approaches by varying $\boldsymbol{W}$ and $\boldsymbol{J_c}$. Here, $\boldsymbol{W}$ determines how the control inputs are distributed over the joints. For instance, if the $\boldsymbol{W}$ metric is to be consistent with the Gauss principle, it is set to $\boldsymbol{M^{-1}}$. This is equivalent to the control approach formulation presented in [Bruyninckx and Khatib, 2000] and can be computed by the constrained solver from Chapter 3. If the robot is to track a Cartesian space trajectory, one obtains the following control law

$$\boldsymbol{u} = \boldsymbol{J}^T \overbrace{(\boldsymbol{JM^{-1}J^T})^{-1}}^{\substack{\text{apparent}\\ \text{inertia}}} \overbrace{\left(\boldsymbol{\ddot{X}_d - K_D(\dot{X} - \dot{X}_d) - K_P(X - X_d) - \dot{J}(q)\dot{q}}\right)}^{\substack{\text{Cartesian acc. resolved}\\ \text{control law, } \bar{\boldsymbol{b}}}} +$$

$$+\boldsymbol{C} + \boldsymbol{M}\underbrace{(\boldsymbol{I} - \overbrace{\boldsymbol{M^{-1}J^T(JM^{-1}J^T)^{-1}}}^{\substack{\text{dynamically consistent}\\ \text{inverse}}}\boldsymbol{J})}_{\substack{\text{dynamically consistent}\\ \text{nullspace}}}\boldsymbol{M^{-1}u_0}, \tag{4.4}$$

with $\boldsymbol{h(q, t) = X(t) - X_d(t) \Rightarrow J_c = J_r = J}$ and $\boldsymbol{u_0}$ is a nullspace vector, $\boldsymbol{\bar{b}(q, \dot{q}) = \ddot{X}_d - K_D(\dot{X} - \dot{X}_d) - K_P(X - X_d) - \dot{J}(q)\dot{q}}$. Although the problem that is solved is the constrained problem of Equation (4.1) and its general solution is as in Equation (4.3), note the structure of $\bar{\boldsymbol{b}}$ in this expression. Strictly taken, this is not true! Because during the differentiation of holonomic constraint $\boldsymbol{h(q, t)}$ to obtain its acceleration level representation, the information on pose and velocity constraints is lost, which causes numerical drift, thus resulting in unstable mathematical representation [Masarati, 2011]! In order to account for these lost constraints, the control should be chosen that stabilizes Equation (4.4). Frequently, this is not exemplified, and just mentioned as a side note by many authors [Hollerbach and Suh, 1987, Luh et al., 1980b, Nakanishi et al., 2007, Nakanishi et al., 2008, Peters et al., 2008]. This can also lead to a false impression that an outcome of the optimization-based control problem is a stable control command.

To summarize, the solvers compute either constrained or unconstrained problems and often rely on optimization-based techniques to compute the former. In order to use them in robot control applications, an additional *stabilizing or controlling* expression is required. In constrained rigid body dynamics literature this kind of stabilization is also referred to as Baumgarte's method [Baumgarte, 1972, Chiou et al., 1999, Flores et al., 2011, Bauchau and Laulusa, 2008, Laulusa and Bauchau, 2008, Lin and Chen, 2011, Masarati, 2011]. Popov-Vereshchagin algorithm in Chapter 3 copes with acceleration level constraints, hence pose and velocity level constraints need

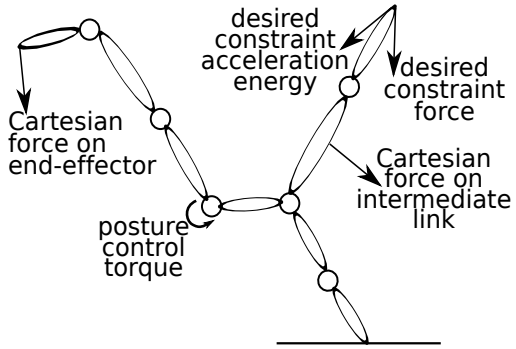


Figure 4.2: Three complementary ways exist to build controllers around the Popov-Vereshchagin algorithm: joint torques, segment forces (actual and constraint) on end-effector or internal segments, and segment acceleration energies generated against specific unit constraint forces (also at end-effector or internal segments).

to be accounted for by introducing a stabilization mechanism. Baumgarte stabilization approach suggest addition of terms that contain pose and velocity information to the linear equations $\boldsymbol{J_c\ddot{q} = \hat{b}}$ or $\boldsymbol{A^T\ddot{X} = b}$, resulting in

$$\begin{aligned} \boldsymbol{J_c\ddot{q}} &= \overbrace{\boldsymbol{\hat{b} - 2\hat{\alpha}\frac{dh(q,t)}{dt} - \hat{\beta}^2 h(q,t)}}^{\boldsymbol{\bar{b}}} \\ \boldsymbol{A^T\ddot{X}} &= \boldsymbol{b - 2\alpha\frac{dh(q,t)}{dt} - \beta^2 h(q,t)}. \end{aligned} \tag{4.5}$$

Here the right choice of $\boldsymbol{\alpha, \hat{\alpha}}$ and $\boldsymbol{\beta, \hat{\beta}}$ gains ensure that the solution of Equation (4.5) converges. This is also the reason for the structure of $\boldsymbol{\hat{b}}$ in Equation 4.4. In [Lin and Chen, 2011], the authors go further to suggest addition of the third integral term to obtain

$$\boldsymbol{J_c\ddot{q} = \hat{b} - 2\hat{\alpha}\frac{dh(q,t)}{dt} - \hat{\beta}^2 h(q,t) - \eta\int_0^t h(q,t)dt}, \tag{4.6}$$

with $\boldsymbol{\eta}$ an integration gain. In textbooks on control theory the equivalents of Equation (4.5) and (4.6) are known as PD and PID type of controllers [Aström and Murray, 2010], respectively. The well-known examples in robotics that use such stabilization approach are joint and Cartesian space trajectory tracking control application with $\boldsymbol{h(q,t) = q_d(t) - q(t)}$ and $\boldsymbol{h(q,t) = X_d(t) - X(q,t)}$, respectively.

The stabilization, i.e., *instantaneous* control of constraints can be applied on any combination of inputs and outputs of the motion/interaction. This results in the following combination of instantaneous control approaches around the constrained dynamic solver as depicted in Figure 4.2:

- external forces $\boldsymbol{F}_i^{ext}$ on the segments. These forces can be actual forces, or virtual, desired constraint forces, applied to any segment, not just the end-effector segments.
- joint forces $\boldsymbol{\tau}_i$ on the joints.
- acceleration energy $\boldsymbol{b}_N$ on the segments, generated against unit constraint forces.

Suggesting "the best" instantaneous control approaches is beyond the scope of this research (and is application dependent, anyway). Section 4.2 on simulation and examples in Sections 5.3 and 5.4 showcase applications of generic control approaches that are tested with the hybrid dynamics solver from Chapter 3.

Contributions. This chapter shows how various stabilizing controllers can be integrated in the same control loop with the constrained dynamics solver. It shows that the general architecture of the control loop does not need to change to implement a different control application. It is rather various configurations of these controllers that need to be switched either on or off.

4.2 Validation in simulation

The validations do not focus on the *quality* of the achieved control, but on showing the *richness* of the set of controller approaches that can be built around the instantaneous constrained hybrid dynamics algorithm, Algorithm 3.3, and on how each controller can be integrated into the algorithm, while keeping its efficient computational properties. This highlights the presented algorithm's major advantage: control can really be integrated into the algorithm's dynamics recursions and need not be done "outside", using the dynamics as a monolithic black box. This integration feature improves further on the computational efficiency of working with acceleration-level constraints, that is, in itself, already an improvement with respect to the state of the art.

So, the algorithm's implementation is tested with several kinematic chain configurations (over-constrained or redundant in their own right), and with

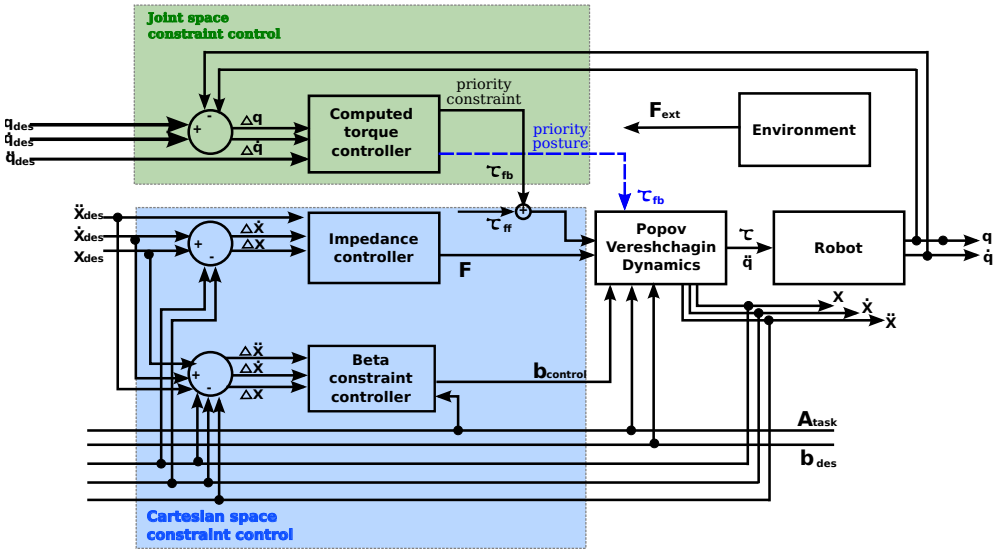


Figure 4.3: This generic control diagram depicts the interactions of various posture and operational space controllers with the constrained dynamics algorithm. It also shows the paths of controller inputs, $\tau_{(.)}$ torques, to the algorithm. Such a decoupling of the input torques allows the implementations of weighting and priority-based control schema, Figure 3.4.

combinations of end-effector constraints and joint space and Cartesian space controllers. A brief description of these configurations is given in Table 4.1.

1. **Setup for 2-DoF planar manipulator (Figure 4.4a)**: two types of settings for these experiments are considered. In the first setting, the goal is to move the manipulator depicted in Figure 4.4(a), from some configuration S_0 to another configuration S_1 in the presence of the gravity and constraints in the end-effector's X direction. In the second setting, the goal is to keep the position of the end-effector constant by applying constraints to both the X and Y degrees of freedom. For both settings, since the acceleration constraint relation is of the linear form $\boldsymbol{A}^T \ddot{\boldsymbol{X}} = \boldsymbol{b}$, one would require a controller to keep it stable. For this purpose one could use an impedance controller and take its effect as a virtual external force in the dynamics algorithm. Another possibility is to directly influence the constraints themselves by introducing an acceleration energy-based controller, which negates excessive acceleration energy introduced in an attempt to satisfy the constraint.

2. **Setup for 4-DoF planar manipulator (Figure 4.4c)**: the system

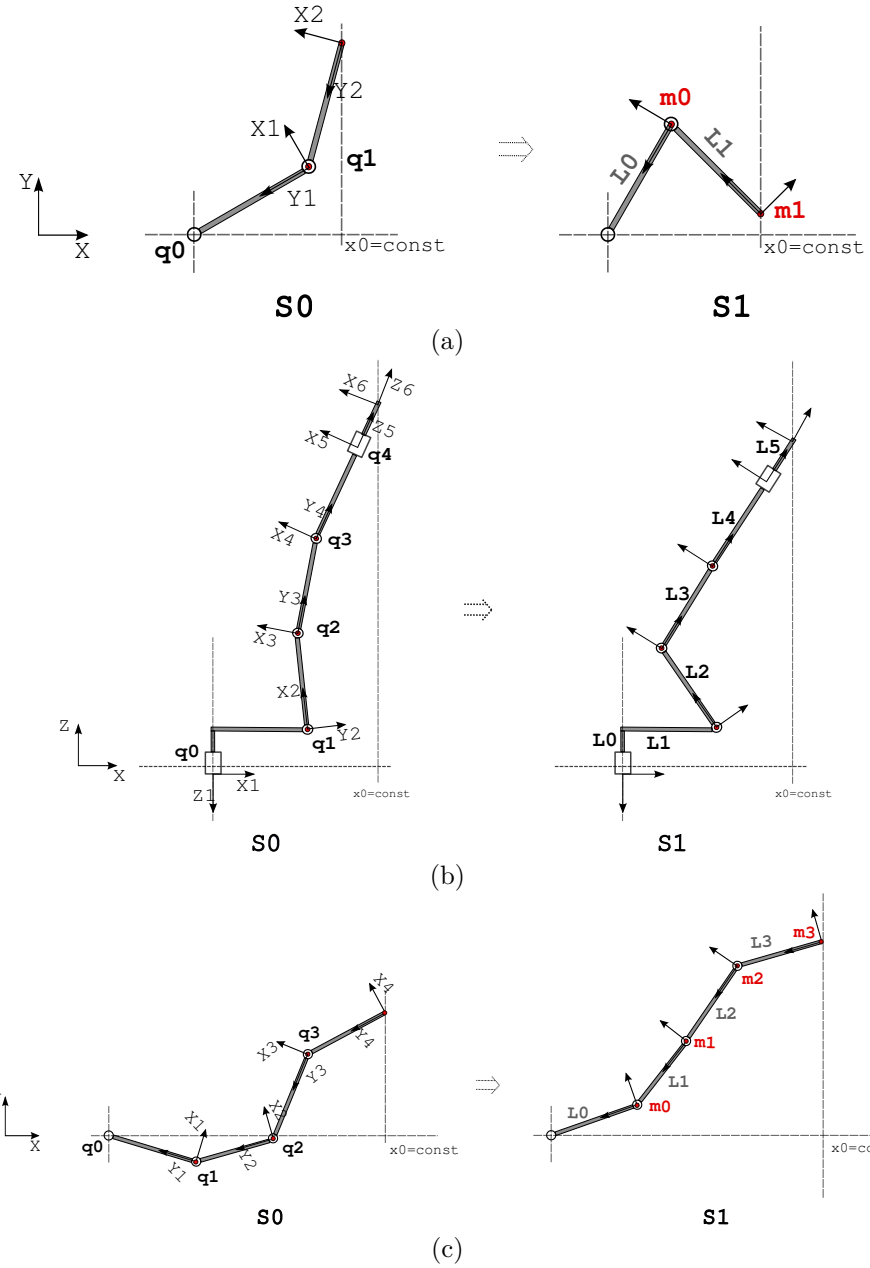


Figure 4.4: Three types of robot chain configurations of Section 4.2: in configuration (a), a 2-DoF planar serial chain is moved from its initial state on the left to the state on the right, under the effect of gravity and end-effector constraints. In configuration (b), a 5-DoF manipulator is moved in the presence of gravity and end-effector constraints. In configuration (c), a 4-DoF planar serial chain is to move vertically down under the effect of gravity and end-effector constraints. In all three cases, the constraints are of the form $\boldsymbol{A}^T \ddot{\boldsymbol{X}} = \boldsymbol{b}$.

considered in this setup is redundant and gives more opportunities to explore the effects of different combinations of constraints and controllers, both in end-effector and joint space coordinates. In particular, it is shown that the algorithm can be used either in *priority* or *weighting*-based multitask control involving robot dynamics, Section 3.5.3. In addition to constraining end-effector coordinates as described in the setup above, the joints are to follow some specified trajectories. In this setting it is shown that depending on where priorities lie, one can perform multiple tasks with different degrees of success. For instance, Figure 4.6 shows that it is possible to do a posture control with small errors and at the same time satisfy end-effector constraints, if there are enough system degrees of freedom, and priority is set to posture trajectory tracking.

3. **Setup for 5-DoF spatial manipulator (Figure 4.4b)**: the system considered in these setups represents a kinematic structure similar to that of youBot manipulator [KUKA, 2010]. In these experiments, a setting is considered where two of the end-effector's translational degrees of freedom, X and Y, are constrained, and the other last translational degree of freedom, Z is to track a polynomial trajectory. Additionally, four of the five available joints are to track polynomial trajectories.

4. **Setup for 7-DoF spatial manipulator**: the system considered in these setups represents the kinematic structure of Baxter [Rethink Robotics, 2012] robot. In experiments here, a setting is considered where two of the end-effector's translational degrees of freedom, X and Y, and one of the rotational degrees of freedom, Z are constrained. Additionally, two joints closest to the end-effector segment are to track polynomial trajectories.

For all the setups described, the Popov-Vereshchagin algorithm is used for the dynamics computations in the form equivalent to that of the articulated body algorithm, [Featherstone, 2008] and with extensions to multitask control [Khatib et al., 2002, Khatib et al., 2008] with constraints. Structurally, the constraint stabilization through acceleration energy and impedance, as well as joint space computed torque controllers are similar. This research also introduces a stabilization term based on the compensation of the acceleration energy $\boldsymbol{b}$ in Equation (3.22). The controller around $\boldsymbol{b}$ is given in the form of

$$\boldsymbol{b} = \boldsymbol{A}^T(\boldsymbol{K_a}\Delta\ddot{\boldsymbol{X}} + \boldsymbol{K_D}\Delta\dot{\boldsymbol{X}} + \boldsymbol{K_P}\Delta\boldsymbol{X}), \tag{4.7}$$

with $\Delta\boldsymbol{X} = \boldsymbol{X}_{\text{desired}} - \boldsymbol{X}_{\text{actual}}$ and $\boldsymbol{K_a}$, $\boldsymbol{K_D}$ and $\boldsymbol{K_P}$ are gains. For the joint space tracking task, a *computed torque* control of the form

$$\boldsymbol{\tau} = \boldsymbol{\tau}_{\text{ct}}(\ddot{\boldsymbol{q}}_{\text{desired}}) + \boldsymbol{K_{Dj}}\Delta\dot{\boldsymbol{q}} + \boldsymbol{K_{Pj}}\Delta\boldsymbol{q} \tag{4.8}$$

Kinematic chain configurations	**Constraints on end-effector**	**Controllers**	**Description of the task**
Planar 2-DoF serial chain (Figure 4.4a)	constraints on X and Y; constraint on X and a polynomial trajectory for Y	acceleration energy or simple impedance controller for the end-effector, and computed torque controller for joint coordinates.	keep the positions of both X and Y coordinates constant; keep the position of X coordinate constant, while following a trajectory in the Y coordinate.
Planar 4-DoF serial chain (Figure 4.4c)	constraint on X and a polynomial trajectory for Y; constraint on X and polynomial trajectories for Y and joint coordinates.	acceleration energy or simple impedance controller for end-effector, and computed torque controller for joint coord.	keep the position of the X coordinate constant, while following a trajectory in the Y coordinate; keep the position of the X coordinate constant, while following trajectories in the Y coordinates and the joint coordinates.
Spatial 5-DoF serial chain (Figure 4.4b)	constraint on X and Y and polynomial trajectory for Z and all joint coordinates.	acceleration energy or simple impedance controller for the end-effector, and computed torque controller for joint coordinates.	keep the position of the X and Y coordinates constant, while following a trajectory in the Z coordinate, and in joint coordinates
Spatial 7-DoF serial chain	constraint on XY linear and Z rotational DoF and polynomial trajectory for Z linear coordinate and all joint coordinates.	acceleration energy or simple impedance controller for the end-effector, and computed torque controller for joint coordinates.	keep the position of the X and Y coordinates, and the angle about the Z axis constant, while following a trajectory in Z coordinate, and joint coordinates.

Table 4.1: Various combinations of constraints, controllers and kinematic chains that were used in the simulations with the conventional control scheme (Figure 4.3).

was used, [Bejczy, 1974]. Here $\Delta \boldsymbol{q} = \boldsymbol{q}_{\text{desired}} - \boldsymbol{q}_{\text{actual}}$ is the tracking error, and $\Delta \dot{q}$ its time derivative; $\boldsymbol{K_{Dj}}$ and $\boldsymbol{K_{Pj}}$ are (virtual) joint space damping and stiffness matrices, respectively. The trajectory of $\boldsymbol{q}(t)$ is interpolated as a third degree polynomial, which provides the *desired* joint accelerations, velocities and positions. The desired joint acceleration $\ddot{\boldsymbol{q}}_{\text{desired}}$ then gives rise to the corresponding torque $\boldsymbol{\tau}_{\text{ct}}(\ddot{\boldsymbol{q}}_{\text{desired}})$, computed by the Popov-Vereshchagin algorithm.

The constraint stabilization that uses the impedance control has the form presented in [Albu-Schäffer et al., 2003]

$$\boldsymbol{F}_{\text{ext}} = \boldsymbol{H}^{A} \Delta \ddot{\boldsymbol{X}} + \boldsymbol{K_{Dc}} \Delta \dot{\boldsymbol{X}} + \boldsymbol{K_{Pc}} \Delta \boldsymbol{X}, \tag{4.9}$$

with $\Delta \boldsymbol{X} = \boldsymbol{X}_{\text{desired}} - \boldsymbol{X}_{\text{actual}}$, $\boldsymbol{H}^{A}$ the articulated body inertia matrix, $\boldsymbol{K_{Dc}}$ and $\boldsymbol{K_{Pc}}$ the Cartesian damping and stiffness matrices, respectively. This type of controller is equivalent in its effects to the constraint stabilization controller around $\boldsymbol{b}$ defined by Equation (4.7). The trajectory of $\boldsymbol{X}(q, t)$ is interpolated as a third degree polynomial, which provides the *desired* Cartesian accelerations, velocities and poses. Figure 4.3 depicts the control loop which performs a whole-body control by integrating different constraint-controller combinations. This control loop can not only be applied to control of the robot motion, but also to interaction type of tasks.

4.2.1 Results and analysis

The values of the parameters of the kinematic chains used in the simulations are as follows:

- For the 2-DoF manipulator, the segment lengths are $L_1 = L_2 = 0.4$m; the segment masses are $m_1 = m_2 = 0.3$kg and they are located at the distal ends of the segments (Figure 4.4a); and the joint axis inertias are $d_1 = d_2 = 0.01$ kg·m^2. The convention for frames is that the Z directions are pointing out of the page, and joint values are positive in a counterclockwise direction.
- For the 4-DoF manipulator, $L_1 = L_2 = L_3 = L_4 = 0.2$m; $m_1 = m_2 = m_3 = m_4 = 0.2$kg; $d_1 = d_2 = d_3 = d_4 = 0.01$kg·m^2; Z direction is pointing out of the page.
- For the 5-DoF manipulator, model parameters can be found in [KUKA, 2010].
- For the 7-DoF manipulator, model parameters can be found in [Rethink Robotics, 2012].

Priority on	computed torque, $\boldsymbol{\tau}(Nm)$ controller	impedance, $\boldsymbol{F}_{ext}(N)$ controller	acceleration energy $\boldsymbol{b}(Nm^2/sec)$ controller	constrained end-effector DoFs and controlled joints
Constraint satisfaction	$K_{q1} = 30.5/T^2_{inertia}$ $K_{q2} = 30.2/T^2_{inertia}$ $K_{q3} = 5.1/T^2_{inertia}$ $K_{q4} = 10.1/T^2_{inertia}$ $D_{q1} = 12.0/T^2_{inertia}$ $D_{q2} = 10.0/T^2_{inertia}$ $D_{q3} = 3.0/T^2_{inertia}$ $D_{q4} = 3.0/T^2_{inertia}$	$K_y = 370.232/T^2_{inertia}$ $D_y = 25.559/T_{inertia}$	$K_x = 70.345/T^2_{inertia}$ $D_x = 95.598/T_{inertia}$ $K_{a_x} = 0.005$	$A^T\ddot{X} = b = 0$, all joints are controlled
Posture satisfaction	$K_{q1} = 30.5/T^2_{inertia}$ $K_{q2} = 30.2/T^2_{inertia}$ $K_{q3} = 35.1/T^2_{inertia}$ $K_{q4} = 60.1/T^2_{inertia}$ $D_{q1} = 12.0/T^2_{inertia}$ $D_{q2} = 10.0/T^2_{inertia}$ $D_{q3} = 3.0/T^2_{inertia}$ $D_{q4} = 3.0/T^2_{inertia}$	$K_y = 370.232/T^2_{inertia}$ $D_y = 25.559/T_{inertia}$	$K_x = 70.345/T^2_{inertia}$ $D_x = 95.598/T_{inertia}$ $K_{a_x} = 0.005$	$A^T\ddot{X} = b = 0$, all joints are controlled

Table 4.2: Controllers and their gain configurations as used in priority-based multitask control with non-conflicting constraints for 4-DoF robot. Here T (sec) is the time required to finish the task and $T_{inertia}$ is the natural inertial time constant of the systems. These values were obtained for $T = 2.5$s, simulation time of $2T$, $T_{inertia} = 0.5$s and sampling rate of 0.001s. Figure 4.6 depicts the results associated with this table. Joint variable $\boldsymbol{q}$ is in *rad*.

Setup 1: one of the coordinates of the end-effector is constrained and the remaining coordinate follows a trajectory : Figure 4.5(a)-(c) shows the results of the simulation with a 2-DoF planar manipulator. In this setting X coordinate of the end-effector is constrained using the acceleration constraint

$$\boldsymbol{A}_N = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \end{pmatrix}, \quad \boldsymbol{b}_N = \begin{pmatrix} 0 \end{pmatrix}. \tag{4.10}$$

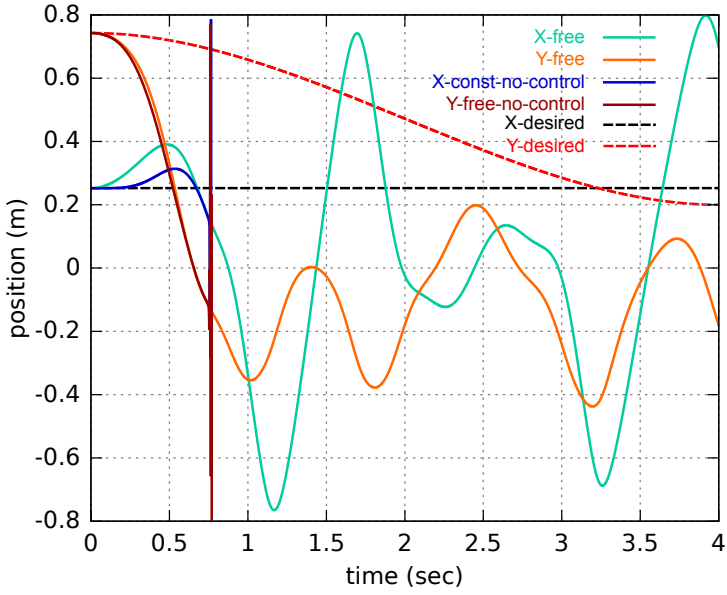
and Y coordinate is to follow a polynomial trajectory as in Equation (4.13) but expressed in Y Cartesian coordinates. Figure 4.5a compares two configurations: X is constrained as in Equation (4.10) and Y is free; both X and Y are free. As expected, when both coordinates are unconstrained the end-effector moves under the influence of the gravity in a free form and since there is no loss due to friction or any other dissipative forces, the motion will continue infinite amount of time. When X is constrained, then there is a kind of virtual external dissipative force on the end effector which alternatingly works with or against the gravity. This constraint force immediately starts to work against the motion along X axis. This effect is observed in the amplitude of the X curve (in blue) when compared to the unconstrained case (in cyan). Since the constrained is not controlled the motion eventually leads to high amplitude oscillations. In order to cope with those, the constraint needs to be controlled using one of the approaches discussed in Section 4.1. In this setup X constraint is controlled using beta controller and Y is prescribed to follow a trajectory, Figure 4.5b. As expected from the simulation Figure 4.5c shows almost a perfect tracking behavior.

Setup 1: fixed end-effector position : In this setting, the task is to keep the pose of the end-effector constant in the presence of the gravity. This is accomplished by constraining both X and Y coordinates of the end-effector using the acceleration constraint

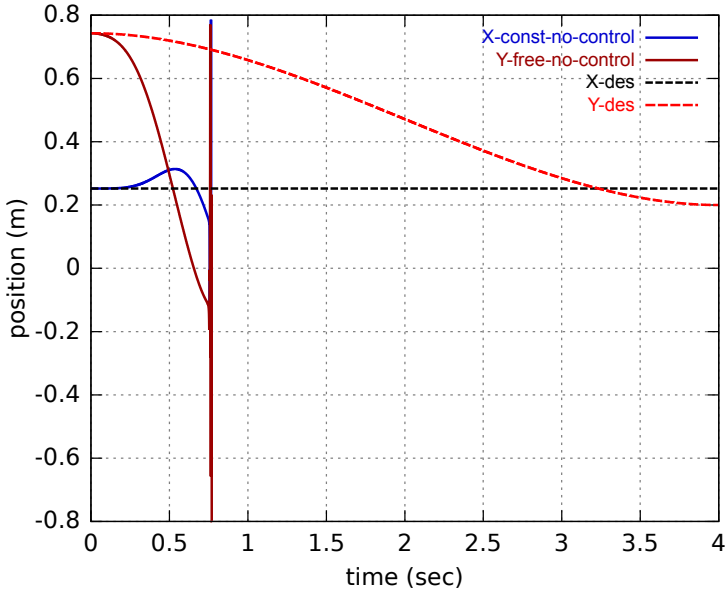
$$\boldsymbol{A}_N = \begin{pmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 1 & 0 \\ 0 & 1 \\ 0 & 0 \end{pmatrix}, \quad \boldsymbol{b}_N = \begin{pmatrix} 0 \\ 0 \end{pmatrix}. \tag{4.11}$$

One can interpret this setup as a static equilibrium, where the manipulator which is under the effect of the gravity is being balanced by holding its tip.

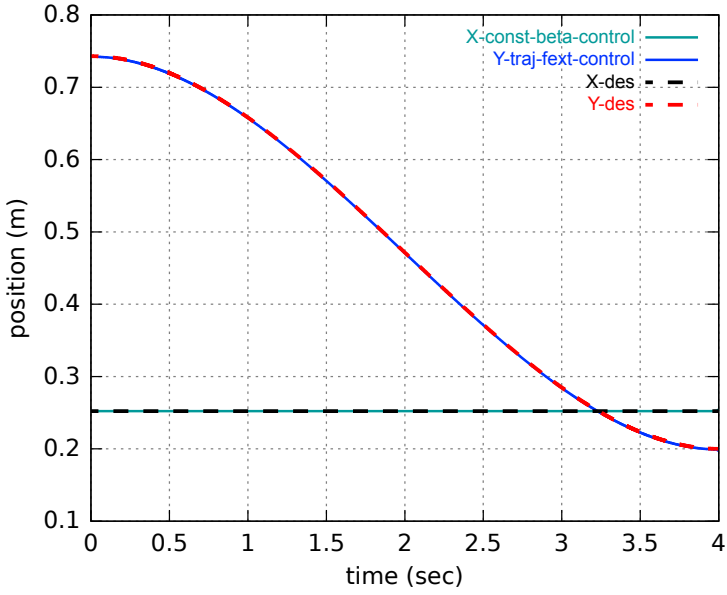
Figure 4.5d compares the configurations when either both or none of the end-effector coordinates are constrained. The outcome is similar to that in the case when only X coordinate is constrained. But unlike Figure 4.5a, now it takes some time before the motion becomes unstable. It can be explained by the fact that there is now an external constraint force that is working against the gravity along Y coordinate. Considering the initial configuration it starts from, the end-effector is in the first quadrant (both X and Y are positive) and it is closer to Y-axis, both X and Y converge to zero. This is a 'bend over' configuration where end-effector coincides with the base. But because of the oscillatory motion that never disappeared, the simulation eventually becomes unstable. But addition of the beta controller-based stabilization, Figure 4.5e, keeps the end-effector as its initial configuration. The same simulation is also performed with different initial configuration, Figure 4.5f that shows also a similar outcome. In this new configuration, constrained in Y coordinates is stabilized using F^{ext}-based simple impedance controller. Though Figure 4.5f shows perfect tracking behavior for both types of controllers, one can see that their performances differ, Figure 4.5g. As can be observed, the system keeps its position after some transition phase. This difference comes from the fact that the controller use different gain values Equation (4.7) vs Equation (4.9). Furthermore, the contributions of the F^{ext} and beta controllers are taken into account during first outward sweep and immediately after the second inward sweep during computations of the constraint forces respectively, Algorithm 3.3. This also contributes to the difference of Y coordinate in Figure 4.5g.



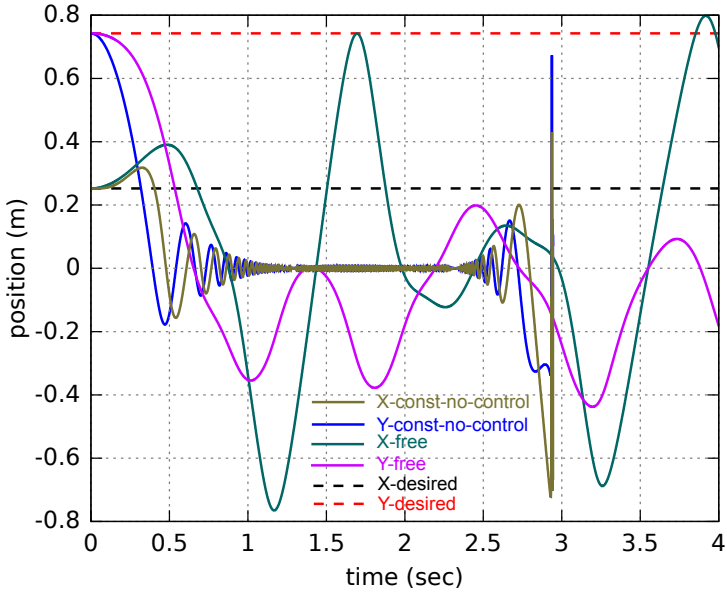
(a) X and Y coordinates of the end-effector when X is constrained but not controlled and Y is free to those when both X and Y are unconstrained/free.



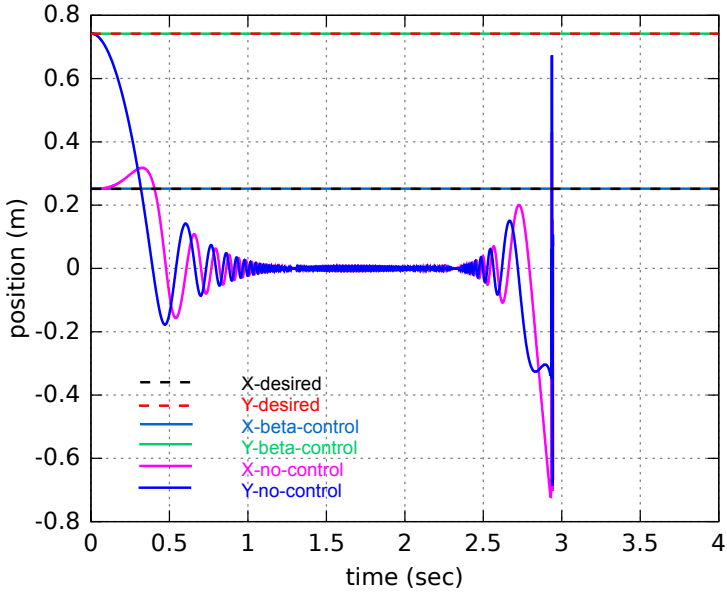
(b) X and Y coordinates when X is constrained and Y is free to those when they are to track some desired motions.



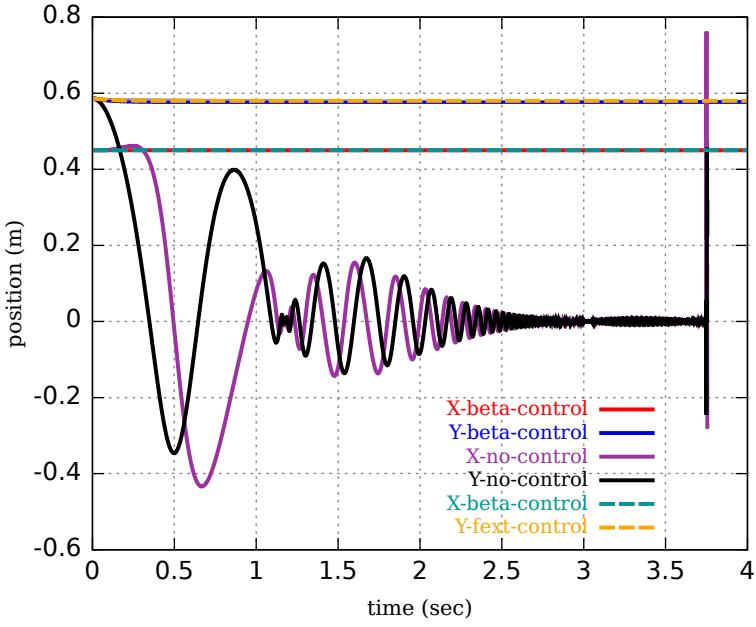
(a) X and Y coordinates when X is constrained and controlled using beta controller, and Y tracks a trajectory and uses F^{ext} based impedance controller.



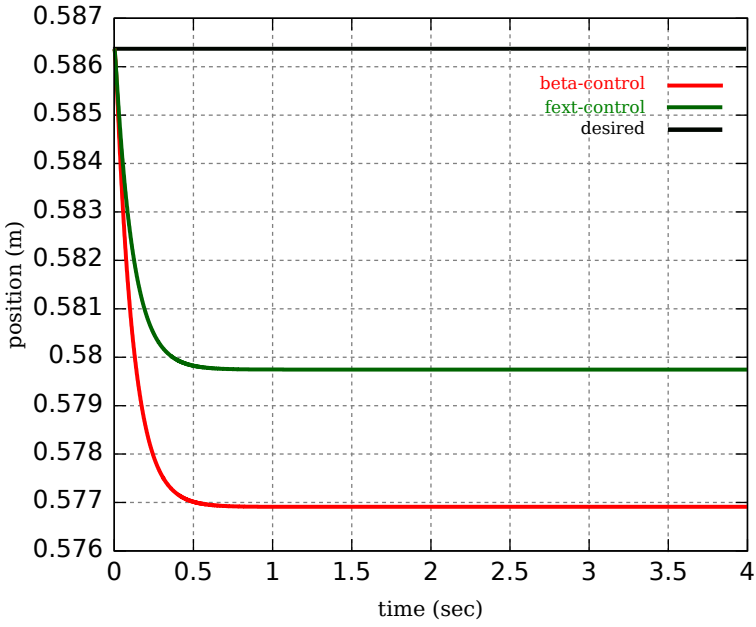
(b) X and Y coordinates of the end-effector when both X and Y are constrained but not controlled to those when both X and Y are unconstrained/free.



(a) X and Y coordinates of the end-effector when both X and Y are constrained and are controlled using beta controllers.



(b) A similar setup as in 4.5d but with different initial configurations. Also, Y is stabilized either by the beta controller as before or by the F^{ext} impedance controller.



(a) The performance of the F^{ext} impedance controller and beta controllers that stabilize Y coordinate in the configuration of 4.5f.

Figure 4.8: Simulation data for a 2-DoF planar manipulator. The first task is to constraint X coordinate and let Y coordinate to follow a trajectory. The second task is to keep the end-effector position fixed by constraining it both in X and Y coordinates.

Setup 2: controlled end-effector fall under the gravity and non-conflicting constraints: in this setup the task is to move the end-effector of the redundant planar 4-DoF robot vertically along the Y axis, while keeping its X coordinate constant. From the point of Cartesian coordinates of the end-effector, this is the same as in the first case of 2-DoF robot:

$$\boldsymbol{A}_N = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \end{pmatrix}, \quad \boldsymbol{b}_N = \begin{pmatrix} 0 \end{pmatrix}. \tag{4.12}$$

But it gets interesting in joint space, because the 4-DoF robot is redundant for this task. Due to this fact, the algorithm will try to search for all possible joint configurations that would satisfy imposed constraint. From the physical point of view, this is not only inefficient, but also unrealistic. Because, most probably some of the solutions are not physically realizable, for instance, joints having infinitely high rates (singular configuration), high torques or joint positions having values bigger than 2π. In order to restrict the solution space to the physically feasible set, one should define a set of tasks in joint/posture space. Such tasks could include avoiding joint limits or following prescribed joint trajectories. Depending on whether Cartesian or posture space constraints need be satisfied best, one can use either weighting or priority-based task control approaches as explained in Section 3.5.3. In this task, in addition to satisfying the end-effector's X DoF constraint, three of the joints of the robot are to follow an (admittedly rather artificially created) trajectory profile as in

$$f(q,t) = q_0 + q_1 t + q_2 t^2 + q_3 t^3, \tag{4.13}$$

where the values of coefficients of the polynomial depend on the initial and final values for that joint. Each of the joints is controlled independently using computed torque controller Equation (4.8). As can be seen in Figures 4.6(a)–(d), the first three joints of the system follow the prescribed trajectory with some bounded error, while the fourth joint is free. In Figure 4.6d the dashed line (desired) represents a motion profile the fourth joint would follow if it were controlled too. This is to show that the tracking in the fourth joint is not present. Otherwise, it might give an impression that some trajectory is being tracked, since all the other curves are stacked together.

Figure 4.6e compares acceleration in X coordinate with different configurations of constraint controller. As can be observed, regardless of whether weighting or priority-based approaches are used, the values of the X DoF acceleration is around **zero**. This is difficult to observe on this scale, but looking at Figure 4.6f

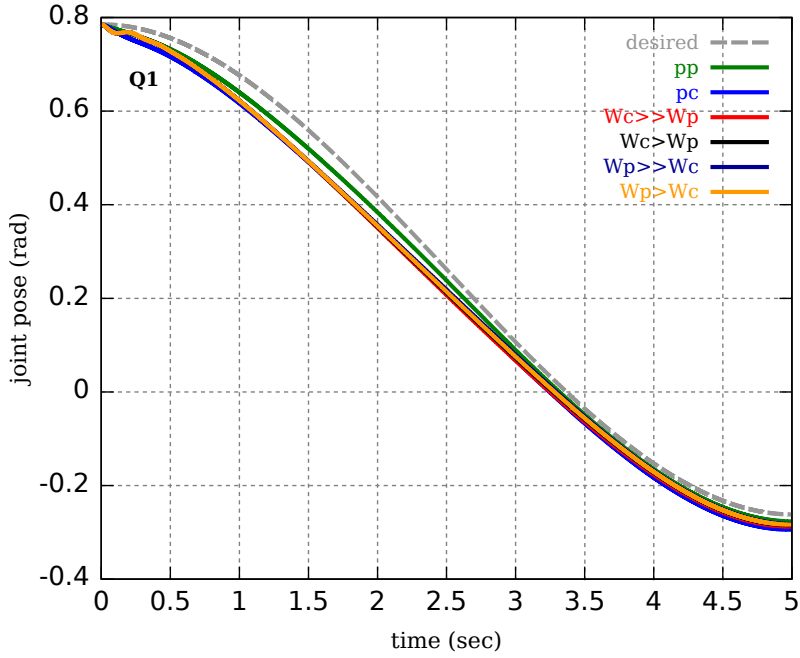
which compares the accelerations in X and Y with different kind of prioritization, it is clear that the motion caused by the acceleration in X are negligible. This can also be confirmed by looking at the value of the position in X, Figure 4.6g. Here, the worst performance is obtained when the priority is given to the satisfaction of joint space trajectories. Then, the worst case absolute error in position is 2.5 cm and with respect to the total length of the manipulator, 4×0.2 m it makes around 3.125%. Figure 4.6g shows a 'zoom in' version of a priority on the end-effector constraint. Here the worst case absolute error is in sub-millimeter scale. Another factor that influences behaviors of the constraint controllers is that weights and priorities are defined for different physical quantities, acceleration energy and joint space acceleration respectively. Furthermore, the weights in acceleration energy controller are numbers that dictate how much acceleration energy should be 'added' to or 'removed' from posture contributions. Whereas, priority is determined by the contributions of the joint space acceleration during different phases of computational sweeps, Section 3.5.3.

Setup 2: controlled end-effector fall under gravity, with conflicting constraints: this setup is exactly the same as the one discussed above, but now all of the four joints of the system are controlled. This clearly leads to a conflict between satisfaction of end-effector constraints and posture constraints. Because unlike in the previous case, the number of degrees of freedom of the system (four) is less than what the task requires (five degrees of freedom). A direct consequence of this, is that, regardless of the control approach used, the constraints are almost *never satisfied* (except when the priorities are on end-effector constraints). This is in contrast to the case above with the non-conflicting constraints, where all control methods satisfied the constraints, even if their quality were different and not perfect.

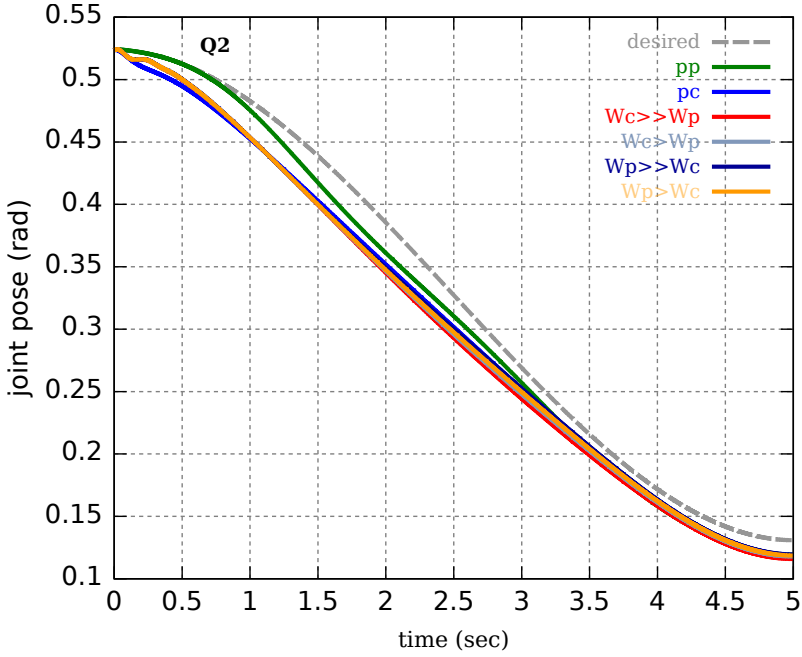
While analyzing data, one is to consider two issues: (i) weighing and prioritization are on different quantities and performed during different stages of the computational sweeps; (ii) because of the conflicting constraints, controllers may not behave as one expects them to. Such effects are well observable in Figures 4.7(a)–(d), where the quality of joint space tracking is very poor, regardless whether the priority is set on posture control or not. One could assume, if the proportional gain of the joint space controller, Equation (4.8), is tuned more aggressively a better performance could be achieved, but that kind of approach will be at the cost of the stability of the system's motion. Furthermore, conflicting constraints exhibit more aggressive dynamics during the transition phase, Figures 4.7e and 4.7f, than that in non-conflicting Figures 4.6e and 4.6f. This could also explain higher torques at the joints, Figures 4.8(a)-(b) than those in non-conflicting cases, Figures 4.8(c)-(d).

For the controllers based on weighing of acceleration energy, the contributions of posture/joint space force and Cartesian space constraint forces are weighed according to Equation (3.42). In Equation (3.30) the forces in inner braces that consist of articulated bias forces contribute to the posture acceleration, whereas the last term involving Lagrange multiplier $\boldsymbol{\nu}$ contributes to Cartesian space constraint acceleration. In weighing-based controller, these two contributions are weighed before computing the acceleration of the system. Figure 4.7g shows different cases that involve these controllers in conflicting constraint setup. At the first impression, one can observe a strange behavior of weighing-based controllers for $W_c > W_p$ and $W_c >> W_p$ (in Equation (3.42), $W_c = w_{ee}$). One would expect $W_c >> W_p$ to be closer to the case with the priorities on the end-effector constraint (default case of the constrained dynamics). The problem here is that this figure is constructed a bit incorrectly, because it assumes the same conditions for all the controllers. But $W_c > W_p$ and $W_c >> W_p$ cases use completely different weights. $W_c > W_p$ and $W_p > W_c$ use one set of gains that are interchanged between posture and constraint contributors, whereas $W_c >> W_p$ and $W_p >> W_c$ use another set of gains that are also interchanged between posture and constraint contributors. That is why, these two sets are symmetric with respect to the nominal constrained dynamics output. It is interesting that this 'mistake' becomes evident in conflicting setup, but not in non-conflicting setup in Figure 4.6g.

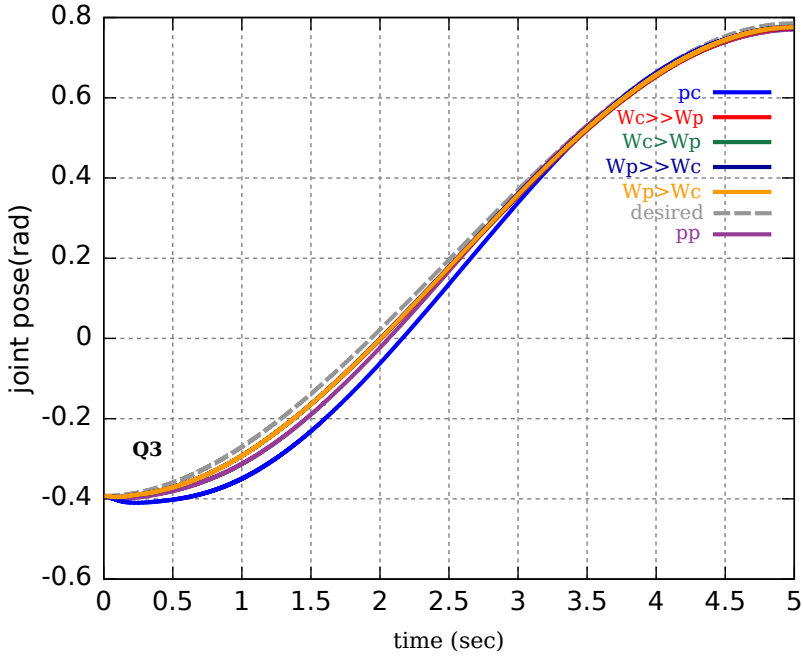
But again, looking at Figure 4.7h (this is 'zoom in' into 4.7g) one can observe that in its default configuration the constrained dynamics still satisfies the end-effector constraint.



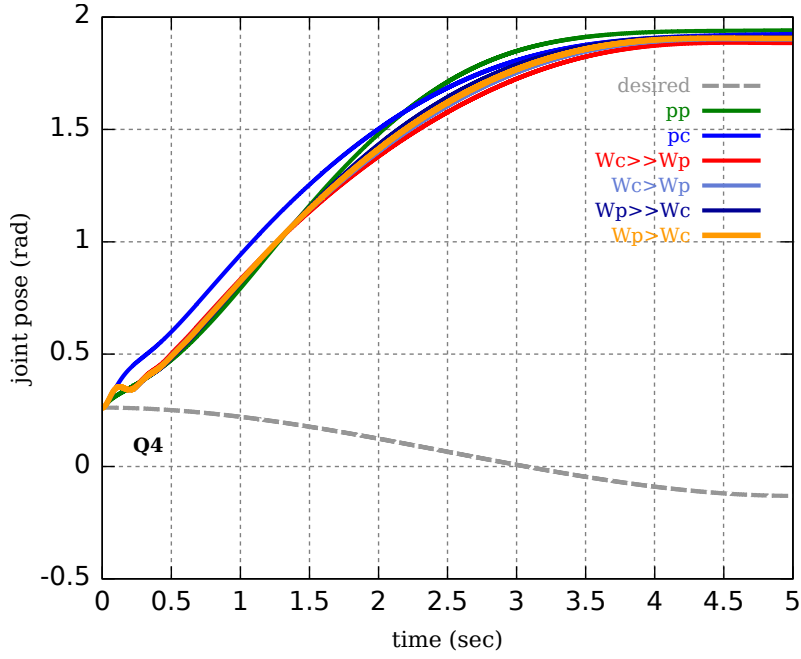
(a) Tracking behaviors of weighing and priority based controllers for the first joint.



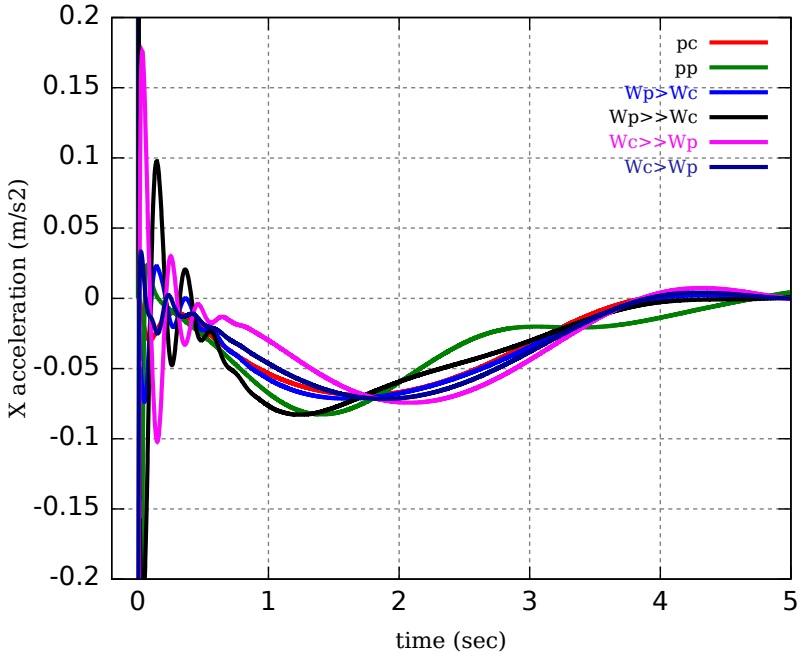
(b) Tracking behaviors of weighing and priority based controllers for the second joint.



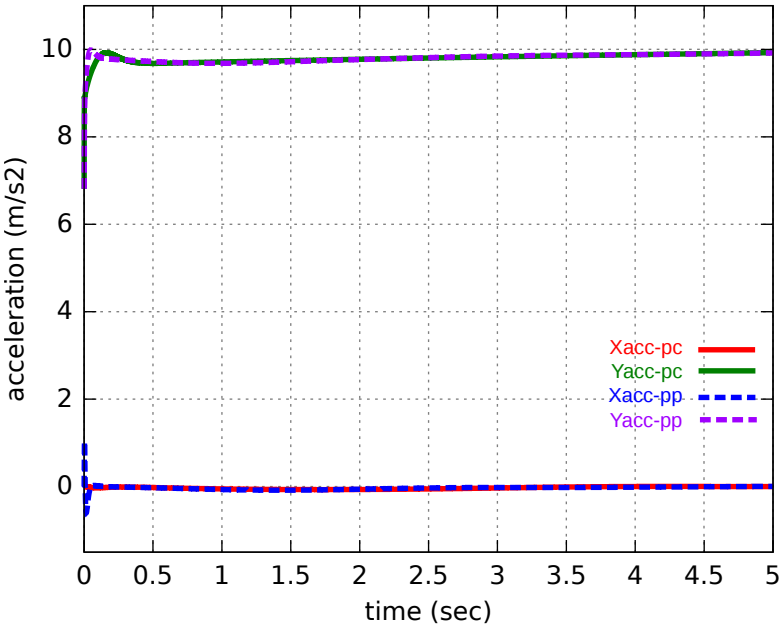
(a) Tracking behaviors of weighing and priority based controllers for the third joint.



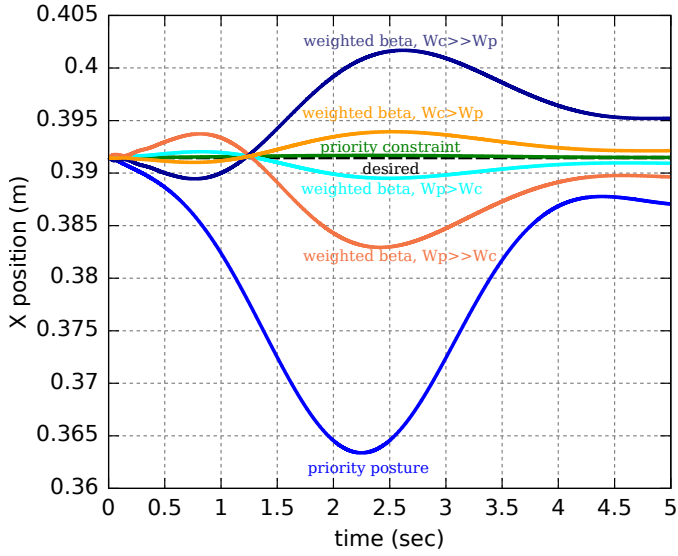
(b) A behavior of the unconstrained fourth joint each time a different controller configuration is used.



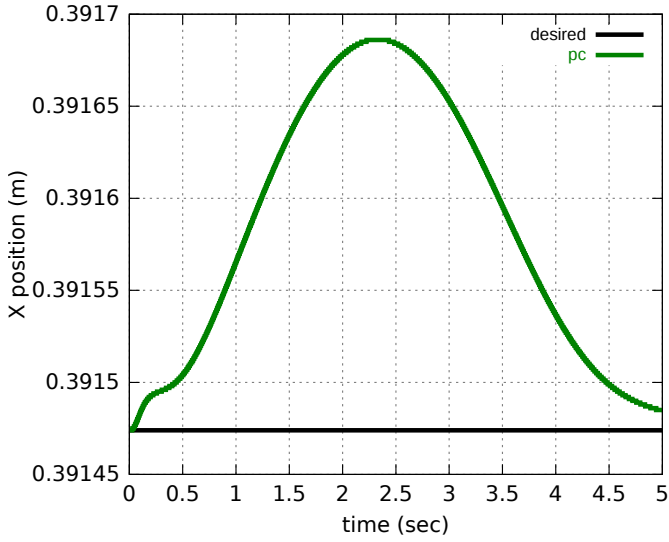
(a) Effects of weighing and priority based controllers on the acceleration in X coordinate.



(b) Effects of priority based controllers on the accelerations in X and Y coordinate.

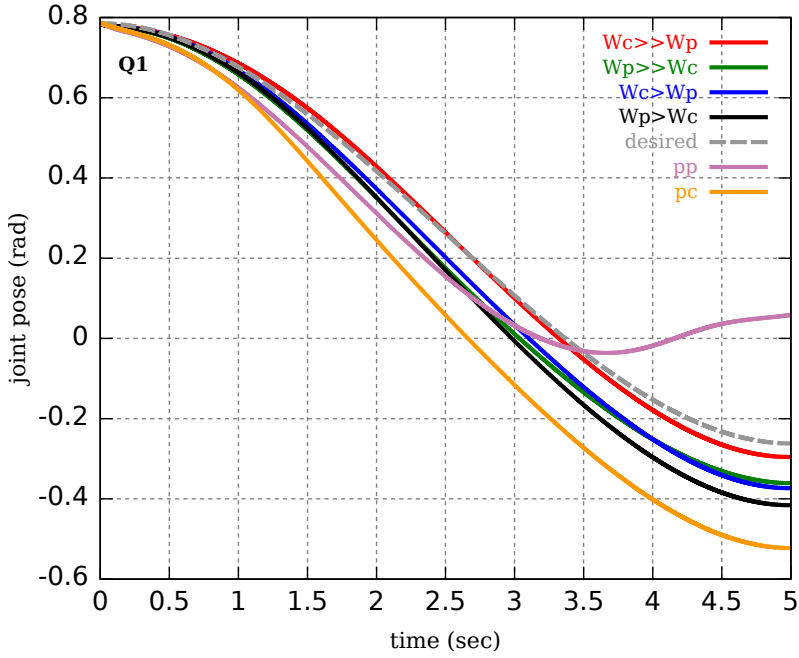


(a) Dynamics of the position constraint on X coordinate under the influence of weighing and priority based controllers.

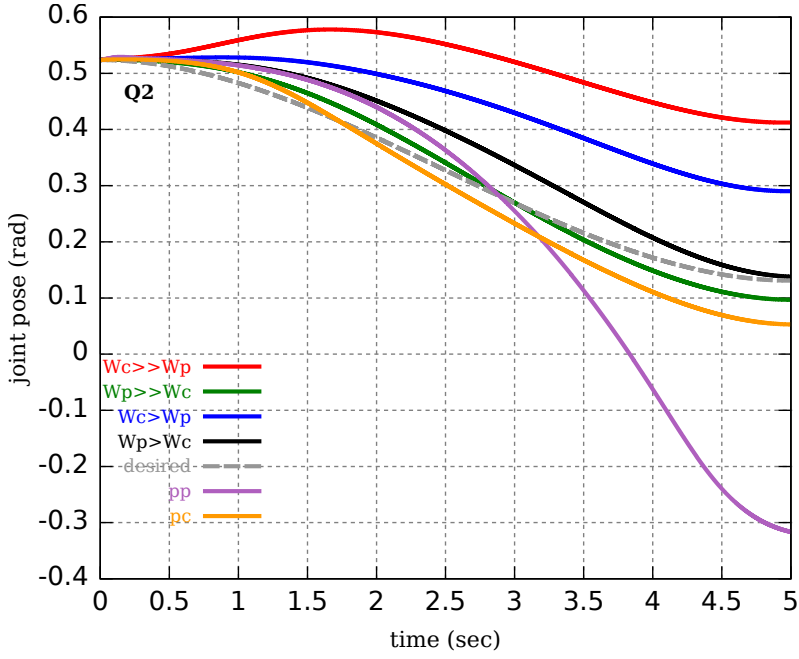


(b) A transition phase dynamics of X coordinate constraint when the controller prioritizes satisfaction of end-effector constraint.

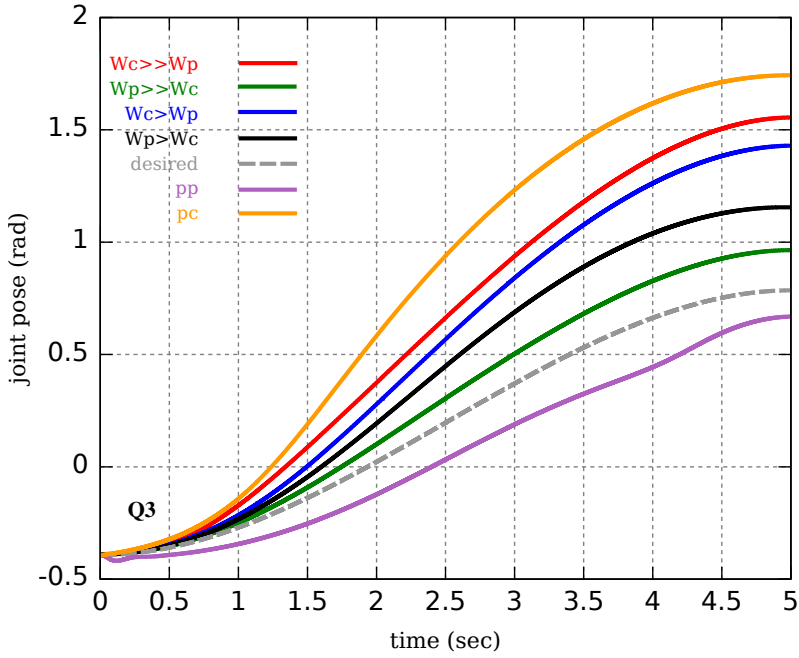
Figure 4.12: A comparison of six different controller setups using priorities and weighting. The data is associated with a setup with *non-conflicting* constraints. W_c, W_p, pp, pc stand for constraint weight, posture weight, priority posture and priority end-effector constraint respectively.



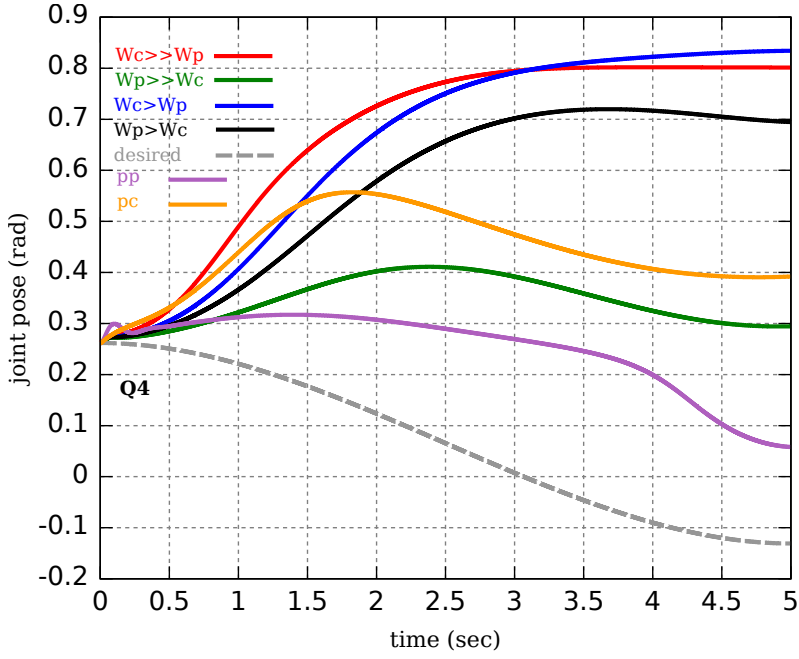
(a) Tracking behaviors of weighing and priority based controllers for the first joint.



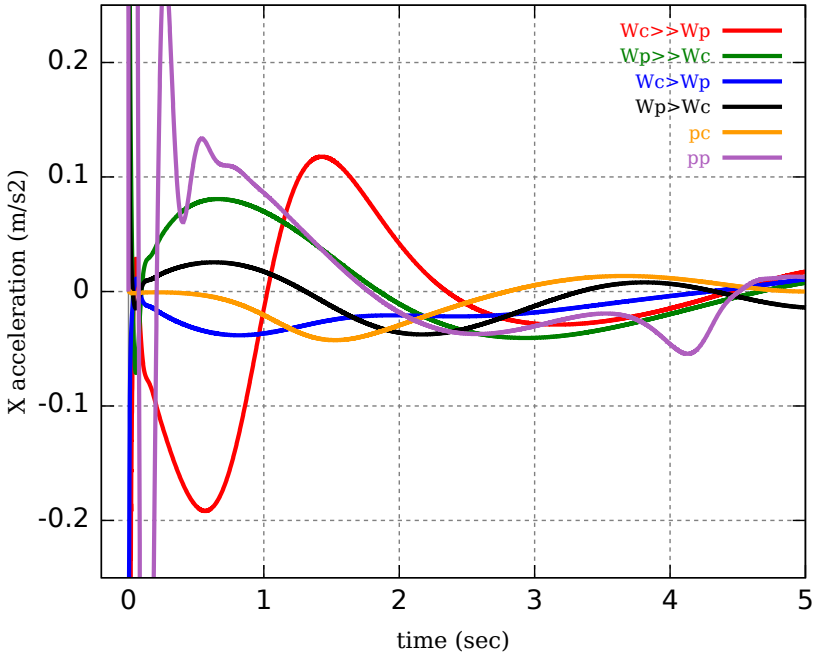
(b) Tracking behaviors of weighing and priority based controllers for the second joint.



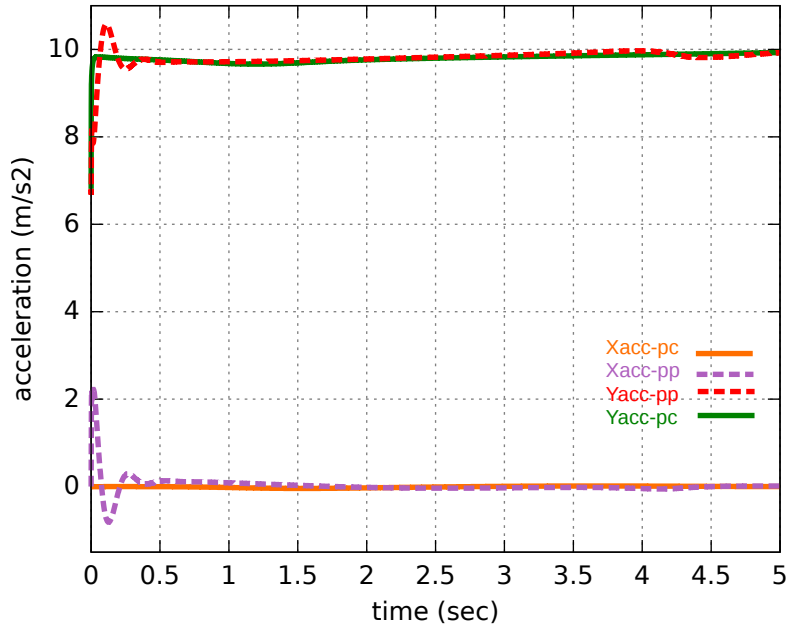
(a) Tracking behaviors of weighing and priority based controllers for the third joint.



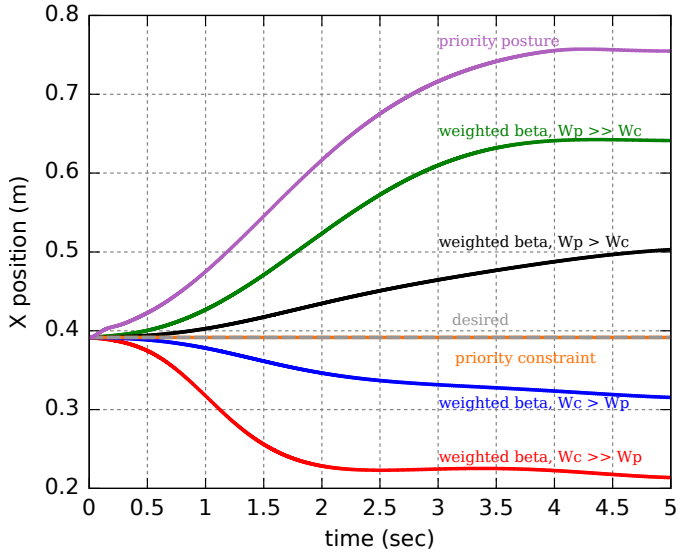
(b) Tracking behaviors of weighing and priority based controllers for the fourth joint.



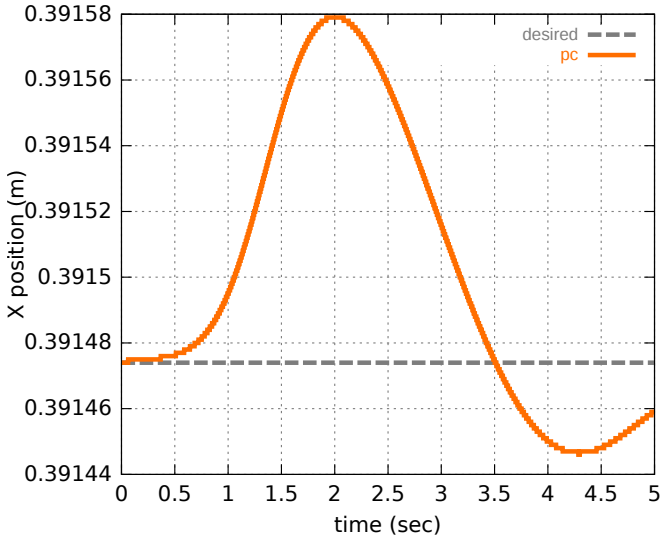
(a) Effects of weighing and priority based controllers on the acceleration in X coordinate.



(b) Effects of priority based controllers on the accelerations in X and Y coordinate.

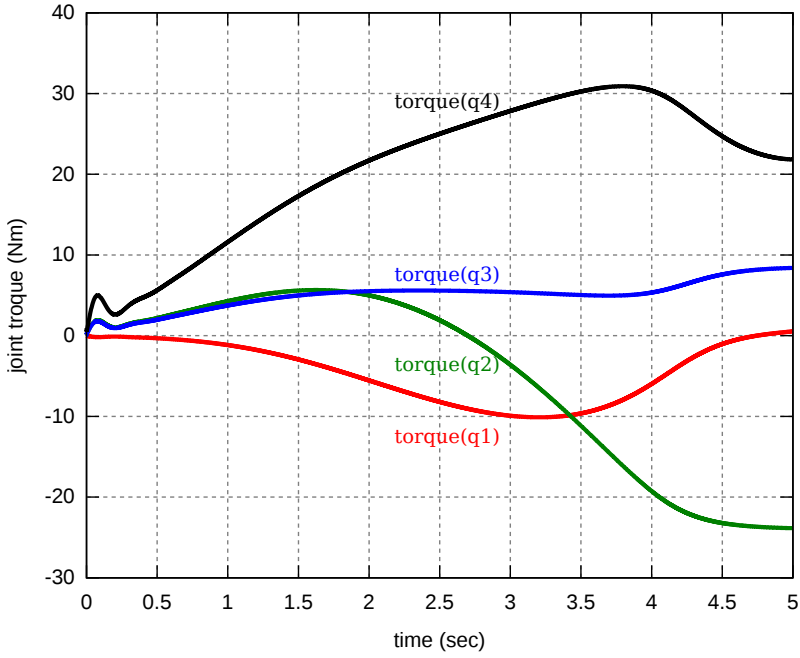


(a) Dynamics of the position constraint on X coordinate under the influence of weighing and priority based controllers.

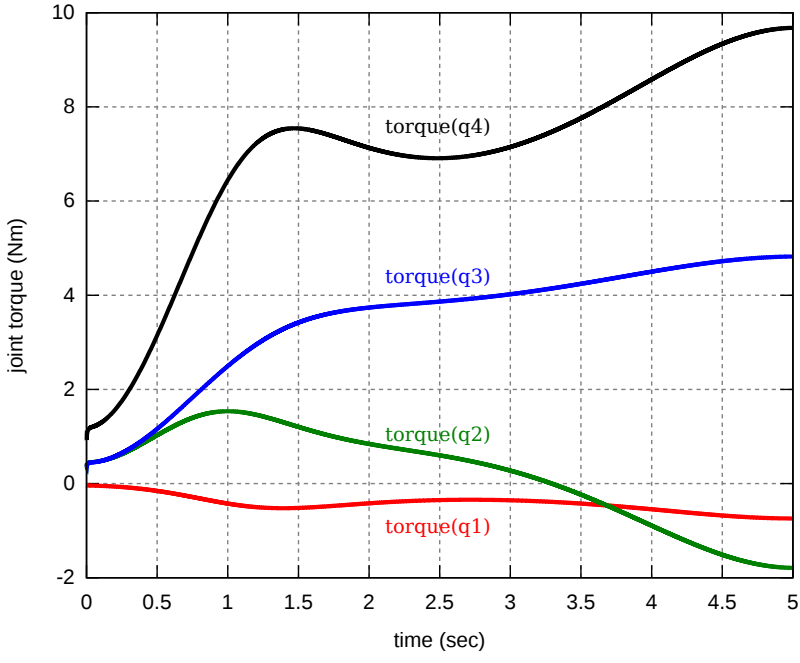


(b) A transition phase dynamics of X coordinate constraint when the controller prioritizes satisfaction of end-effector constraint.

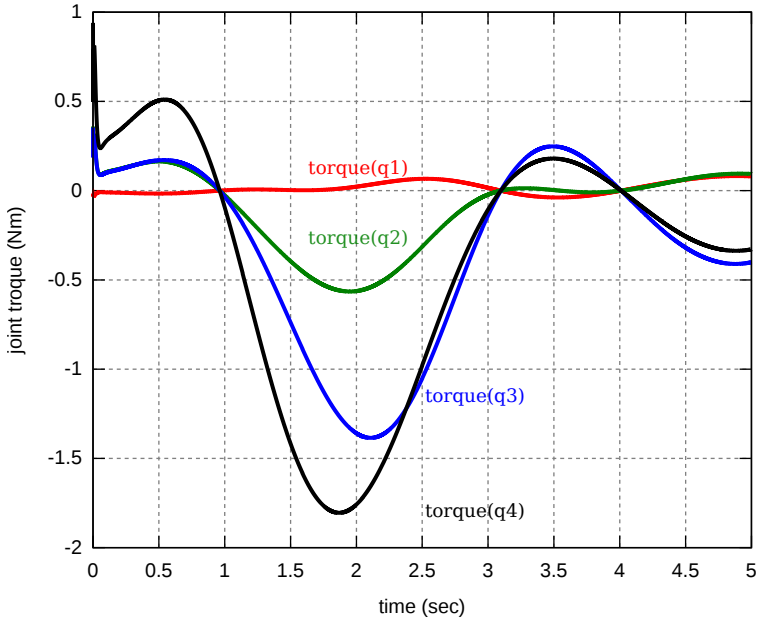
Figure 4.16: A comparison of six different controller setups using priorities and weighting. The data is associated with a setup with *conflicting* constraints. W_c, W_p, pp, pc stand for constraint weight, posture weight, priority posture and priority end-effector constraint respectively.



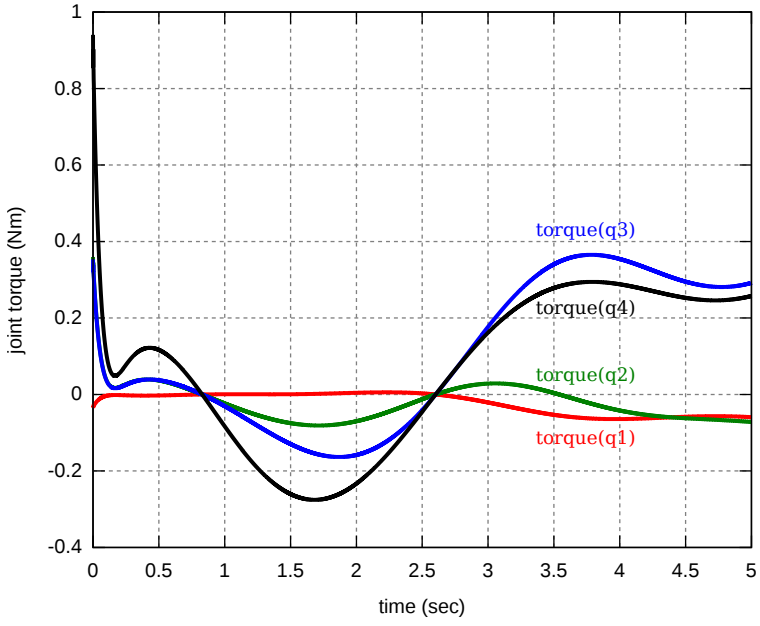
(a) Joint torque profiles when the posture and end-effector constraints are conflicting and with the priority to satisfy the posture constraints



(b) Joint torque profiles when the posture and end-effector constraints are conflicting and with the priority to satisfy the end-effector constraints.



(a) Joint torque profiles when the posture and end-effector constraints are non-conflicting and with the priority to satisfy the posture constraints.



(b) Joint torque profiles when the posture and end-effector constraints are non-conflicting and with the priority to satisfy the end-effector constraints.

Figure 4.18: A comparison of the joint torque data for the setups with conflicting and non-conflicting constraints on 4-DoF serial chain.

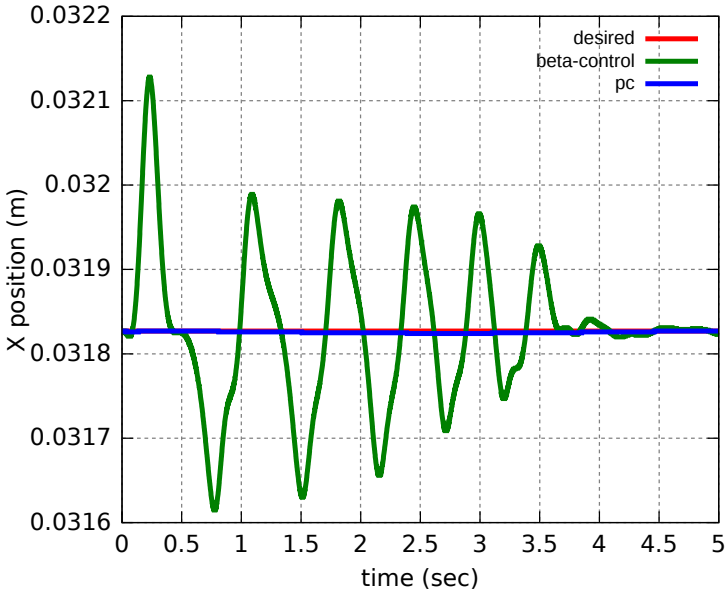
Setup 3: controlled end-effector fall of a 5-DoF spatial robot under gravity, with conflicting constraints: in this setup the motion of the end-effector of a 5-DoF youBot manipulator, [KUKA, 2010], is constrained in X and Y coordinates. The constraint specification is given as

$$\boldsymbol{A}_N = \begin{pmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 1 & 0 \\ 0 & 1 \\ 0 & 0 \end{pmatrix}, \quad \boldsymbol{b}_N = \begin{pmatrix} 0 \\ 0 \end{pmatrix}. \tag{4.14}$$

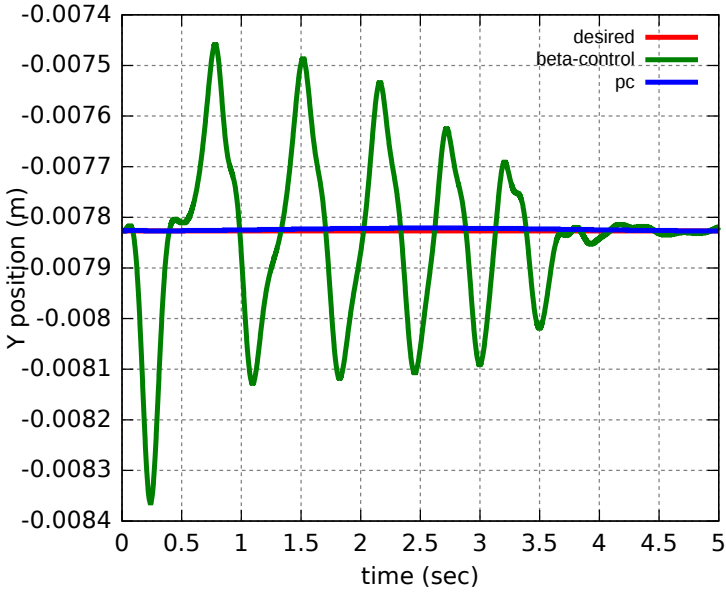
The end-effector is to follow a polynomial trajectory in Z coordinate, Equation (4.13). Figures 4.9(a)–(c) depict the data on X, Y and Z position coordinates. In X and Y coordinates the behavior is as expected. First, X and Y coordinates of the end-effector are constrained and no joint space trajectories are present (green), i.e., non-conflicting situation with three linear DoF of the end-effector are involved. The X and Y are controlled with beta controller and Z is tracking a polynomial trajectory using a simple impedance controller input to the dynamics through $\boldsymbol{F^{ext}}$. The performance of the latter is poor, but it is clearly tracking the given trajectory, while X and Y after undergoing some oscillations during the transition phase converge to their values imposed by the constraints. In the second case, four joints are to track trajectories of their own, thus creating an over-constrained setup, i.e., two acceleration constraints and five trajectories. During the configuration of the controllers, a priority is given to the satisfaction of the end-effector acceleration constraints, i.e., default configuration of the dynamics algorithm. X and Y directly converge top their desired values, while Z coordinate does not track its trajectory anymore. The latter is due to the fact that $\boldsymbol{F^{ext}}$ used to control the trajectory is input during the initial outward sweep of the algorithm and its contribution to the end-effector motion is weighted down by the articulated body inertia during the second inward sweep. Figure 4.9d shows ZYX-Euler angles that oscillate with a high amplitude when there is only stabilization on the end-effector constraints and no posture control. On the other hand, addition of the posture controllers can lead to less erratic behavior. This might be explained by the fact that there are fewer free system DoFs that can be allocated to the motion of the orientation.

Figure 4.9e compares joint motion profiles, when there are only end-effector constraints are present and the joints are free to move. The desired trajectory serves as a reference, in order to show that no motion is being tracked. Figure 4.9f depicts the over-constrained situation, when four of the five joints are controlled to track polynomial trajectories. There are two factors that contribute to this outcome: (i) the priority is set to satisfy the end-effector acceleration constraints,

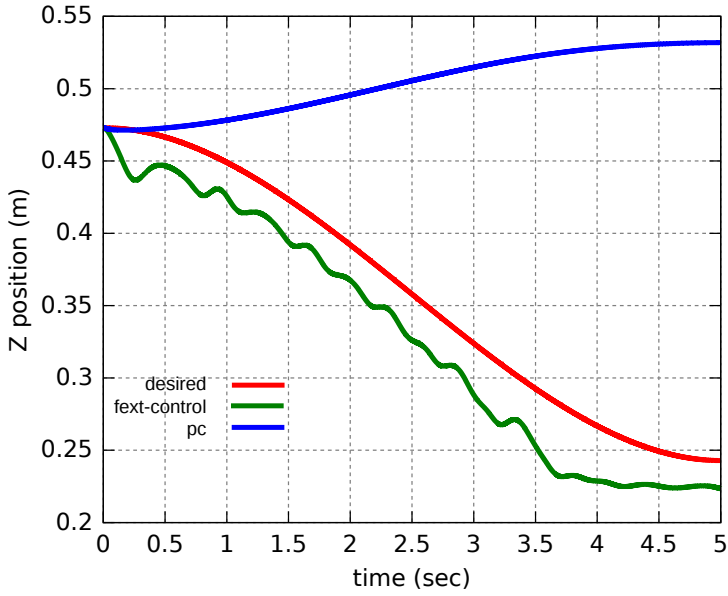
(ii) the system is over-constrained which makes it difficult to obtain desired outcome for all tasks.



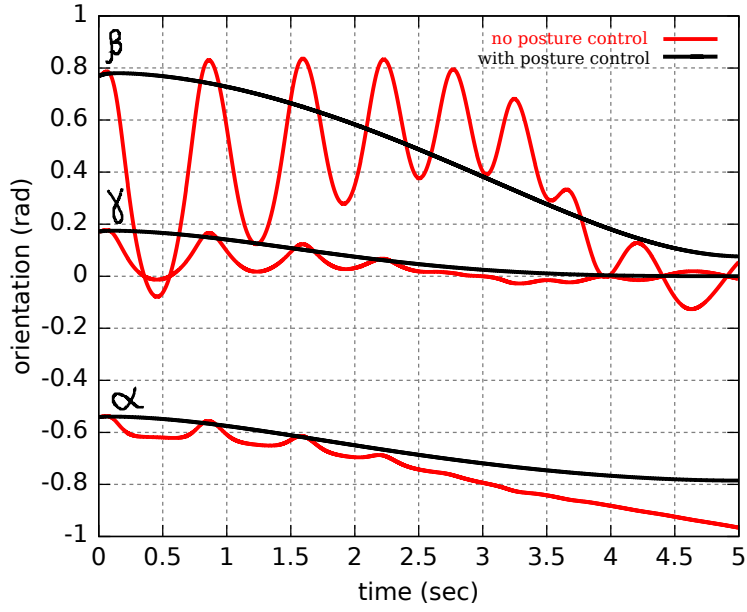
(a) Data on X coordinates. In the first case (green), X is constrained and controlled with beta controller but no posture trajectories are imposed. In the second case (blue), the posture trajectories are present but a priority is given to end-effector constraint.



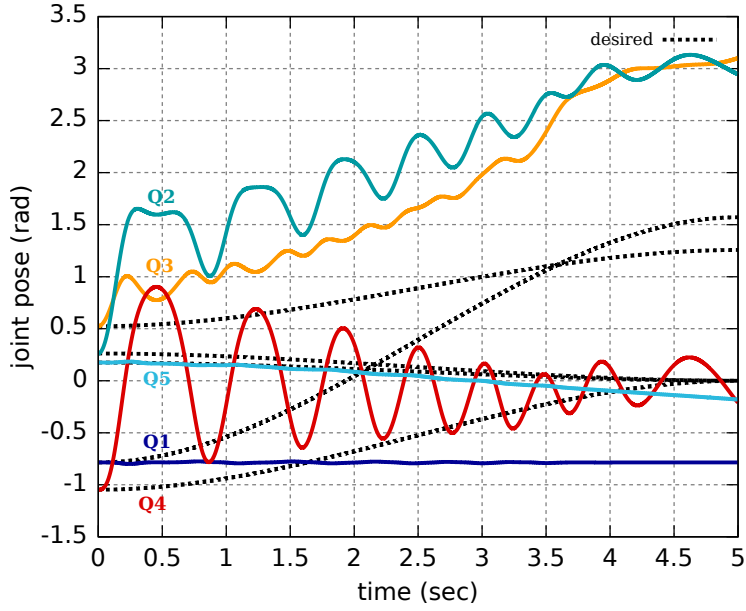
(b) Data on Y coordinates. In the first case (green), Y is constrained and controlled with beta controller but no posture trajectories are imposed. In the second case (blue), the posture trajectories are present but a priority is given to end-effector constraint.



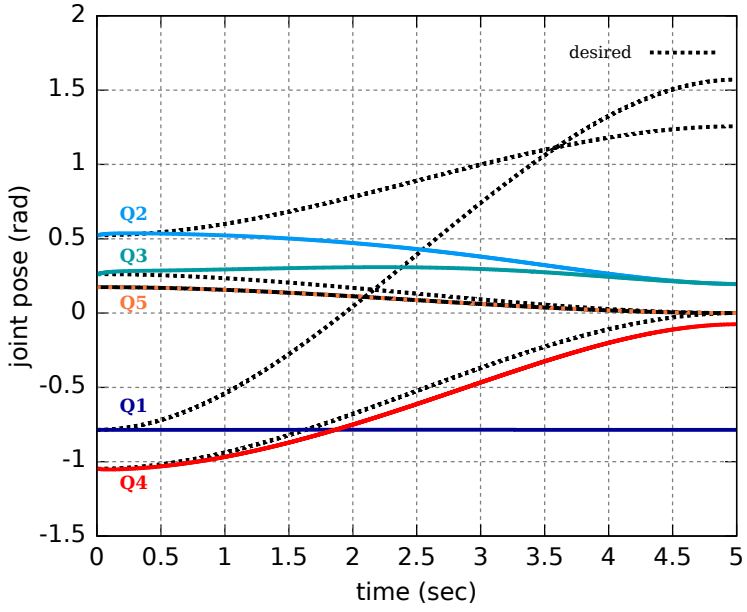
(a) Data on Z coordinates. In the first case (green), Z is tracking a polynomial trajectory and is controlled using F^{ext} impedance controller but no posture trajectories are imposed. In the second case (blue), the posture trajectories are present but a priority is given to end-effector constraint.



(b) Changes in orientation about ZYX axes. In the first case (red), X and Y are constrained and Z is tracking a trajectory using F^{ext} impedance controller but no posture constraint are present. In the second case (black) four joints are tracking trajectories. Here $\boldsymbol{\alpha}^\circ$ is about Z, $\boldsymbol{\beta}^\circ$ about Y and $\boldsymbol{\gamma}^\circ$ about X.



(a) Joint motion profiles in the presence of the controlled end-effector constraints. The joints are 'free'.



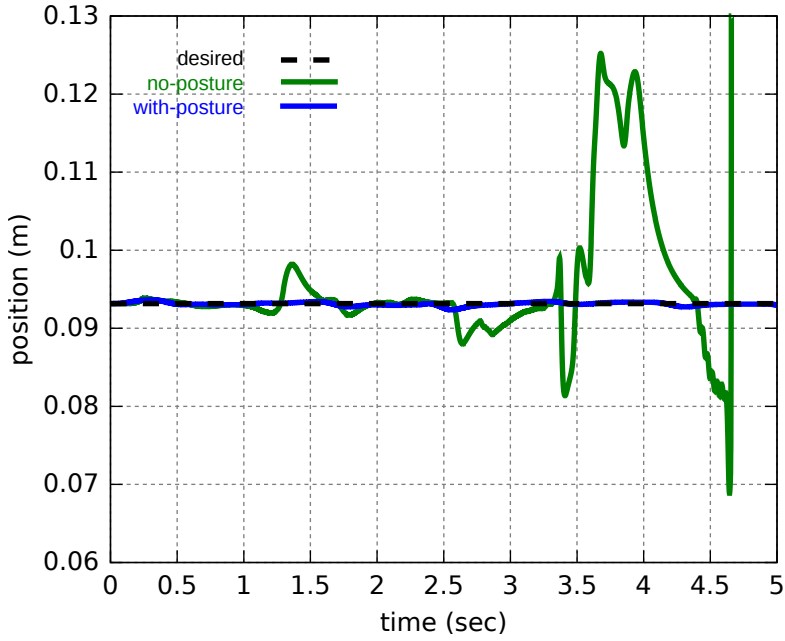
(b) Joint motion profiles when the robot is overconstrained, four joint trajectories, one Cartesian trajectory and two Cartesian acceleration constraints.

Figure 4.21: A comparison of the acceleration energy and impedance based end-effector controllers for the 5-DoF serial chain under the conflicting constraints.

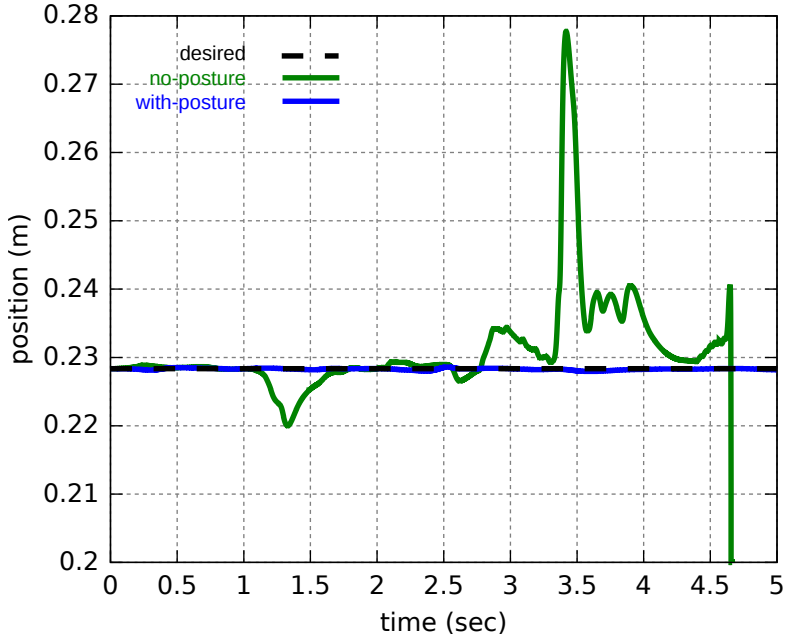
Setup 4: controlled end-effector motion of a redundant 7-DoF spatial robot under gravity, and with non-conflicting constraints: in this setup the motion of the end-effector of a 7-DoF Baxter robot, [Rethink Robotics, 2012], is constrained in its linear X and Y degrees of freedom, and its Z rotational degree of freedom. The specification for this task is given as

$$\boldsymbol{A}_N = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix}, \quad \boldsymbol{b}_N = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}. \tag{4.15}$$

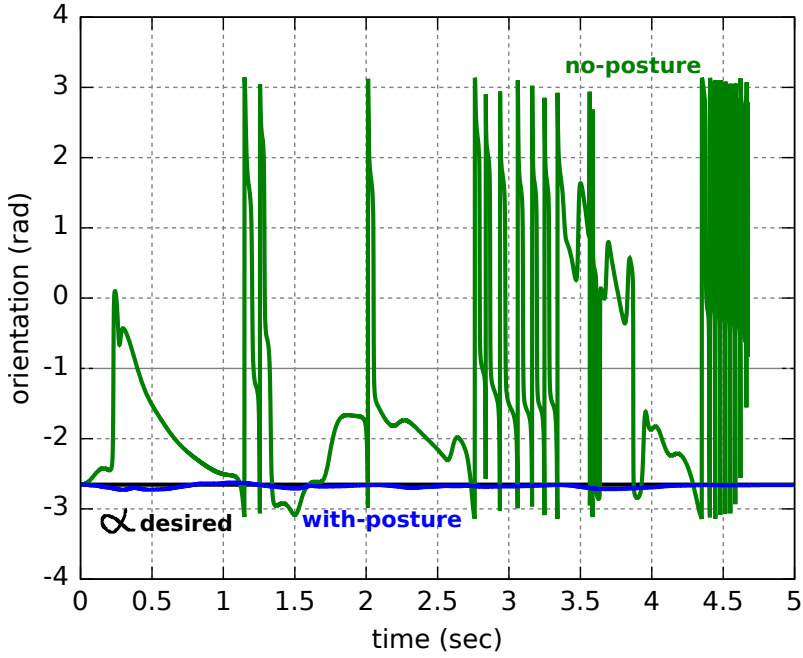
According to this motion specification the robot is only to move along its Z axis, while keeping its orientation about the axis constant. Figures 4.10a and 4.10b show X and Y coordinates of the end-effector, respectively. They compare two configurations: (i) in the first configuration, there are only constraints on the end-effector and none in the joint space. As can be seen, the use of controllers did not improve stability of the motion. This does not necessarily mean that the controllers are poorly tuned, but can also have been caused by the 'jump' in orientation coordinates as depicted in Figures 4.10c and 4.10d, which themselves can be due to poorly chosen specific coordinate representation, i.e., Euler angles. Here, more simulation data using such continuous representations as quaternions and rotation matrices is required, (ii) in the second configuration, in addition to the end-effector constraints, two of the outermost joint in the kinematic chain (closest to the end-effector segment) are to track imposed trajectories, Figure 4.10e. The introduction of these trajectories greatly improves tracking dynamics of all the constraints. The reason for choosing specifically these two joint is the fact that their motion contributions are not strongly weighted by the robot's inertia.



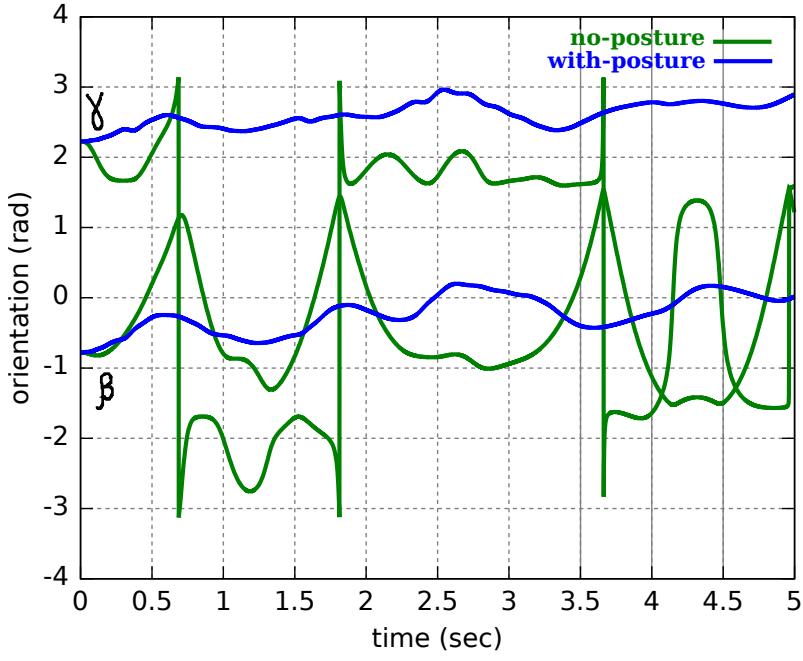
(a) X coordinate of the end-effector when only the end-effector constraints are presents (green) and when additional joint space trajectories are prescribed (blue).



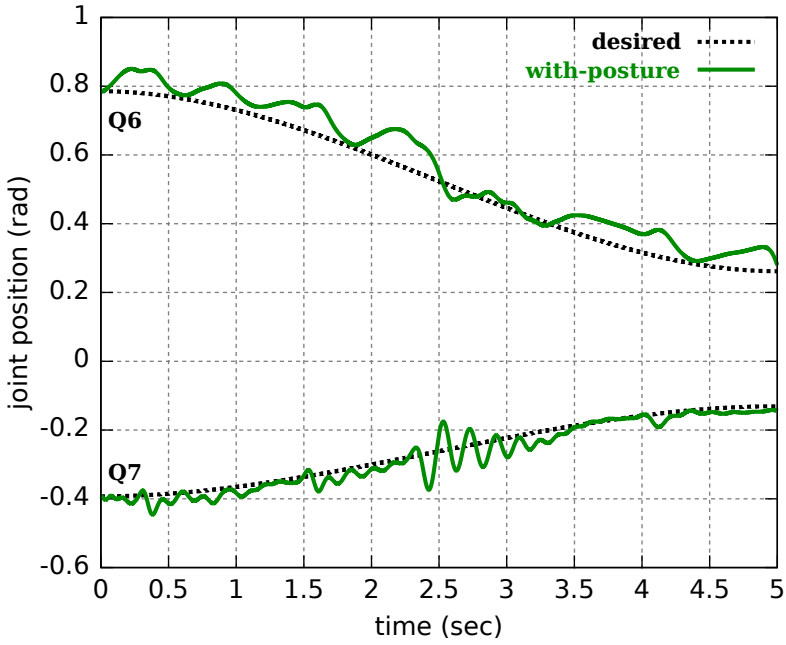
(b) Y coordinate of the end-effector when only the end-effector constraints are presents (green) and when additional joint space trajectories are prescribed (blue).



(a) Dynamics of the constraint rotational DoF about the Z axis.



(b) Dynamics of the orientations changes about the X axis, $\gamma°$ and the Y axis, $\beta°$.



(a) The motion profiles of the last two joints in the kinematic chain. The joints are controlled to track trajectories. The configuration uses a computed-torque controller.

Figure 4.24: A comparison of constraint stabilization using impedance and joint space controllers for a 7-DoF manipulator model. All data represent non-conflicting constraint configurations with a priority on satisfying the end-effector acceleration contraints.

4.3 Discussions and conclusions

This chapter presented how the existing constraint solvers are related to a constrained optimization problem. Specifically, it looked into an optimization-based formulation of the constrained dynamics solver from Chapter 3. It is shown that the constraint solvers often compute a solution to the constrained optimization problem, where a cost function is determined by one of the fundamental principles of mechanics. The choice of the latter determines what quantity is being optimized and is often given as a quadratic expression that includes weighed product of a motion and a force space physical quantities. For example, the cost function can be a kinetic energy/instantaneous power (a product of momentum and screw velocity twist), an acceleration energy (a product of screw force wrench and screw acceleration twist), etc. The weighing of the motion and force products is often performed by the spatial inertia term. The choice of such a metric weighing term enables physically consistent outcome of the constrained dynamics.

It is shown that the result of the optimization computations does not generate a stable motion. This follows from the fact that in order to obtain the acceleration level constraints, the position level holonomic constraints are often directly differentiated. This causes the loss of the position and velocity level constraint information, which results in a numerical drift problem. A common approach that is adopted to cope with this issue is changing the shape of the linear constraint differential equation with diverging solution(s) to the one whose solution(s) converge(s) as in Equation (4.5). In multi-body dynamics, the additional negative terms in this equation are known as Baumgarte stabilization terms. In the context of robotics, most conventional control approaches, impedance control, computed torque control, and other PD and PID controllers have this stabilizing structure.

The chapter also presented the results of the validation of the *implementations* of the constrained hybrid dynamics and stabilizing controllers. The validation is performed in simulation for multiple setups that involved various kinematic chain structures. It is shown that the control architecture built around the constrained hybrid dynamics does not differ from that when the unconstrained dynamics are used. It is shown that the current implementation supports prioritization and weighing of the controllers as outlined in Section 3.5.3 and most of the time their behaviors are as expected. Most of the simulations have been performed on serial kinematic chains with the end-effector being frequently constrained in translational degrees of freedom. Therefore, other constraint configurations, involving kinematic trees and rotational degrees of freedom, as outlined by the theory, but not implemented yet need to be tested in the future. Furthermore, current implementation uses one specific coordinate representation for various

geometric relations involved. As it is shown in simulations, Figures 4.10d and 4.10e, the choice of the specific coordinate may not always be the right one for the specified task. Therefore, a lot more effort is required to make the implementation truly coordinate representation independent as is outlined in Chapter 3. With respect to the over-constrained cases, the simulations show that it is not always possible to influence the outcome of the prioritization and weighing toward the desired performance by just changing the values of the weights or order of the computations in the sweeps. Therefore, more research is required to check how the proposed control approaches in this thesis relate to and favor against other existing task prioritization and weighing-based approaches out there.

Chapter 5

Examples of implementation of motion control stack

5.1 Introduction

Chapters 2 and 3 presented the methods to specify the *robot* and the *solver* using semantic models. Furthermore, Chapter 3 considered the specification of some types of the *task constraints* in a combination with the dynamics solver. Chapter 4 covered the conventional approaches to cope with the *constraint control* problem. So far, the functionalities and the design of each of these constrained task components were discussed separately. The results of this chapter should be viewed as the cumulative of the research presented so far. It discusses how the task components interact with each other, both on the functional level and in the context of motion programming stack, in order to implement the motion task as it is presented in Figure 1.2.

Figure 5.1 shows a conceptual organization of the motion programming stack that implements the symbolic task in Figure 1.2. It illustrates what level of the stack the programmatic representations of the components belong to. From the functional point of view, this motion programming stack implements WBCA which is at the core of many robot motion task control frameworks. In the context of this research WBCA refers to an architecture that enables a control of a robotic system whose Cartesian space, joint space, and sensor space coordinates are constrained simultaneously.

The following sections analyze the architecture of WBCA, its constituting

components and their underlying mathematical formulations. Finally, these sections show how a motion task based on WBCA can programmatically be realized using the motion stack semantic models and their DSLs implementations.

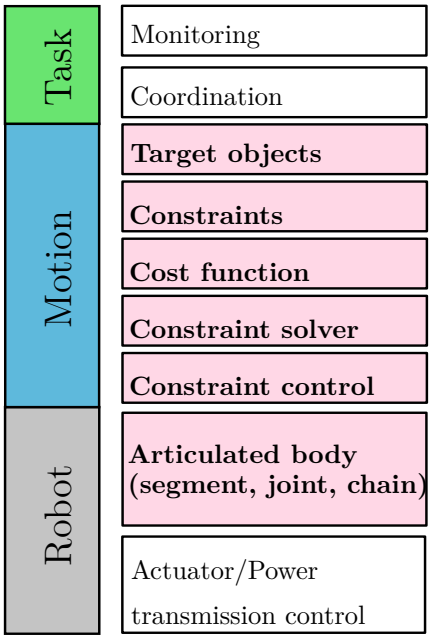


Figure 5.1: An implementation of the WBCA can be expressed using a composition of various DSLs that are part of the (motion)task programming stack.

Contributions This chapter discusses how WBCA can be realized in software using the formulation of the motion programming stack introduced in Chapters 2 and 3. Specifically, it presents a number of example task control architectures and their implementations. (i) It shows how WBCA can be implemented using the specialization of a hybrid dynamics solver on a mobile arm platform; (ii) it develops an optimization-based predictive algorithm that can adapt the gains of the constrain controllers. This algorithm is then used with the hybrid dynamics solver to realize WBCA for the tasks with conflicting and non-conflicting constraints.

5.2 General structure of WBCA

The constrained system in Equation (1.4) and the one with the optimal control formulation in Equation (4.1) addressed multi-task cases with either joint/posture space or Cartesian space constraints and controllers [Peters et al., 2008]. These included computed-torque-like control techniques [An et al., 1987] for posture space constraints, as well as indirect force control using robot's impedance formulation [Hogan, 1985] for the Cartesian space constraints. All of these methods have a structure similar to that of Baumgarte stabilization [Laulusa and Bauchau, 2008, Lin and Chen, 2011], Chapter 4.

In addition to the joint and the Cartesian space constraints, most of the contemporary task control framework implementations can also cope with the constraints specified in the sensor space of the robot. A class of the problems that consider such cases is visual servoing. In visual servoing the task of the robot is to track the motion of the objects (features) in the sensor space, often in a camera image [Espiau et al., 1992, De Laet et al., 2012, Hutchinson et al., 1996]. Mathematically, one could express a constrained system that is under posture, Cartesian (or end-effector), and sensor space constraints as

$$\begin{aligned} \boldsymbol{M}(\boldsymbol{q})\ddot{\boldsymbol{q}} &= \boldsymbol{\tau}_{\boldsymbol{a}}^{\boldsymbol{p}}(\boldsymbol{q}) - \boldsymbol{\tau}_{\boldsymbol{c}}^{\boldsymbol{ee}}(\boldsymbol{q}) - \boldsymbol{\tau}_{\boldsymbol{c}}^{\boldsymbol{s}}(\boldsymbol{q}) - \boldsymbol{C}(\boldsymbol{q}, \dot{\boldsymbol{q}}) \\ \boldsymbol{h}^{\boldsymbol{p}}(\boldsymbol{q}) = \boldsymbol{0} &\Rightarrow \hat{\boldsymbol{J}}_{\boldsymbol{c}}^{\boldsymbol{p}}\ddot{\boldsymbol{q}} = \hat{\boldsymbol{b}}^{\boldsymbol{p}} \quad \text{posture space,} \\ \boldsymbol{h}^{\boldsymbol{ee}}(\boldsymbol{q}) = \boldsymbol{0} &\Rightarrow \hat{\boldsymbol{J}}_{\boldsymbol{c}}^{\boldsymbol{ee}}\ddot{\boldsymbol{q}} = \hat{\boldsymbol{b}}_{\boldsymbol{c}}^{\boldsymbol{ee}} \quad \text{Cartesian space,} \\ \boldsymbol{h}^{\boldsymbol{s}}(\boldsymbol{q}) = \boldsymbol{0} &\Rightarrow \hat{\boldsymbol{J}}_{\boldsymbol{c}}^{\boldsymbol{s}}\ddot{\boldsymbol{q}} = \hat{\boldsymbol{b}}^{\boldsymbol{s}} \quad \text{sensor space.} \end{aligned} \tag{5.1}$$

Here, the input to the motion of the robot is decomposed into the inputs to address the constraints in each of the three constraint spaces. Note that the Equation (5.1) does not involve the control terms yet. The goal of the task control framework is to find control inputs $\boldsymbol{\tau}_{\boldsymbol{a}}^{\boldsymbol{p}}(\boldsymbol{q})$, $\boldsymbol{\tau}_{\boldsymbol{c}}^{\boldsymbol{ee}}(\boldsymbol{q})$ and $\boldsymbol{\tau}_{\boldsymbol{c}}^{\boldsymbol{s}}(\boldsymbol{q})$ that do not interfere with each other and satisfy their constraints as closely as possible. Such control inputs are usually referred to as *dynamically consistently decoupled* [Peters et al., 2008, Sentis, 2007] and the control framework is said to implement the *whole-body control architecture*. Note the similarity between Equation (5.1), Equation (1.4), and Equation (3.1). The latter two are explicitly accounting for 'one constraint'. Depending on where this constraint is specified $\boldsymbol{J}_c$ performs the mapping either from the end-effector space and is equivalent to $\boldsymbol{J}_r$, Equation (4.4) or maps to the posture/joint space itself and is an identity matrix, respectively. In Chapter (4), the kinematic chains in the simulations were configured with constraints in posture and Cartesian spaces, thus distributing available degrees of freedom of the robot to satisfy the constraints across two

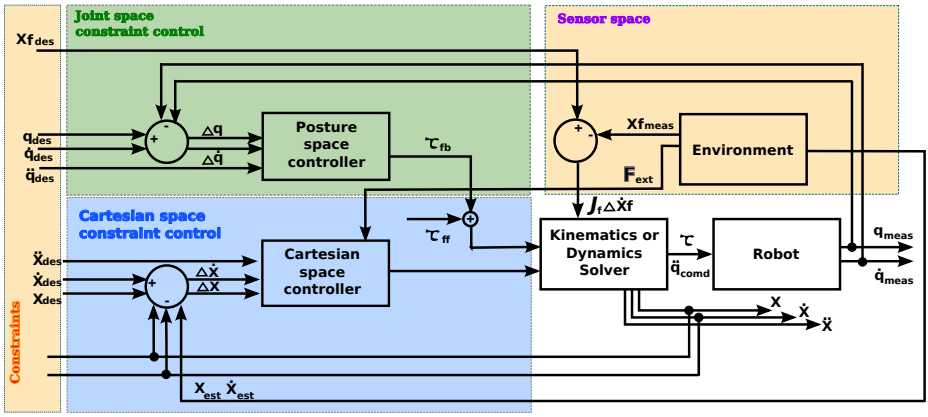


Figure 5.2: The block diagram shows a generic whole-body control architecture. It summarizes that any whole-body control approach consists of constraints, constraint controllers in one of the constraint spaces and a kinematic or a dynamic model solver. Here $\boldsymbol{Xf}_{(.)}$ are feature poses in the sensor space and subscripts *des*, *est*, and *meas* stand for desired, estimated, and measured respectively.

spaces. These are then controlled according to Equation (4.5). Equation (5.1) generalizes Equation (4.5) to address the constraints specified in sensor space as well, thus distributing available degrees of freedom of the robot to satisfy the constraints across three spaces. One can view WBCA as the generalization of the classical control architectures with a solver as the central component. This allows a design of various task control loops around the *solver* component, e.g., hybrid dynamics or inverse kinematics. Such whole-body control architecture in general case can be depicted as in the Figure 5.2. The block diagram shows several possible control loops formed around the solver component. This shows that the control loops can be closed in either posture, Cartesian or sensor spaces, depending on where the constraints were specified. All these constraints require a mapping of the motions (i.e., coordinates) in each space onto robot generalized coordinates in which the robot is usually commanded. This mapping is performed by the Jacobian matrices, as depicted in Equation (5.1).

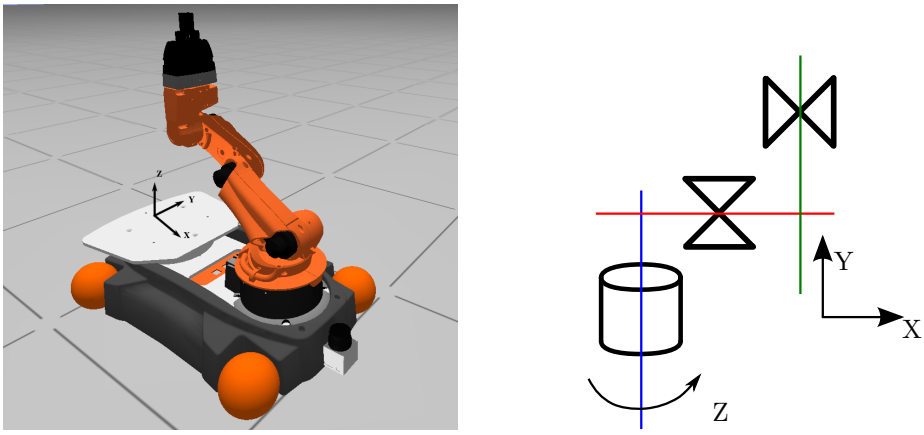
WBCA implies that all the specified constraints can be active simultaneously. The example for such a case is demonstrated in [Vanthienen et al., 2011a]. Physically speaking, each constraint keeps a number of degrees of freedom of the robot busy. Therefore, it can be that there are not enough free robot degrees of freedom to satisfy the constraints, thus the tasks. Such situations lead to a performance degradation and are called *conflicting*. In order to address this problem, most of WBCA frameworks implement some form of *scheduling or*

coordination policy. This policy can be implemented as a separate component of the task, or be part of the solver component, as is explained in Section 3.5.3. In Section 3.5.3 and Section 4.2, such a policy is part of the solver component and the scheduling policy is based on priorities. One could also implement a control architecture that switches off and on a specific set of constraint control loops depending on the application context, as is done in Figure 5.8. This solution relies on the coordination policy to cope with the over-constrained situation. In robotics, a set of constraints, controllers and coordination policies that are valid in a specific context are often referred to as *skills* or *sub-tasks*.

Many frameworks, specifically their solvers opt for the scheduling policy based on priorities, because it relies on the physical capabilities of the system that are taken into account during the constraint resolution [Nakamura et al., 1987]. This point is complementary to the dynamically consistent decoupling discussed before. It also requires that the constraint inputs to the solver should have a consistent formulation that reflects the schedules of each task constraint. Such a formulation is usually achieved by projecting the tasks of a 'lesser' importance or priority onto the nullspace of the Jacobian matrix relating the constraint space and the robot's generalized coordinate space. For instance, in [Khatib et al., 2008, Sentis, 2007], the authors introduce a posture and a constraint consistent nullspace projection operator. It enables decoupling of the primary task inputs from those of the posture and other constraints such as a distance to an obstacle, for example. This process can be repeated on many constraints and one can invert priorities which imply a different set of projections.

Consequently, a formulation and a decomposition of WBCA into the solver, controllers, and constraints components and associating them with the appropriate constraint space,as in Figure 5.2, simplifies the identification of its *architectural variabilities*. The latter facilitates a software realization of each component using the motion stack and component specification DSLs. The decomposition and consequent analysis of WBCA components is achieved by applying a feature oriented domain analysis method [Kang et al., 1990, Czarnecki and Eisenecker, 2000], which relies on such modelling principles as aggregation/decomposition, generalization/specialization and parametrization, Section 1.5.1.

5.3 WBCA with the inverse kinematics and dynamics solvers and the Cartesian impedance controller for the youBot mobile manipulator robot



(a) (b)

Figure 5.3: Shows youBot mobile manipulator platform. (a) The robot has a 5-DoF arm and omni-directionally actuated mobile base. (b) serial 3-DoF joint representation of the robot's base.

This demonstration shows how a mobile arm platform can be commanded to move its end-effector to some point which is not directly in the reachability space of the arm in the XY plane. The XY plane is the plane where the robot base can move freely. The demonstration uses a youBot robot platform, which has a 3-DoF redundant base with four active and four passive joints and a 5-DoF arm. A graphical representation of the robot on the plane is depicted in Figure 5.3(a). For this setup, the task is to reach some pose in space with the robot's end-effector. The pose can be specified anywhere as long as its position value along the Z axis is not bigger than the robot's height (base and arm combined).

This task can be realized in multiple ways. One could specify a Cartesian trajectory between the initial pose of the end-effecotr and the pose it needs to reach and perform controlled motion along this trajectory. This motion itself can be decomposed into the motion of the base and the arm. In many similar robotics applications, the robot base's motion is treated separately. That is,

the base is first navigated (controlled motion along the trajectory) to some pose in the vicinity of the desired end-effector pose and the arm performs another controlled motion along the trajectory for the remaining portion of the pose. Here, the base and the arm are treated as two separate systems. In the context of WBCA, the mobile arm can be treated as a single kinematic chain. The desired Cartesian pose will then be a Cartesian space holonomic constraint on the motion of this kinematic chain. This motion can then be controlled using a velocity or an acceleration resolved control schema, if an appropriate solver is available. For the domain specific solver as the one in Chapter 3, the structure of the kinematic chain is important. In the case of the youBot robot, this structure is a kinematic tree, because the base has four joints with one actively actuated and one passive DoF (because of the mecanum wheels). Therefore, the solver needs to be able to cope with such robot structures, which can lead to complex computations. One can also simplify this model and use instead an equivalent single 3-DoF joint for the base or three 1-DoF joints connected serially. The former requires that the solver can deal with joints with more than one DoFs. This example opts for the model of the robot base with three 1-DoF joints, which forms an RPP manipulator as depicted in Figure 5.3(b).

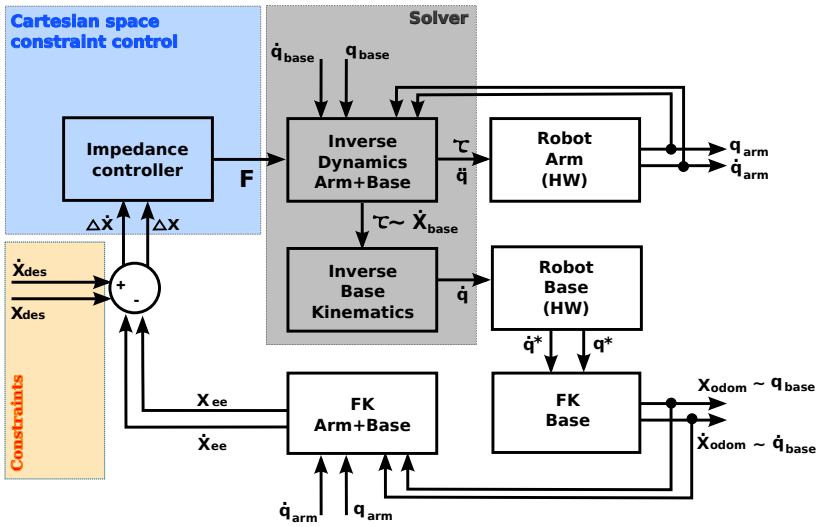


Figure 5.4: This control block diagram describes a dynamically compliant motion control of the youBot mobile platform. The motion of the platform is generated by specifying a set-point in the Cartesian space of the manipulator. The distance to this set-point is used to generate an external force which is an input to the inverse dynamics solver. Here subscripts *ee* and *odom* stand for an end-effector and an odometry.

Combined with the 5-DoF (5xR) arm serial chain (fig. 4.4(b)), the simplified model of the youBot robot becomes 8-DoF a serial kinematic chain with the joint configuration of (RPP)+(5xR).

In this example, the motion of the robot is triggered by a controlled virtual external screw force wrench, which can be constant or a function of the pose difference of the robot to the goal and the robot's instantaneous twist. That is, it is a simplified indirect force control. This controlled wrench then serves as an input to the solver. Since the base has four actuated real joints, one has to make sure that the input values computed by the solver are mapped correctly. This is achieved by the inverse base kinematics or the odometry. The complete WBCA configuration for the task is depicted in Figure 5.4.

The inverse dynamics solver computes torques for the modeled base joints. In the physical world they correspond to the wrench components $(\boldsymbol{f}_X, \boldsymbol{f}_Y, \boldsymbol{\tau}_Z)$ applied on the base. Therefore, they need to be projected on each wheel to obtain the real torque values at each of the four real base joints. If the base is commanded in a velocity mode, then the wrench needs to be converted to the base twist with components $(\boldsymbol{\nu}_X, \boldsymbol{\nu}_Y, \boldsymbol{\omega}_Z)$. It will then require the solution of the inverse kinematics problem to find each joint's velocity as depicted in the solver component in Figure 5.4. Since the solvers require the values of the robot state $(\boldsymbol{q}, \dot{\boldsymbol{q}})$, the forward problem also needs to be solved to obtain the odometry values for the base. One could summarize that in this demonstration, there are a number of solvers working together. Of course, these solvers could have been replaced by a single solver, which could deal with multiple DoF joints or be based on numerical techniques from optimization. But then its implementation complexity would increase significantly.

5.4 WBCA with the hybrid dynamics solver and a controller with adaptive gains for the youBot manipulator robot

The control approaches and the control loop scheme in Figure 4.3 discussed above can be applied to solve a broad range of "offline pre-defined" robotic tasks. But in reality most of these tasks may change their conditions during run-time. This in turn may lead to a poor system performance or even degrade the stability of the controllers. Some key reasons that lead to such situations are (i) uncertainties and variation in the friction and inertia parameters of the dynamic model of the robot, and (ii) the controller gains, i.e., inertia, stiffness, damping factors, are robot configuration dependent ([Albu-Schäffer et al., 2007b] provides a model-

based solution to the latter at a computational cost of $\mathcal{O}(N^3)$). In classical control, when choosing the value of these parameters, one defines a configuration interval in which some constant parameter values achieve the best results. If the system needs to go through a different configuration interval the controller gains need to be adapted accordingly. This is usually realized by devising a hybrid control system with a "high level" finite automaton that encodes the state or configuration of interval switches [Egerstedt and Hu, 2002]. Ideally, the controller should have some mechanism to adapt the gains as the robot's configuration changes. There are many approaches ranging from adaptive to optimal control that can be used to realize such adaptive control behavior, as summarized in Section 1.5.4.

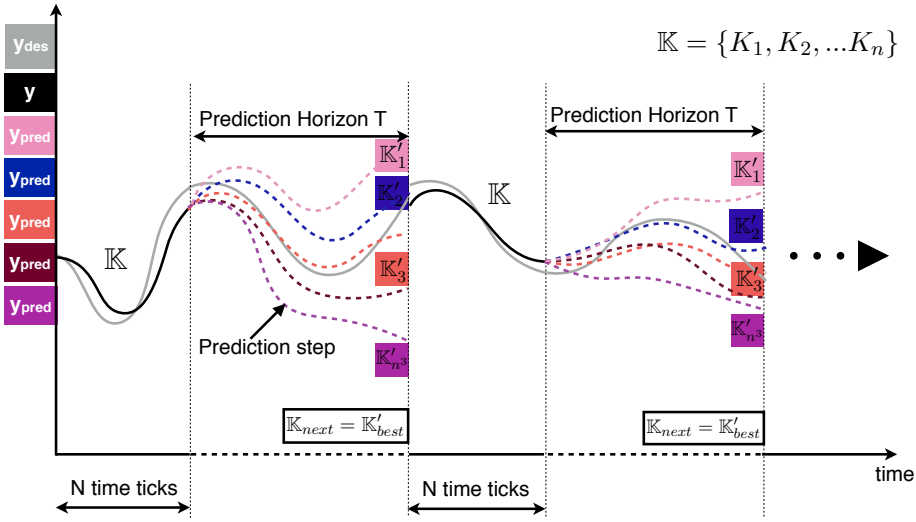


Figure 5.5: The performance of the controllers is adapted by the adaptation block, which uses a prediction algorithm and cost functions to tune the controller gains. Here **K** is a set of gains that are being changed. The number of sets is defined by the allowed variations of the gain values (color coded). For instance in the experiment for circle tracking, gains were chosen to vary 10% from their starting some nominal value $(K + 0.1K), K, (K - 0.1K)$.

This section presents an algorithm that allows to adapt gains of the constraint controllers based on the optimization of some cost function. The adaptation process occurs by taking into account the future process dynamics of the robot. It resembles the receding horizon prediction model used in MPC. The algorithm adapts the gain(s) of an already designed controller based on the forecasted system behavior. This is unlike the approaches discussed in Section 1.5.4, where

the outcome of the optimization is the optimal control input itself.

The idea behind this demonstration is to show that most task control paradigms can be viewed as the special instances of the more generic WBCA. Therefore, one can reconfigure or replace core components of WBCA to adapt to the requirements of the task. Additionally, one would like to combine a domain specific hybrid dynamics solver with an optimization-based controller. The hypothesis is that such a combination will have advantages over the domain-invariant purely optimization-based solver, because the latter always solves ODE of the robot motion model numerically, in addition to searching for the optimal control inputs.

5.4.1 Controller based on predictive gain adaptation*

In this algorithm, the prediction process is used to simulate a set of possible state trajectories that the system model (i.e robot motion model) might undergo with the given configuration of the controllers and constraints. That is, each trajectory is the outcome of the robot control loop with the given constraints. The difference to the conventional scheme is that each trajectory is generated by varying the *gains* of the constraint controllers and evaluating their performance based on some cost function.

Let's explain this procedure on the basis of Figure 5.5. The system starts with a set of predefined control loops with an 'acceptable' performance (i.e., stable but with a poor tracking behavior). At some point $\boldsymbol{t_1}$ during the run-time the prediction algorithm starts searching for the best possible set of gains. In this prediction phase $\boldsymbol{T}$, which (can) run(s) in parallel to other system activities, possible (observable) system state trajectories are generated by computing system dynamics, controllers, and cost functions. The number of the sets of gains, $\boldsymbol{K_n}$, to be tried out is given in advance. This is done by defining allowed variations of the gain parameters in each set. Such a strategy will not achieve a globally optimal solution, but will lead to locally sub-optimal results. These are acceptable in service robotics tasks which are often to be compliant rather than follow an exact geometric trajectory. But has to note that this is also a big limitation of the current implementation, and requires the application developer to account for such an appropriate set of gains. For instance in Figure 5.6, the *stiffness parameters*, $\boldsymbol{K_{P^*}}$ in computed torque, impedance, and acceleration energy controllers can be allowed to change 10% from their nominal initial values, $\{0.9, 1, 1.1\}\boldsymbol{K_{prev}}$. This gives 27 variation sets $\boldsymbol{K_n}$ leading to an equal number of the trajectories to evaluate. The similar trajectories are shown in color-codes in Figure 5.5 during the prediction horizon.

*The content of this section is partially based on the research presented in [Copejans, 2014]

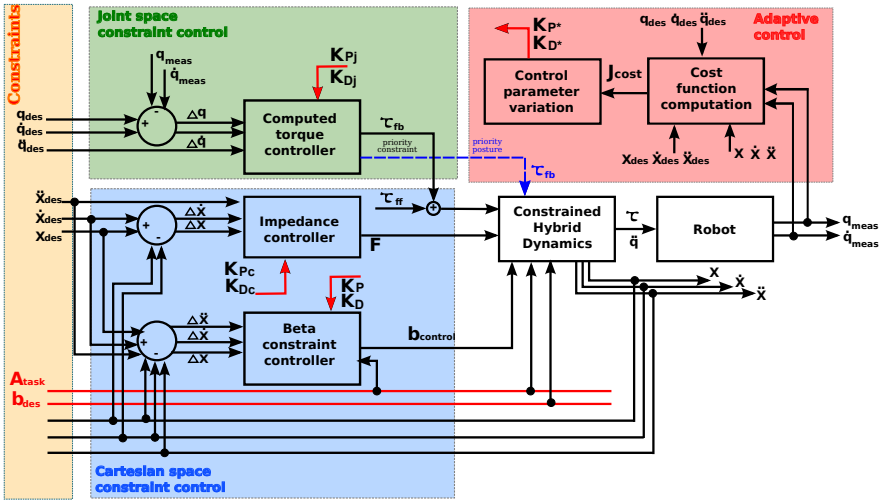


Figure 5.6: This control block diagram depicts the interaction of the conventional control loops with the receding horizon prediction-based adaptation algorithm. The algorithm adapts the gains of the conventional controllers based on the value of the cost function and a naive search strategy. Here $\boldsymbol{K_{P^*}}$ and $\boldsymbol{K_{D^*}}$ are predicted *stiffness* and *damping* gains of the respective controllers.

Then, based on the value of the cost function, the best set is chosen. The cost function has the form of the weighted square of some physical quantity. Some examples of the cost functions that were tested are given in Table 5.1. Here $\boldsymbol{Q}$ are square weighting matrices that have similar semantics as those described in [Hollerbach and Suh, 1987] and discussed in Chapter 4. The weighted quadratic form of the cost functions is physically consistent and is justifiable from the physical point of view in the context of controlled mechanical systems. This is also in line with the principle of Gauss and has also been discussed in [Peters et al., 2008, Kalaba et al., 2004, Udwadia and Kalaba, 2002].

After the prediction phase, the control loops using the chosen best set of the gains are run for some period of time, indicated by $\boldsymbol{N}$ in Figure 5.5. Then the procedure is repeated till the task execution time runs out. Algorithm B.1 in Appendix A describes this procedure in a sequential form. A control block diagram showing the integration of the adaptive gain evaluation blocks with WBCA is given in Figure 5.6. The adaptation block does not modify the original WBCA from Figure 4.3. But it might incur a performance penalty, not only because of the cost function computations, but also the dynamics

Penalized behavior	**Cost function**
Position error	$E_{pos} = \Delta X^T Q_p \Delta X$
Velocity error	$E_{vel} = \Delta \dot{X}^T Q_v \Delta \dot{X}$
Acceleration error	$E_{acc} = \Delta \ddot{X}^T Q_a \Delta \ddot{X}$
Joint position error	$E_{jointpose} = \Delta q^T Q_j \Delta q$
Torque magnitude	$E_{jointtorque} = \tau^T Q_\tau \tau$
Control gain gradient	$E_{controlgain} = (K(t_i) - K(t_{i-1})^T) Q_{gain} (K(t_i) - K(t_{i-1}))$
Combination of all	$E_{total} = a \cdot E_{pos} + b \cdot E_{vel} + c \cdot E_{acc} + d \cdot E_{jointpose} + f \cdot E_{jointtorque} + g \cdot E_{controlgain}$

Table 5.1: Cost functions and the errors $\boldsymbol{E}_{(\cdot)}$ they penalize. Here, a, b, c, d, f and g are weights.

computations during the prediction time. The interesting point of the algorithm is that it searches for the solutions in the controller parameter space. This ensures that there is a stable control input, regardless of the search result. An application programmer has to ensure that all the gains in the chosen set of controller gains $\boldsymbol{K_N}$ result in stable behaviors. This is a big limitation of the current implementation. This is unlike other optimal control approaches, which directly search for the control inputs themselves. Therefore, not finding the optimum might lead to issues with the control, if the previous optimal inputs were discarded. A reader can find more details on the evaluation of the algorithm in different simulation and experimental setups with conflicting and non-conflicting constraints in Appendix B.

5.5 An implementation of WBCA using the motion programming stack

This section discusses how WBCA and the components of the demonstrations above are implemented using the DSLs of the motion programming stack. Furthermore, every model and functionality specified using these DSLs should have a software form that can be run on the robot's computational hardware. The implementations of WBCA demonstrations above use a component software framework. This enables an allocation of WBCA component functionalities to separate software components with clear responsibilities and that can interact

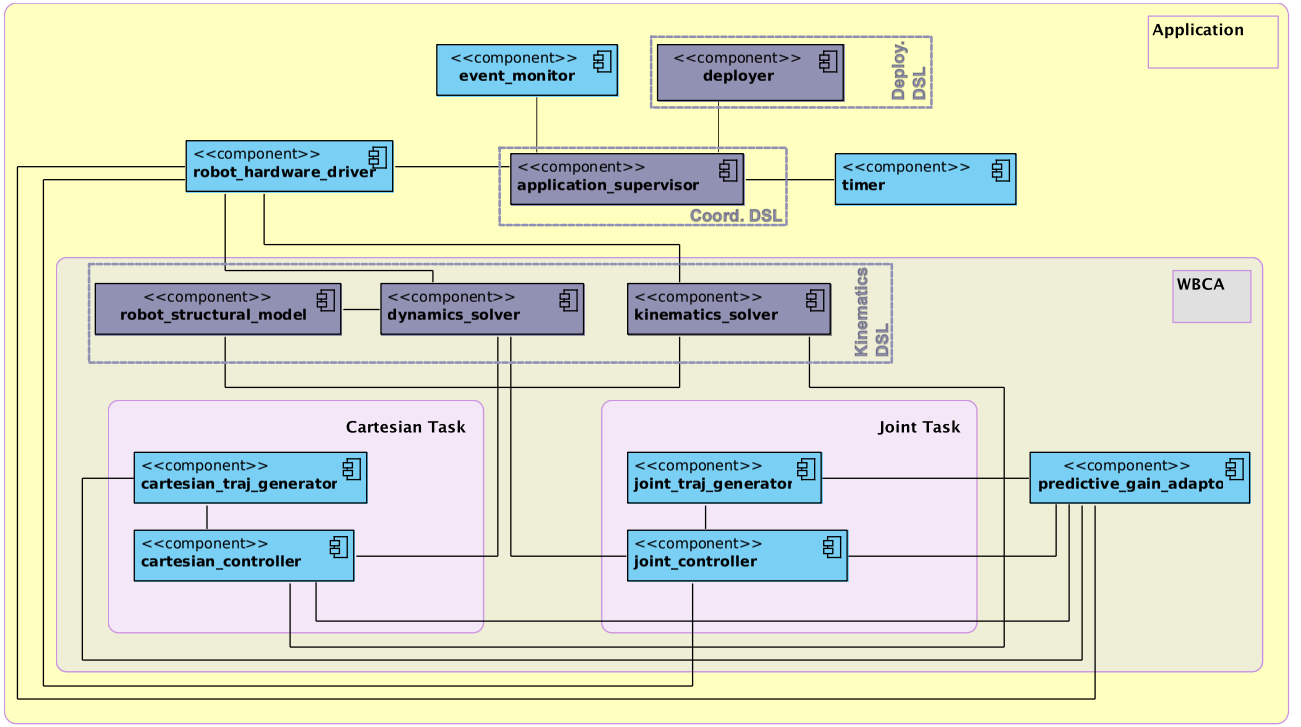


Figure 5.7: A software component architecture for the mobile manipulation application in Section 5.3 and WBCA using a predictive algorithm to adapt the controller gains in Section 5.4. The figure does not include all inter-connections. The components *deployer, supervisor*, and *solver* depicted in grey use different DSLs to express their functionalities. For instance, the functionality model of the application supervisor component shown in Figure 5.8 is specified using the rFSM DSL for the coordination [Klotzbücher and Bruyninckx, 2012].

with each other through ports. This is often referred to as the component software architecture. There are two aspects of such an architecture:

- *a structural aspect* - is concerned with the organization of the functionality of the control architecture, in this case WBCA, into interacting software components. The interactions among the components are established through the communication ports of the components. The ports are connected to each other through the connectors, which are often implemented using some communication middleware infrastructure. This trio of component, port and connector often have the same model for all the components of the software architecture. Therefore, many modern software component frameworks offer a tool-chain and a DSL that automate an implementation of these structural primitives. The tool-chain interprets the DSL specification of the components and generates what is known as a *stub code*. The stubs are empty component templates that can be *filled in with* the necessary robot *domain specific functionalities*. By the introduction of the motion programming stack, such a domain functionality is expressed using the DSLs for each domain. For example, the inverse dynamics (ID) software components is implemented in two stages. First, the component stub code is generated using the component specification DSL and then the functionality of the ID is filled in using the kinematics and dynamics DSL.
- *a behavioral aspect* - is concerned with the coordination and scheduling either within the components, e.g component's internal state coordination, or the software architecture wide. In the latter case, the task coordination or schedule in WBCA is reflected in the *run-time* coordination of the components in the software architecture.

After the structural and behavioral aspects of the software architecture for WBCA have been laid out, the architecture needs to be deployed on the computational environment (operating system + hardware) of the robot. The deployment process is often complex in itself and follows some deployment model as well [Hochgeschwender et al., 2013]. Therefore, it is also implemented using a DSL.

Figures 5.7 and 5.8 are the final products of the implementation of WBCA using the motion programming stack and the procedures explained above. Figure 5.7 distinguishes between the components that contribute to the functionality of WBCA and those that have utility functions such as synchronization, deployment, and robot interfacing. Furthermore, among WBCA components task relevant components that implement task constraints such as trajectories or set-points, and task constraint controllers are explicitly grouped together.

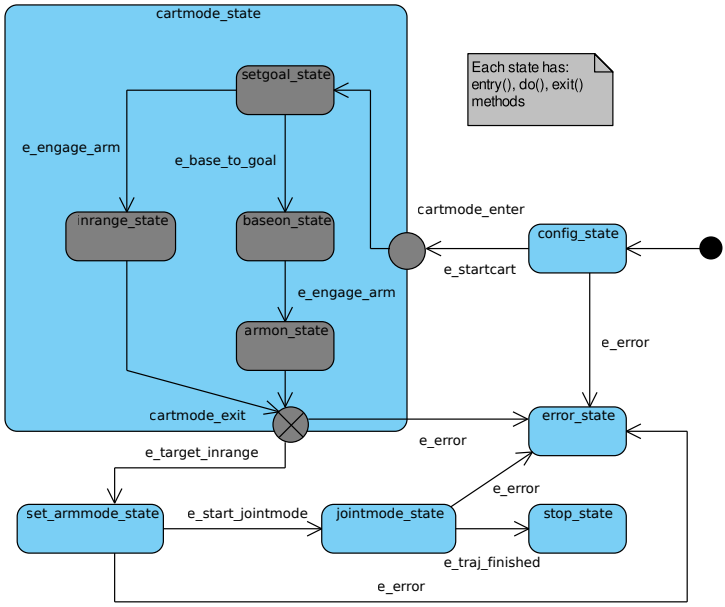


Figure 5.8: State machine diagram of the application that realizes the architecture in Figure 5.4. Here, the transitions between the states are triggered by the events $\boldsymbol{e}\_(\cdot)$.

Although it is desired and possible to express the functionalities of all components through motion stack DSLs, for the demonstrations in Sections 5.3 and 5.4 the DSLs that are developed in this thesis and are packaged with the component software framework are used. These are indicated in grey in Figure 5.7. The functionalities of the solver and robot structural model components that are part of WBCA are realized using DSL from Chapter 2. The application supervisor component coordinates WBCA components according to the state machine model depicted in Figure 5.8. This state machine model is implemented using rFSM DSL. Finally, the deployer component initializes and configures the properties of all components and their connections. It uses a deployment DSL that is part of the OROCOS RTT framework [Klotzbücher et al., 2010].

5.6 Task control applications: Linked data with semantics

The discussions so far focused on making the semantics of the models for kinematic structures and motion controllers explicit. Machine-readable representations for some of these models were also defined. But the full realization of the constrained task control applications requires similar efforts for other involved sub-domains, such as a constrained optimization, software components and architectures. Furthermore, the semantics and implementations of these models need to be explicitly *linked* to each other: for example, how a synchronization mechanism of the components in the robot control loop relates to its finite state machine implementation in some software component framework. In the demonstrations in Sections 5.3 and 5.4, this linking of the models and implementations took place implicitly and relied on the knowledge and experience of the author. But for a developer who cannot bring such background knowledge, all these sub-domain models will look like disjoint 'islands' of the models without direct connections. This section proposes a set of guidelines on how a constrained task control application can explicitly be represented in the form of linked models and implementations from the sub-domains. The approach relies on the principles of the 'Linked Data' as defined in [Bizer et al., 2009].

According to [Bizer et al., 2009, Heath and Bizer, 2011], 'Linked Data' or 'Web of Data' technically refers to data published on the Web in such a way that it is machine-readable, its meaning is explicitly defined, it is linked to other external data sets, and can in turn be linked to/from external data sets. More specifically, the statement implies that there should be three ingredients to define linked data: *the machine-readable models of the data*, *the semantics of the data*, and *the ability to reference other data sets with different contexts*. In particular, distinguishing the semantics of the data from their specific instances enables the definition of *explicit structures* on the models, such as *data hierarchies, conceptual taxonomies* and *ontologies*. Once such high-order semantic concepts are defined, the linked data they are associated with can be traversed in different ways and consequently, be reasoned on. A machine-readable form of such link data is frequently referred to as the Semantic Web [Lange, 2013, Bizer et al., 2009].

Recently, in robotics the topic of linked data has started shifting from just being relevant to the ontologies [Compton et al., 2013] and the Semantic Web. There are already demonstrations of physical robots that reason on platform capabilities expressed as the linked data and report whether some real world task can be completed or perform it directly. One such relevant contribution is Semantic Robot Description Language (SRDL) [Kunze et al., 2011, Chitta et al., 2012].

SRDL matches task specifications given to a robot by verifying the required capabilities and hardware components the robots should have to be able to complete the task. The authors distinguish four concepts, *robot, component, action*, and *capability*, which are at the core of SRDL and its inference mechanism. The authors use Web Ontology Language (OWL) to model and reason on taxonomies associated with these concepts. In its current form, the language is limited to adding the semantic information to already existing robot kinematic structure descriptions that are specified in Unified Robot Description Format (URDF) [Willow Garage, 2009]. The approach does not detail on possible links between the actions, the capabilities and their software implementations. Furthermore, it contains the limitations associated with URDF model itself, which implicitly includes a number of coordinate specific constraints.

The approach presented here adapts the formulations of the principles of Linked Data to the context of robot task control.

- The linked robot data has a graph model. The graph nodes represent resources (e.g., data, semantics, models) that belong to different subdomains. The links between the nodes represent semantic relations (or constraints) on the resources.
- Every resource and relation must be uniquely identifiable. The node and link identities can be in any format, including URI.
- The graph with its nodes and links, as well as the resources they represent should have a machine-readable format.
- There should be a mechanism (a query protocol) making it possible to look up useful robot task relevant information.

Every item in the list can be implemented using the existing Semantic Web technologies. For instance, the graph model of the linked data can be expressed in Resource Description Framework (RDF) [Manola et al., 2004]. RDF enables users to make statements about resources in the form of triplets: *subject, predicate*, and *object*. Here, the subject and the object represent resources, i.e., nodes in the graph, while the predicate represents the relationship/constraints on these resources, i.e., links in the graph. For instance, Figure 5.9 graphically visualizes the relationship between the robot platform and its constituting components, such as a segment, a link, a joint etc. There are a number of description languages[†] that can be used to express this RDF graph in a machine-readable form. These include the Turtle family of languages (N-Triples, Turtle,

[†]Also called a *serialization format* in the context of Web programming.

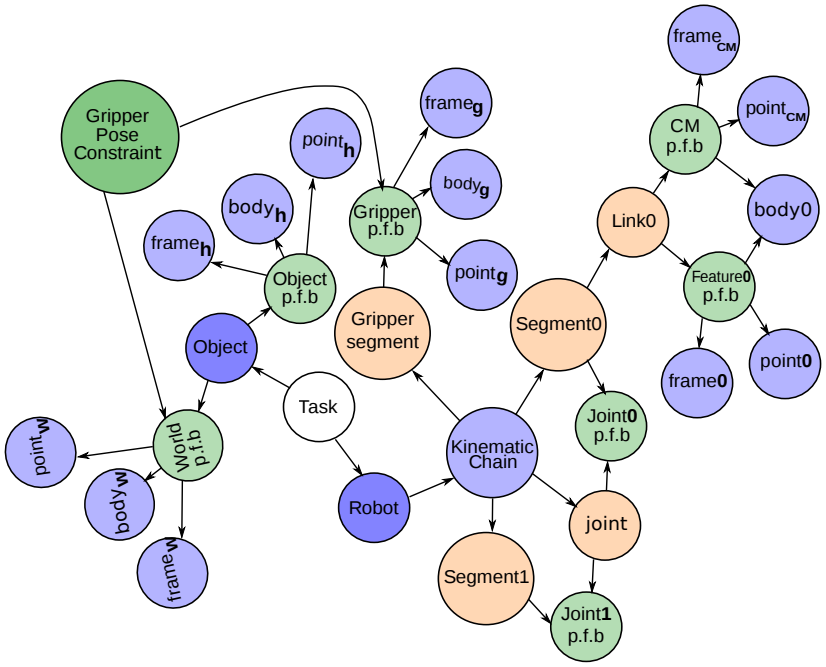


Figure 5.9: A linked data graph representation of the constrained task which shows the relation between its constituting primitives. Here the task has the components as depicted in Figure 1.2 and **p.f.b** stands for point, frame, body. In this example the Robot node is expanded to a node of a KinematicChain type, which itself has connections to other types of nodes such as Segment, Joint etc. In addition to a unique identity and a type, every node can have further properties defined by its type.

TriG and N-Quads) [Beckett et al., 2008], JSON-LD [Sporny et al., 2013] which uses JSON-based syntax [Zyp et al., 2013, Crockford, 2006], RDFa for HTML and XML embedding [Adida et al., 2012], and others. Here, the use-cases opted for the use of the JSON-LD format. In addition to being widely adopted and syntactically less verbose in comparison to others, JSON-LD has explicit language primitives for graph modeling. In JSON-LD representation, one creates a graph with nodes and links, which have specific types. In the robot task control, these types are one of the semantic models in the related sub-domains. For instance, one possible node type can be a *Segment* as defined in Section 2 or a finite state machine coordination model as depicted in Figure 5.8. These types do not have to be defined anew, but can be referenced from within the JSON-LD. In order to do this, every relevant type (i.e., model) should be encoded in a JSON or a JSON-LD schema. The schema serves as a *meta*

model that defines core structures, syntactic constraints and semantics of the models/types. Every instance of the models used in the specification of the graph is validated against these schema. Another interesting and important issue which JSON-LD exemplifies is the concept of *context*. The context, as its meaning suggests, defines the scope where the semantics and the values of the types and the properties of the nodes and the links are defined. Furthermore, it allows definitions of *vocabularies*. These are diverse specifications of the *same* semantic models. For example, a 'communication channel' and a 'connection' both represent the same thing, but may have been developed by different developers and given different notations. With the help of the vocabularies, one can choose which notation to use. Listing A.2 in Appendix A shows an excerpt of a JSON-LD representation of the graph model in Figure 5.9.

5.7 Discussions and conclusions

The material presented in this chapter is the convergence of the research presented in Chapters 2 and 3. The chapter re-introduced WBCA as the modular and reconfigurable architecture. Furthermore, it showed how WBCA relates to the motion stack, how its different functional components can systematically be implemented using the specification DSLs and eventually map to the specific software architecture on robot level.

The explicit separation of WBCA into blocks of constraint spaces, constraints, controllers and solvers allows one to systematically structure and implement any task specification problem as a mere reconfiguration of the same architectural model. This architectural organization emphasizes the fact that as long as the right type of the constraint solver is chosen, WBCA can cope with any type of constraints on any robot platform.

In order to justify this claim, the chapter presented two demonstrations using the mobile arm and the fixed arm platforms. The demonstration on the mobile arm involved a set-point and a trajectory constraint in the joint space of the arm and the Cartesian space of the base and the arm, respectively. This demonstration used a combination of inverse kinematics and dynamics solvers. The constraint controllers used a conventional Baumgarte form with the constant gains. The second setup based on the fixed arm platform used the constrained hybrid dynamics solver of Chapter 3 in combination with the controllers with the adaptive gains. The only additional differences of this setup to the one for the mobile arm platform are the additional adaptive control block and the structural robot models used by the solvers (fig. 5.6). The WBCA architecture did not undergo any further modifications. It is also

noteworthy that the setups do not need to use the domain specific solvers, but rely on the domain independent general purpose optimization solvers, e.g., qpOASES [Ferreau et al., 2008], IPOPT [for Operations Research, 2005], KNITRO [Ziena, 2015]. This would not introduce considerable modifications to the design of the architecture.

With respect to the programmatic realization of WBCA, there are two points to consider. These are the *DSLs* that help to *formulate* different WBCA components in computer readable form and *tool-chains* that can *interpret, convert*, and/or *execute* these formulations while accounting for the domain models. Since both the task control and programming involve multiple domains, structuring each in terms of the DSLs would not only improve their design and implementation, but also facilitate their programmatic *validation*. The latter improvement is of utmost importance for constructing physically correct robotic applications.

Current realization has few DSLs and tool-chains that take into account the semantic models, i.e., primitives and constraints, of their domains. Those are indicated in Figure 5.7. A part of the future efforts includes the formalization of such semantic models to express the geometric constraints, e.g., an orthogonality or a path, and the controllers. These models will then be used to implement the DSLs and the tool-chains that will allow one to correctly specify and realize these components. The future goal is to integrate several such language tool-chains that will enable correct by construction implementations of WBCA architecture.

Finally, one of the most important messages of this chapter is that a programming the robotic applications is not just the definitions of a set of abstractions. It either requires that an application developer understand the innate semantic models involved or the tooling infrastructure that supports such models in a systematic way.

Chapter 6

Discussions and conclusions

This doctoral thesis sets out to improve the development of robot task control applications, by providing a systematic and a formal foundation to the currently often ambiguous practice of the robot, task and software modelling, and specification. From a functional point of view, little is added to the large body of research and supporting software implementations that already exists. But, although the development process of the motion control tasks is rather repetitive and well-understood, the state of the art still has lots of difficulties to ensure correct-by-construction development of such tasks, and even more to ensure that the software components provided by independent development groups can be integrated automatically via formal checks of their interface semantics. In this respect, this thesis identified and researched two complementary topics that can directly contribute to the improvement of correct-by-construction development of motion task control applications: *(i)* composable semantic models, and *(*ii) variability points in task control architectures.

Composable semantic models for robot kinematic chains and computations:

By analyzing existing approaches to task control, it is identified that many task control applications can be expressed in terms of four constituents:

- target objects in an environment, including robots;
- constraints on the objects' physical relations;
- controlled actions that satisfy these constraints;
- computational constraints in the form of scheduling and coordination.

Each of the above needs to be specified during the implementation of the task and it is clear that the topological and geometric relations are central to this specification. Each constraint, controller, and computational algorithm that is based on such geometric data should be both unique and distinguishable, in order to be handled correctly. It is shown that enabling these properties on the basis of purely numerical data, which is associated with the specific (coordinate) representation, is error-prone, hence requires semantic meta data be added in a systematic way.

Section 1.3.1 introduces the MDE-based approach that identifies and separates semantic (physical) models from, first of all, their geometric and coordinate specific representations, and subsequently from framework specific implementations (Figure 1.7). Based on this approach, Chapter 2 develops semantic models for the kinematic chain structural primitives, and for the operations that work on these primitives. Building robot kinematic chain models using first their semantics and only then instantiating them to a concrete coordinate representation, allows to uniquely identify all features and all computations on the kinematic chain. This structured approach gives an opportunity to validate formally, whether the constructed kinematic chains and associated algorithm are semantically correct.

The thesis develops a series of DSLs that supports developers with the specification and the implementation of the composable kinematic chains and computational algorithm models that conform to the semantic models. The implementations of the DSLs make use of the existing geometric relations library by De Laet et al. [De Laet et al., 2013a] as an essential building block, and extends it along the same systematic ways from the geometry of frames to the geometry and dynamics of kinematic chains. The design of the DSL is driven by separating the structural data (e.g., a link, a joint, a chain etc.), from the operations/computations on those data (e.g., pose or twist transformations). This decomposition approach enables a use of the functional programming concepts such as a functor (a function object with no or little internal state), a high-order function (takes other functions as arguments) and a functional composition in the implementation of the DSLs. It also facilitates a programmatic expression of the complex robot kinematic and dynamic motion models as tree data structures with the operations updating the properties of the nodes and links.

The importance of this research is not so much in the (limited number of) new functionalities that it provides, but in the methodology of how it structures the complex domain of software for robot motion and task specification and control using the DSLs. Currently, the available DSLs are limited to the specification of robot kinematic chain structures and their operations, but the methodology can be applied with the same formal rigour to the immediate

neighbouring functionalities of controllers and estimators. In particular, Chapter 4 discusses how the domain specific constrained dynamics solver interacts with the existing control/stabilization approaches, hence providing a starting point to identify coordinate-free semantic models that can be used in the design and implementation of a DSL for the specification of the controllers.

The main challenge in the process of the DSL development is the determination of the domain specific semantic constraints on the domain primitives and operations. This process requires a lot of time and expert knowledge of the domain to get right. Furthermore, the DSLs require careful implementation of the tool-chains that can compile provided specifications into a semantically valid application code. The development of such tool-chains should take place in tandem with a domain expert. In this thesis, only a tiny portion of such tools is developed, since this requires tremendous efforts to be done professionally.

Chapter 3 introduces the functionalities and new extensions of the domain-specific constrained hybrid dynamics solver. The first example using the original solver was presented by the Russian roboticists Popov et al. in [Popov et al., 1978]. The functionality of the original solver is extended to cope with priority and weighting-based control approaches. These are achieved by following the design guidelines presented in Chapters 1 and 2. These new extensions of the solver come at a low cost by reconfiguration of the computational sweep inputs, and enable one to implement various configurations of WBCA that can cope with Cartesian and joint (posture) space constraints. The advantage of using these extensions of the solver over other similar approaches is that the presented solver does not require explicit nullspace projection of the constraint Jacobian to implement prioritization. The implementation and its various WBCA configurations were extensively tested in simulation and a simple real world application. Again, the foci and contributions of the presented research were in the systematic and formal modelling as well as the structured and configurable software implementations, and not at all on the quality or performance of the controllers or estimators that are required in all real-world robotics applications.

Chapters 3 and 4 also position the constrained hybrid dynamics solver in retrospect to domain-independent optimization-based numerical solvers. It is observed that the optimization-based numerical solvers treat constraints, robot motion model and cost function in batch and compute the optimal control inputs to the system. This is unlike domain-specific solvers which require each aspect be treated independently, thus the need for a separate controller design in WBCA setups that are presented. Another difference with the domain-specific solvers is that they allow the incorporation of the semantic models which can be used to validate whether the motion task setup is correct.

Finally, during the course of the software development in the context of this research, the author came to realize one seemingly intuitive but important point that often shifts to the background in the development process. It is the fact that the programming of the robot systems should always rely on the domain-specific semantics and not only on software abstractions that frequently tend to ignore the physical nature of the systems.

Variability in whole-body control architecture:
Chapter 5 shows that WBCA is the underlying control architecture of many motion task control applications. It argues that even though existing motion task control frameworks originally had different focus points, they all realize some specific configurations of WBCA. The issue with these frameworks is that they not only use specific representations for each WBCA component, but also it is not evident how these components interact with each other and are related to their software implementations. It frequently leads to ambiguity about how and for what kind of applications they can be utilized; these ambiguities quickly come to the surface when application builders have to integrate software components developed by independent groups. As can easily be observed in the state of the practice, such multi-group software systems are still extremely rare, which is major barrier to progress.

In order to address this problem, this research decomposes WBCA into its constituting components and analyzes their variations. The explicit separation of WBCA into constraint spaces, constraints, controllers and solvers allow to systematically structure any task specification problem as a mere reconfiguration of the same architectural model (Figure 5.2). This architectural organization emphasizes the fact that as long as the right type of constraint solver is chosen, any constrained task can be implemented. On the software side, since the responsibilities and inter-relations of the WBCA components are clear, it is simpler to organize the whole application as the composition of DSLs with clear roles and constraints. Figure 5.1 illustrates multiple DSLs in the motion stack that are used to implement WBCA for several example applications. Furthermore, the identification of the component variations in WBCA and associated motion stack enable an integration of the non-conventional receding horizon prediction algorithm for the controller gains as a component in its own.

Future work

Even though the results of this research are positive and are already available for use on real robots, there is still a lot that needs to be realized in order to improve constrained motion modelling and programming. Some of the immediate topics

that need to be addressed, but are not delivered in this thesis, are summarized below.

- With respect to semantics and DSL for composable kinematic chain modelling:
  - The presented definitions of the semantic models do not yet cover the primitives that are required to model kinematic chains and computations with dynamic properties such as impedance properties of joint transmissions and a body that a segment is the part of, explicit specification of the robot Jacobian, and unit systems that describe physical relations.
  - The presented kinematics and dynamics DSL implementation uses C++ templates. In the next iteration, a dependence on such a programming language specific feature should be avoided. However, the introduction of a new representation, which is independent of a general programming language, i.e., not internal DSL, would also imply that the necessary compiler tool-chain needs to be developed from the ground up. So, both options need to be weighed.
  - The current use of geometric relations semantics is limited to the kinematic chain models. They are used neither for the specifications of the geometric constraints such as a path, an orthogonality, a direction of motion etc., nor the specification of the constraint controllers. Extending the use of the semantics to these components will further improve the determinism and the validation features of the task specification.
- With respect to the solver, WBCA and motion stack:
  - The current implementation of the constrained hybrid dynamics solver uses only 1-DoF joint model and can only takes into account Cartesian constraints at the end-effector segments. There is already a support to perform forward kinematic computations on tree structured kinematic chains using the semantic models. But full dynamics computations for the tree structures with multi-DoF are to be implemented. The future implementation should also provide support for the constraints that can be placed anywhere on the kinematic chain. This will help one to cope with the mobile and other platform types without the need for the additional kinematic chain manipulations as in the example presented in Section 5.3. These extensions require only more software development work, since the theoretical aspects of tree- and graph-structured chains, as well as multi-segment constraints, were developed several decades ago.

- The presented WBCA setups only use a constrained hybrid dynamics solver. This research does not consider how WBCA and the hybrid dynamics solver can be used in cooperation with more general purpose optimization solvers and toolkits such as ACADO [Houska et al., 2009], CasADi [Andersson, 2013] and qpOASES [Ferreau et al., 2014]. The research by Aertbeliën and De Schutter [Aertbeliën and Schutter, 2014] shows how such an integration can be realised.
- All of the presented WBCA software implementations use multiple DSLs with one DSL per task component. But this is still cumbersome to use, because one often needs to switch between several syntaxes and environments and know how these various compile and run-time environments interact with each other. Therefore, a compiler tool-chain that can support multiple semantic models, e.g., the specification of the motion task components, the description of software components, their architectural deployments, in a single "Integrated Development Environment" would be a very welcome (or rather, an indispensable) development. Several language workbenches, e.g., JetBrains MPS [JetBRAINS, 2015] and compiler-compiler frameworks, e.g., JastAdd [Computer Science, Lund University, 2015], promise to have such features and that can be used to implement such a motion programming infrastructure.

Appendix A

JSON representation for the semantic models of kinematic chains

```
"@geometry": {

  "@dsltype":"relationship",
  "name": " pos1",
  "id": "pos1-id",

  "@geomtype":"Position",

  "semantics":{
    "target":{
      "point": {
        "@semantic_primitive": {
          "@dsltype":"primitive",
          "name": "tp1",
          "id": "tp1-id",
          "@semanticstype": "Point"
        }
      },
      "body": {
        "@semantic_primitive": {
          "@dsltype":"primitive",
          "name": "tb1",
          "id": "tb1-id",
          "@semanticstype": "Body"
        }
      }
    },
```

```
28
29      "reference":{
30        "point": {
31          "@semantic_primitive": {
32            "@dsltype":"primitive",
33            "name": "rp1",
34            "id": "rp1-id",
35            "@semanticstype": "Point"
36          }
37        },
38        "body": {
39          "@semantic_primitive": {
40            "@dsltype":"primitive",
41            "name": "rb1",
42            "id": "rb1-id",
43            "@semanticstype": "Body"
44          }
45        }
46      },
47
48      "coordinateFrame":{
49        "@semantic_primitive": {
50          "@dsltype":"primitive",
51          "name": "rcf1",
52          "id": "rcf1-id",
53          "@semanticstype": "Frame",
54          "axes":[
55          "X",
56          "Y",
57          "Z"
58          ]
59        }
60      },
61
62      "physicalquantity": "Length"
63    },
64
65    "coordinates": {
66      "x": 0.0,
67      "y": 0.0,
68      "z": 0.0,
69      "physicalunit":"meter"
70    }
71  }
```

Listing A.1: Pose semantics in JSON representation as defined in terms of a point, a frame and a body modeling primitives.

```
{
  "@context":
  {
    "TaskDSL":"http://some-url/task-dsl.jsonld",
    "KinematicChainDSL":"http://some-url/kinematic-chain-dsl.
        jsonld",
    "GeometricRelationsDSL":"http://some-url/geometric-relations-
        dsl.jsonld"
  },
  "@graph":
  [
    {
      "@id":"_b0",
      "@type": "TaskDSL::Task",
      "name": "Task_one",
      "has_a":{"@id":"_b1"}
    },

    {
      "@id":"_b1",
      "@type": "TaskDSL::Robot",
      "name": "youBot",
      "is_a": {"@id":"_b2"}
    },

    {
      "@id":"_b2",
      "@type": "KinematicChainDSL::KinematicChain",
      "name": "youBotStructuralModel",
      "hasSegments": {[{"@id":"_s0"}, {"@id":"_s1"}]},
      "hasJoints": {[{"@id":"_j2"}]}
    },

    {
      "@id":"_s0",
      "@type": "KinematicChainDSL::Segment",
      "name": "Segment0",
      "hasLink": {"@id":"_l0"},
      "hasFrames": {[{"@id":"_f0"}]}
    },

    {
      "@id":"_l0",
      "@type": "KinematicChainDSL::Link",
      "name": "Link0",
      "composedOf": {[{"@id":"_tipFrame"}, {"@id":"_rootFrame"
          }]},
      "hasBody":{"@id":"_body0"}
    },

    {
      "@id":"_tipFrame",
      "@type": "GeometricRelationsDSL::Pose",
```

```
51        "name": "tip0Pose",
52        "hasSemantics": {[{"@id":"_tippoint0"}, {"@id":"_tipframe0"
              }, {"@id":"_tipbody0"},{"@id":"_refpoint0"}, {"@id":"
              _refframe0"}, {"@id":"_refbody0"}]},
53        "hasCoordSemantics": {[{"@id":"_tippoint0"}, {"@id":"
              _tipframe0"}, {"@id":"_tipbody0"},{"@id":"_refpoint0"},
              {"@id":"_refframe0"}, {"@id":"_refbody0"}, {"@id":"
              _coordframeR"}]},
54        "hasCoordinates":{"@id":"_coord1"}
55      },
56
57    ]
58 }
```

Listing A.2: An excerpt of an JSON-LD representation of the graph which models a structure of a robot.

Appendix B

Receding horizon predictive gain adaptation algorithm *

This appendix provides additional content on the gain adaptation algorithm which is first discussed in Section 5.4.1 in the context of WBCA with a constrained hybrid dynamics solver. It presents a number of experiments that compare tracking behaviors of the controllers that use gain adaptation. It also analyzes what different parameters of the receding horizon adaptation algorithm may influence the performance of the controllers.

Following a circular Cartesian trajectory: In this experiment, the end-effector of a 5-DoF KUKA youBot is to track a circular trajectory in the YZ plane. Here, the X degree of freedom is constrained as follows:

$$\boldsymbol{A}_N = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \end{pmatrix}, \quad \boldsymbol{b}_N = \begin{pmatrix} 0 \end{pmatrix}. \tag{B.1}$$

For a comparison, the simulations were run in a conventional control loop (Figure 4.3) with constant controller parameters and in a control loop with the receding horizon adaptation algorithm (Figure 5.6). The plot in Figure B.1(a)

*The content of this appendix is based on the research conducted under the supervision of the author and presented in [Copejans, 2014]

shows the overall performance of the tracking when the prediction is either on or off with respect to the desired trajectory. Figures B.1(b)–(d) show the error trajectory in each degree of freedom. While the tracking performance in some degrees of freedom does not improve considerably when the prediction is on, the overall tracking behavior improves compared to the case without the algorithm. The advantage of the adaptation is better visualized when in addition to the Cartesian space constraints, there are also joint space constraints. This is shown in the following setup.

Following an infinity sign Cartesian trajectory: As in the previous case, only X DoF is constrained in the Cartesian space. But now the 5-DoF serial chain is to track an infinity sign in YZ plane and the mechanical limits of two of its joints, the second and the third, are taken into account during the motion. Figure B.2(a) shows that the overall performance is considerably better when the prediction is on. This is also reflected in position error dynamics in each DoF as in Figures B.2(b)–(d), which show less fluctuations. An introduction of the prediction of the gains also incorporates computation of the cost functions that penalize the controllers' behaviors whenever some task constraints are not satisfied. In this motion task, joint limits were taken into account too. As can be seen in Figures B.2(e)–(f), these constraints are also satisfied when the prediction algorithm is on. Figures B.3 show the dynamics of the controller gains that are being adapted during the prediction process. Figures B.4 show the influence of the cost function on the tracking performance and the dynamics of the controller gains.

Reaching for an object pose: In this setup the 5-DoF serial robot is to make a return trip between two points. Here, the motion does not take place along the specified trajectory. It is a set point control motion. Since at the beginning of the motion the relative error, which is defined by the distance between the points, is large, the motions from the origin to the destination and back look abrupt. After the robot reaches the destined point, it pauses for some time and then returns to the original point. This behavior can be observed in Figures B.5(a)–(c). The relative error serves as an input to the impedance controller (Equation (4.9)). Such abrupt motions put a lot of strain on the robot joints, in particular on the joints that are between the base and the end-effector joints, because of the high accelerations that are generated. This can be seen in Figure B.5(d), where the torque values at these joints are high. Over time, it may wear the mechanical components out and eventually cause the failure of the drive block. This issue can be elevated by penalizing high torque and acceleration values and adapting the control gains. We have applied our receding horizon predictive gain adaptation algorithm in combination with such

penalizing cost functions. The results of the setup depicted in Figures B.5(e)–(f) show that the peak values of the torques are reduced considerably.

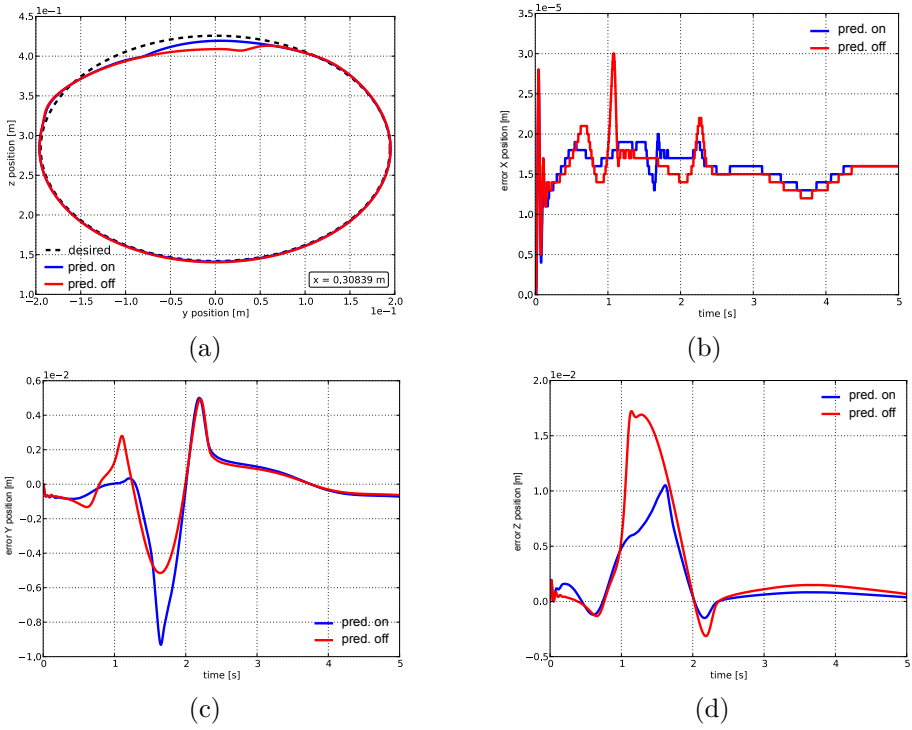


Figure B.1: The Figures show the performance comparison between a conventional and a predictive gain adaptation algorithm based control schema for the tracking of a circular trajectory in the YZ plane with the X degree of freedom being acceleration-constrained. The figures (b)–(d) show the error dynamics for each degree of freedom.

Algorithm B.1: Dynamically compliant motion tracking control with a receding horizon gain adaptation

Input : Robot geometric, inertial data, initial configuration$(q, \dot{q}, \ddot{X}_0, f_{ext})$, constraints

Output : Constraint accelerations and forces

```
1  begin
2      while t < T_task do
3          Save state X, Ẋ, Ẍ, τ, F_e, b
4          Construct control gain sets K_n to be tried
           // compute possible trajectories using the gains from K_n
5          for K_j in K_n do
6              while t_p ≤ T do
7                  Compute desired trajectories for joint and Cartesian space
8                  Compute control laws with K_j
9                  Solve constrained dynamics (Alg. 3.3)
10                 Compute and compare cost functions
11                 Integrate to obtain state for the next prediction iteration
12                 if (t_p = T) then
13                     Set the optimal gains K_optimal = K_j
14                 t_p = t_p + Δt_pred−integ
           // use the optimal gains to compute the real trajectory
15         for t_non−pred ← 0 to N do
16             Compute desired trajectories for joint and Cartesian space
17             Compute control laws using K_optimal
18             Solve constrained dynamics (Alg. 3.3)
19             Send the outcome to the robot
               // or for a simulation loop
20             Integrate to obtain state for the next iteration
21             t_non−pred = t_non−pred + Δt_integ
22         t = t + N + T
```

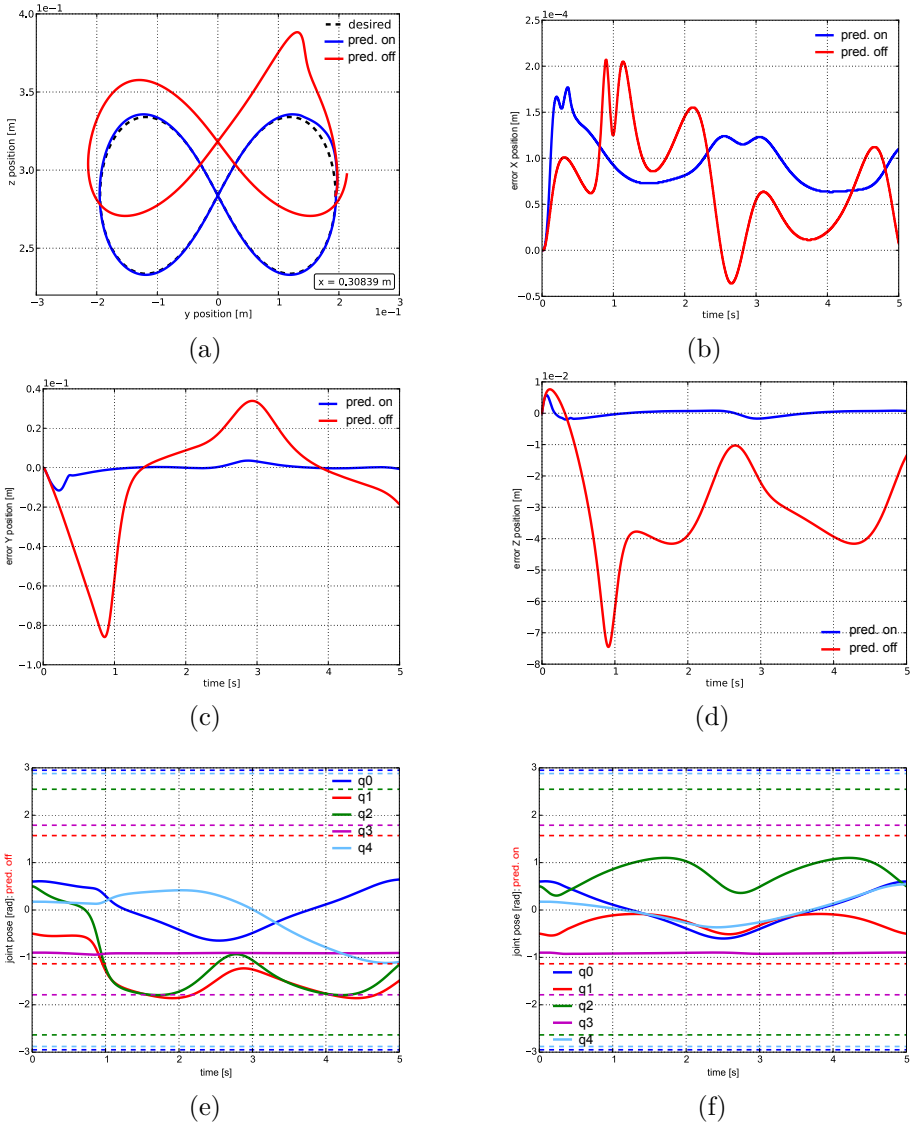


Figure B.2: The figures show a performance comparison between a conventional and a predictive gain adaptation algorithm based control schema for the tracking of an infinity sign shaped trajectory in the YZ plane with the X degree of freedom being acceleration-constrained. Figures(b)–(d) compare the position error dynamics of each control approach for each degree of freedom. Figures(e)–(f) show the joint positions. Here the dashed lines represent the joint limits. Figure(f) shows that the motions of the second and the third joints do not go over their limits, whenever the prediction algorithm is on.

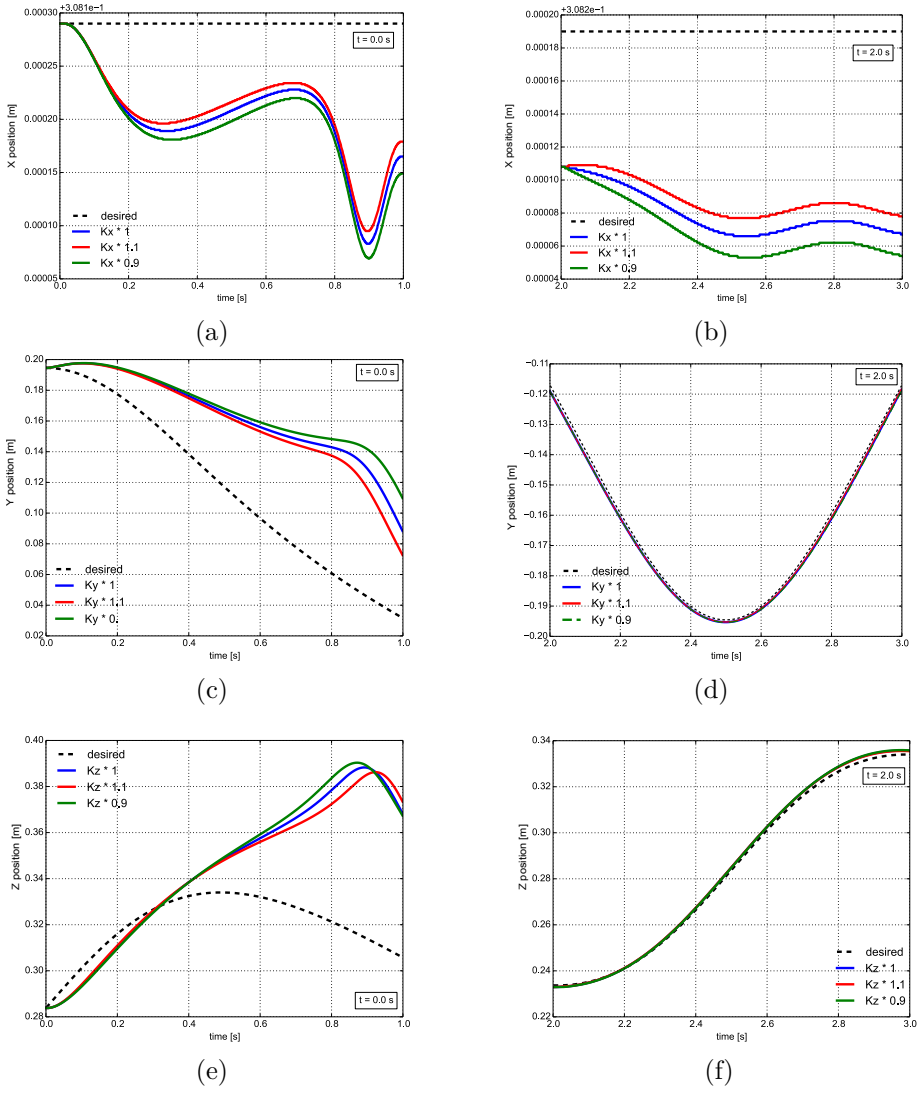


Figure B.3: shows how a different set of controller gains perform during the prediction process. Here, the prediction period is $\boldsymbol{T} = 1000$ *ticks*, an integration step during the prediction is $\boldsymbol{\Delta} t_{pred-integ} = 1$ *tick*, the prediction takes place every $10\boldsymbol{\Delta} t$ *ticks* and the controller gain set $\boldsymbol{K}$ is of size 27. For the latter the gain values are set manually to differ by 10% from their initial nominals, thus $\{0.9\mathrm{K}_{(*)}\ , \mathrm{K}_{(*)}\ , 1.1\mathrm{K}_{(*)}\}$. (a)–(b) show how the acceleration gains $\boldsymbol{K_a}$ of the beta control law (Equation (4.7)) in X DoF at the first and the second predictions fare. (c)–(f) show the results for the Y and Z DoF stiffness gains $\boldsymbol{K_c}$ of the impedance control law (Equation (4.9)).

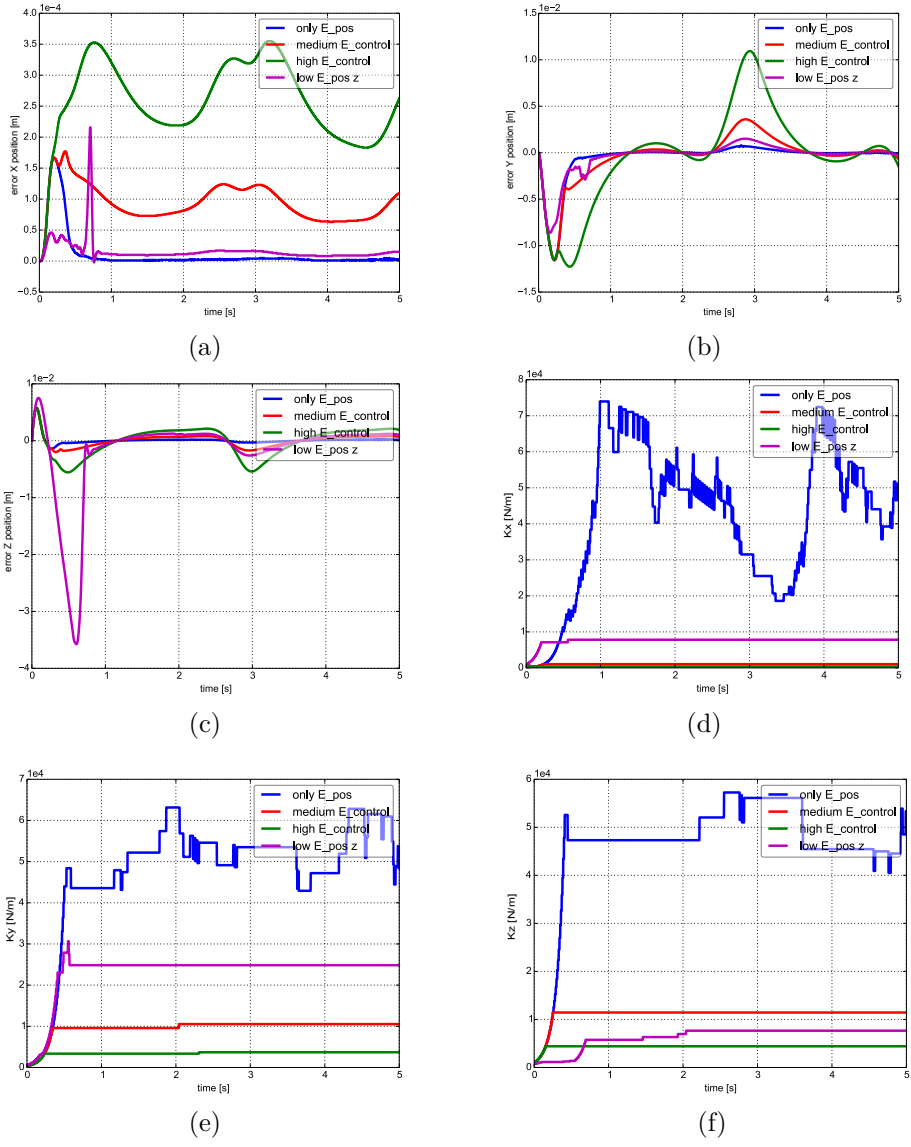


Figure B.4: shows how the form of the cost function affects the performance of the tracking. (a)–(c) show the error dynamics in X, Y and Z DoF. (d)–(f) show how the gains $\boldsymbol{K_{ax}}$, $\boldsymbol{K_{cy}}$ and $\boldsymbol{K_{cz}}$ vary with the change of the cost function.

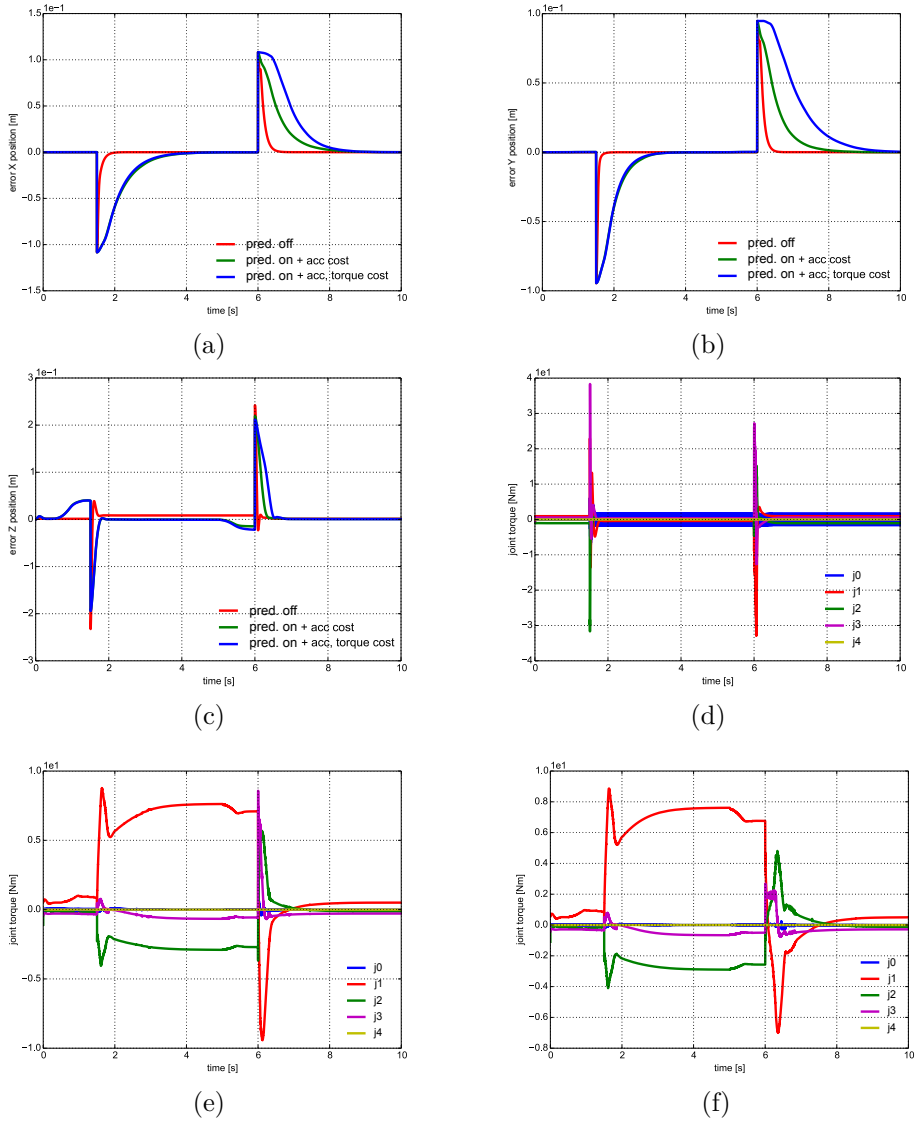


Figure B.5: shows the performance of the set-point controller. The robot's end-effector is to make a return trip between two points using an impedance and beta control law, while keeping the torques at the joints as low as possible in order not to damage them. (a)–(c) show the position error dynamics along each linear DoF. (d) shows the torque profiles of each joint when the predictive gain adaptation is off. (e)–(f) show the torque profiles when the predictive adaptation is on and both joint torques and the end-effector acceleration are penalized.

Bibliography

[Abrahams and Gurtovoy, 2004] Abrahams, D. and Gurtovoy, A. (2004). *C++ template metaprogramming: concepts, tools, and techniques from Boost and beyond*. Pearson Education.

[Adida et al., 2012] Adida, B., Herman, I., Sporny, M., and Birbeck, M. (2012). RDFa 1.1 primer. http://www.w3.org/TR/rdfa-primer/.

[Aertbeliën and Schutter, 2014] Aertbeliën, E. and Schutter, J. D. (2014). eTaSL/eTC: A constraint-based task specification language and robot controller using expression graphs. In *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems, Chicago, IL, USA, September 14-18, 2014*, pages 1540–1546. IEEE.

[Albu-Schäffer et al., 2007a] Albu-Schäffer, A., Haddadin, S., Ott, C., Stemmer, A., Wimböck, T., and Hirzinger, G. (2007a). The DLR lightweight robot: design and control concepts for robots in human environments. *Industrial Robot: An International Journal*, 34(5):376–385.

[Albu-Schäffer et al., 2003] Albu-Schäffer, A., Ott, C., Frese, U., and Hirzinger, G. (2003). Cartesian impedance control of redundant robots. Recent results with the DLR Light-Weight-Arms. In *Proceedings of the 2003 IEEE International Conference on Robotics and Automation*, pages 3704–3709, Taipeh, Taiwan.

[Albu-Schäffer et al., 2007b] Albu-Schäffer, A., Ott, C., and Hirzinger, G. (2007b). A unified passivity-based control framework for position, torque and impedance control of flexible joint robots. *The International Journal of Robotics Research*, 26(1):23–39.

[Alexandrescu, 2001] Alexandrescu, A. (2001). *Modern C++ design: generic programming and design patterns applied*. Addison-Wesley.

[An et al., 1987] An, C. H., Atkeson, C. G., Griffiths, J. D., and Hollerbach, J. M. (1987). Experimental evaluation of feedforward and computed torque control. In *Proceedings of the 1987 IEEE International Conference on Robotics and Automation*, pages 165–168, Raleigh, NC.

[Andersson, 2013] Andersson, J. (2013). *A general-purpose software framework for dynamic optimization*. PhD thesis, Faculty of Engineering, KU Leuven, Leuven, Belgium.

[Appell, 1899] Appell, P. (1899). Sur une forme générale des équations de la Dynamique. *Comptes rendus hebdomadaires des séances de l'Académie des sciences*, 129:423–427, 459–460.

[Aström and Murray, 2010] Aström, K. J. and Murray, R. M. (2010). *Feedback systems: an introduction for scientists and engineers*. Princeton university press.

[Atkinson and Kühne, 2003] Atkinson, C. and Kühne, T. (2003). Model-driven development: a metamodeling foundation. *IEEE software*, 20(5):36–41.

[AutomationML, 2008] AutomationML (2008). Automation Markup Language. http://www.autosar.org/.

[Barnes and Finch, 2008] Barnes, M. and Finch, E. L. (2008). COLLADA—Digital Asset Schema Release 1.5.0. http://www.collada.org. Last visited August 2013.

[Bass et al., 2003] Bass, L., Clements, P., and Kazman, R. (2003). *Software Architecture in Practice*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2 edition.

[Bauchau and Laulusa, 2008] Bauchau, O. A. and Laulusa, A. (2008). Review of contemporary approaches for constraint enforcement in multibody systems. *Transactions of the ASME, Journal of Computational and Nonlinear Dynamics*, 3(1):011005–1–8.

[Baumgarte, 1972] Baumgarte, J. W. (1972). Stabilization of constraints and integrals of motion in dynamical systems. *Computer Methods in Applied Mechanics and Engineering*, 1(1):1–16.

[Beckett et al., 2008] Beckett, D., Berners-Lee, T., and Prud'hommeaux, E. (2008). Turtle-terse rdf triple language. *W3C Team Submission*, 14.

[Bejczy, 1974] Bejczy, A. K. (1974). Robot arm dynamics and control. Technical report, NASA, Jet Propulsion Laboratory.

[Bellman, 1957] Bellman, R. (1957). *Dynamic Programming*. Princeton University Press, Princeton, New Jersey.

[Bentley, 1986] Bentley, J. (1986). Programming pearls: little languages. *ACM Communications Magazine*, 29(8):711–721.

[Berger et al., 2013] Berger, T., She, S., Lotufo, R., Wasowski, A., and Czarnecki, K. (2013). A study of variability models and languages in the systems software domain. *Software Engineering, IEEE Transactions on*, 39(12):1611–1640.

[Bertsekas, 1995] Bertsekas, D. P. (1995). *Dynamic Programming and Optimal Control, Volume I*. Athena Scientific, Belmont Massachusetts.

[Betts, 2010] Betts, J. T. (2010). *Practical methods for optimal control and estimation using nonlinear programming*, volume 19. Siam.

[Bézivin, 2005] Bézivin, J. (2005). On the unification power of models. *Software and Systems Modeling*, 4(2):171–188.

[Biggs and Makarenko, 2010] Biggs, G. and Makarenko, A. (2010). The GearBox project. http://gearbox.sourceforge.net/.

[Bischoff et al., 2010] Bischoff, R., Guhl, T., Prassler, E., Nowak, W., Kraetzschmar, G., Bruyninckx, H., Soetens, P., Hägele, M., Pott, A., Breedveld, P., Broenink, J., Brugali, D., and Tomatis, N. (2010). BRICS—Best practice in robotics. In *41st International Symposium on Robotics*, pages 968–975, Munich, Germany.

[Bizer et al., 2009] Bizer, C., Heath, T., and Berners-Lee, T. (2009). Linked data-the story so far. *International Journal on Semantic Web and Information Systems*, 5(3):22.

[Bock and Plitt, 1984] Bock, H. G. and Plitt, K.-J. (1984). A multiple shooting algorithm for direct solution of optimal control problems. In *Proceeding of the International Federation of Automatic Control*.

[Bruyninckx and De Schutter, 1996] Bruyninckx, H. and De Schutter, J. (1996). Specification of force-controlled actions in the "Task Frame Formalism": A survey. *IEEE Transactions on Robotics and Automation*, 12(5):581–589.

[Bruyninckx and Khatib, 2000] Bruyninckx, H. and Khatib, O. (2000). Gauss' Principle and the dynamics of redundant and constrained manipulators. In *Proceedings of the 2000 IEEE International Conference on Robotics and Automation*, pages 2563–2568, San Francisco, CA.

[Bruyninckx et al., 2013] Bruyninckx, H., Klotzbücher, M., Hochgeschwender, N., Kraetzschmar, G., Gherardi, L., and Brugali, D. (2013). The BRICS Component Model: A model-based development paradigm for complex robotics software systems. In *28th ACM Symposium On Applied Computing*, pages 1758–1764.

[Cataldo, 2006] Cataldo, A. (2006). *The Power of Higher-Order Composition Languages in System Design.* PhD thesis, University of California, Berkeley.

[Chakrabarti et al., 2003] Chakrabarti, A., de Alfaro, L., and Henzinger, T. A. (2003). Resource Interfaces. In *Proceedings of the Third International Conference on Embedded Software (EMSOFT)*, pages 117–133. Springer-Verlag.

[Chaumette, 1998] Chaumette, F. (1998). Potential problems of stability and convergence in image-based and position-based visual servoing. In Kriegman, D., Hager, G. ., and Morse, A., editors, *The Confluence of Vision and Control*, pages 66–78. LNCIS Series, No 237, Springer-Verlag.

[Chaumette and Hutchinson, 2006] Chaumette, F. and Hutchinson, S. (2006). Visual servo control Part I: Basic approaches. *IEEE Robotics and Automation Magazine*, 13(4):82–90.

[Chaumette and Hutchinson, 2007] Chaumette, F. and Hutchinson, S. (2007). Visual servo control Part II: Advanced approaches. *IEEE Robotics and Automation Magazine*, 14(1):109–118.

[Chen et al., 2009] Chen, L., Ali Babar, M., and Ali, N. (2009). Variability management in software product lines: a systematic review. In *Proceedings of the 13th International Software Product Line Conference*, pages 81–90. Carnegie Mellon University.

[Chiou et al., 1999] Chiou, J. C., Yang, J. Y., and Wu, S. D. (1999). Stability analysis of Baumgarte constraint stabilization technique in multibody dynamic systems. *Journal of Guidance, Control, and Dynamics*, 22(1):160–162.

[Chitta et al., 2012] Chitta, S., Hsiao, K., Jones, G., Sucan, I., and Hsu, J. (2012). Semantic Robot Description Format (SRDF). http://www.ros.org/wiki/srdf.

[Clements et al., 2002] Clements, P., Kazman, R., and Klein, M. (2002). *Evaluating Software Architectures: Methods and Case Studies.* Addison-Wesley.

[Compton et al., 2013] Compton, M., Barnaghi, P., Bermudez, L., Garía-Castro, R., Corcho, O., Cox, Simon Graybeal, J., Hauswirth, M., Henson, C., Herzog, A., Huang, V., Janowicz, K., Kelsey, W. D., Phuoc, D. L., Lefort, L., Leggieri, M., Neuhaus, H., Nikolov, A., Page, K., Passant, A., Sheth, A., and Taylor, K. (2013). The SSN Ontology of the W3C semantic sensor network incubator group. *Journal of Web Semantics*.

[Computer Science, Lund University, 2015] Computer Science, Lund University (2015). JastAdd. http://jastadd.org/web/documentation/concept-overview.php.

[Consortium, 2008] Consortium, O. S. M. (2008). OpenModelica. http://www.openmodelica.org. Last visited August 2012.

[Copejans, 2014] Copejans, M. (2014). Adaptive control with a new constrained dynamics algorithm on the KUKA youBot robot. Master's thesis, University of Leuven, Belgium.

[Crockford, 2006] Crockford, D. (2006). The application/json Media Type for JavaScript Object Notation (JSON). http://tools.ietf.org/html/rfc4627.

[Czarnecki and Eisenecker, 2000] Czarnecki, K. and Eisenecker, U. W. (2000). *Generative Programming: Methods, Tools, and Applications*. ACM Press/Addison-Wesley Publishing Co., New York, NY, USA.

[Dantam et al., 2010] Dantam, N., Kolhe, P., and Stilman, M. (2010). The motion grammar for physical human-robot games. In *Proceedings of the IEEE International Conference on Robotics and Automation*, Anchorage, Alaska, USA.

[Dantam and Stilman, 2012] Dantam, N. and Stilman, M. (2012). The motion grammar: Linguistic perception, planning, and control. *Robotics: Science and Systems VII*, page 49.

[DARPA, 2004a] DARPA (2004a). DARPA grand challenge. http://archive.darpa.mil/grandchallenge04/. Last visited October 2014.

[DARPA, 2004b] DARPA (2004b). DARPA urban challenge. http://archive.darpa.mil/grandchallenge/. Last visited October 2014.

[DARPA, 2005] DARPA (2005). DARPA grand challenge. http://archive.darpa.mil/grandchallenge05/. Last visited October 2014.

[DARPA, 2013] DARPA (2013). DARPA robotics challenge. http://www.theroboticschallenge.org. Last visited October 2014.

[de Alfaro and Henzinger, 2001] de Alfaro, L. and Henzinger, T. A. (2001). Interface Theories for Component-Based Design. In *EMSOFT '01: Proceedings of the First International Workshop on Embedded Software*, pages 148–165. Springer-Verlag.

[De Laet and Bellens, 2012] De Laet, T. and Bellens, S. (2012). Geometric semantics software. http://www.orocos.org/wiki/geometric-relations-semantics-wiki. Last visited September 2012.

[De Laet et al., 2013a] De Laet, T., Bellens, S., Bruyninckx, H., and De Schutter, J. (2013a). Geometric relations between rigid bodies (part 2): from semantics to software. *IEEE Robotics and Automation Magazine*, 20(2):91–102.

[De Laet et al., 2013b] De Laet, T., Bellens, S., Smits, R., Aertbeliën, E., Bruyninckx, H., and De Schutter, J. (2013b). Geometric relations between rigid bodies (Part 1): Semantics for standardization. *IEEE Robotics and Automation Magazine*, 20(1):84–93.

[De Laet et al., 2013c] De Laet, T., Bruyninckx, H., and De Schutter, J. (2013c). Rigid body pose and twist scene graph founded on geometric relations semantics for robotic applications. In *Proceedings of the 2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Tokyo, Japan.

[De Laet et al., 2012] De Laet, T., Smits, R., Bruyninckx, H., and De Schutter, J. (2012). Constraint-based task specification and control for visual servoing application scenarios. *Automatisierungstechnik*, 5:260–269.

[De Schutter et al., 2007] De Schutter, J., De Laet, T., Rutgeerts, J., Decré, W., Smits, R., Aertbeliën, E., Claes, K., and Bruyninckx, H. (2007). Constraint-based task specification and estimation for sensor-based robot systems in the presence of geometric uncertainty. *The International Journal of Robotics Research*, 26(5):433–455.

[Debrouwere et al., 2013] Debrouwere, F., Van Loock, W., Pipeleers, G., Tran Dinh, Q., Diehl, M., De Schutter, J., and Swevers, J. (2013). Time-optimal path following for robots with trajectory jerk constraints using sequential convex programming. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 1908–1913, Karlsruhe, Germany.

[Decré et al., 2010] Decré, W., Bruyninckx, H., and De Schutter, J. (2010). An optimization-based estimation and adaptive control approach for human-robot cooperation. In *12th International Symposium on Experimental Robotics*, Delhi, India.

[Denavit and Hartenberg, 1955] Denavit, J. and Hartenberg, R. S. (1955). A kinematic notation for lower-pair mechanisms based on matrices. *Transactions of the ASME, Journal of Applied Mechanics*, 23:215–221.

[Diehl, 2001] Diehl, M. (2001). *Real-time optimization for large scale nonlinear processes*. PhD thesis, University of Heidelberg.

[Diehl et al., 2006] Diehl, M., Bock, H., Diedam, H., and Wieber, P.-B. (2006). Fast direct multiple shooting algorithms for optimal robot control. In Diehl, M. and Mombaur, K., editors, *Fast Motions in Biomechanics and Robotics*, volume 340 of *Lecture Notes in Control and Information Sciences*, pages 65–93. Springer Berlin.

[Doty et al., 1993] Doty, K. L., Melchiorri, C., and Bonivento, C. (1993). A theory of generalized inverses applied to robotics. *The International Journal of Robotics Research*, 12(1):1–19.

[Doyen et al., 2008] Doyen, L., Henzinger, T. A., Jobstmann, B., and Petrov, T. (2008). Interface theories with component reuse. In *EMSOFT '08: Proceedings of the 8th ACM international conference on Embedded software*, pages 79–88. ACM.

[Duffy, 1990] Duffy, J. (1990). The fallacy of modern hybrid control theory that is based on "orthogonal complements" of twist and wrench spaces. *Journal of Robotic Systems*, 7(2):139–144.

[Egerstedt and Hu, 2002] Egerstedt, M. and Hu, X. (2002). A hybrid control approach to action coordination for mobile robots. *Automatica*, 38(1):125–130.

[Eichelberger and Schmid, 2013] Eichelberger, H. and Schmid, K. (2013). A systematic analysis of textual variability modeling languages. In *Proceedings of the 17th International Software Product Line Conference*, pages 12–21. ACM.

[Eker and Janneck, 2003] Eker, J. and Janneck, J. W. (2003). CAL Language Report Specification of the CAL Actor Language. Technical Report UCB/ERL M03/48, EECS Department, University of California, Berkeley.

[Eker et al., 2003a] Eker, J., Janneck, J. W., Lee, E. A., Liu, J., Liu, X., Ludvig, J., Neuendorffer, S., Sachs, S., and Xiong, Y. (2003a). Taming Heterogeneity - The Ptolemy Approach. In *Proceedings of the IEEE*, pages 127–144.

[Eker et al., 2003b] Eker, J., Janneck, J. W., Lee, E. A., Liu, J., Liu, X., Ludvig, J., Neuendorffer, S., Sachs, S., and Xiong, Y. (2003b). Taming heterogeneity—The Ptolemy approach. *Proceedings of the IEEE*, 91(1):127–144.

[Erdweg et al., 2013] Erdweg, S., van der Storm, T., Völter, M., Boersma, M., Bosman, R., Cook, W. R., Gerritsen, A., Hulshout, A., Kelly, S., Loh, A., et al. (2013). The State of the Art in Language Workbenches: Conclusions from the Language Workbench Challenge. In *Software Language Engineering*, pages 197–217. Springer.

[Erez et al., 2013] Erez, T., Lowrey, K., Tassa, Y., Kumar, V., Kolev, S., and Todorov, E. (2013). An integrated system for real-time Model Predictive Control of humanoid robots. In *EEE/RAS International Conference on Humanoid Robots*.

[Erez et al., 2011] Erez, T., Tassa, Y., and Todorov, E. (2011). Infinite-horizon model predictive control for periodic tasks with contacts. In *Proceedings of Robotics: Science and Systems*, Los Angeles, USA.

[Erez and Todorov, 2012] Erez, T. and Todorov, E. (2012). Trajectory optimization for domains with contacts using inverse dynamics. In *Proceedings of the 2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Vilamoura, Portugal.

[Escande et al., 2010] Escande, A., Mansard, N., and Wieber, P.-B. (2010). Fast resolution of hierarchized inverse kinematics with inequality constraints. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 3733–3738, Anchorage, Alaska, USA.

[Espiau et al., 1992] Espiau, B., Chaumette, F., and Rives, P. (1992). A new approach to visual servoing in robotics. *IEEE Transactions on Robotics and Automation*, 8(3):313–326.

[Euler, 1776] Euler, L. (1776). *Leonhardi Euleri Opera omnia. Series 2: Opera mechanica et astronomica*. Fussli Zurich. exact date unknown.

[Featherstone, 1983] Featherstone, R. (1983). The calculation of robot dynamics using articulated-body inertias. *The International Journal of Robotics Research*, 2(1):13–30.

[Featherstone, 2001] Featherstone, R. (2001). The acceleration vector of a rigid body. *The International Journal of Robotics Research*, 20(11):841–846.

[Featherstone, 2006] Featherstone, R. (2006). Plücker basis vectors. In *Proceedings of the 2006 IEEE International Conference on Robotics and Automation*, pages 1892–1897, Orlando, U.S.A.

[Featherstone, 2008] Featherstone, R. (2008). *Rigid Body Dynamics Algorithms*. Springer.

[Featherstone, 2010a] Featherstone, R. (2010a). A beginner's guide to 6-d vectors (part 1). *Robotics Automation Magazine, IEEE*, 17(3):83–94.

[Featherstone, 2010b] Featherstone, R. (2010b). A beginner's guide to 6-d vectors (part 2). *Robotics Automation Magazine, IEEE*, 17(4):88–99.

[Felis, 2011] Felis, M. (2011). RBDL: Rigid Body Dynamics Library. http://rbdl.bitbucket.org/.

[Ferreau et al., 2014] Ferreau, H., Kirches, C., Potschka, A., Bock, H., and Diehl, M. (2014). qpOASES: A parametric active-set algorithm for quadratic programming. *Mathematical Programming Computation*, 6(4):327–363.

[Ferreau et al., 2008] Ferreau, H. J., Bock, H. G., and Diehl, M. (2008). An online active set strategy to overcome the limitations of explicit MPC. *International Journal of Robust and Nonlinear Control*, 18(8):816–830.

[Flores et al., 2011] Flores, Paulo Machado, M., Seabra, E., and Tavares da Silva, M. (2011). A parametric study on the Baumgarte stabilization method for forward dynamics of constrained multibody system. *Transactions of the ASME, Journal of Computational and Nonlinear Dynamics*, 6(1):011019–1–6.

[Foote et al., 2011a] Foote, T., Marder-Eppstein, E., and Meeussen, W. (2011a). tf. http://ros.org/wiki/tf. Last visited 2012.

[Foote et al., 2011b] Foote, T., Meeussen, W., and Marder-Eppstein, E. (2011b). tf2. http://ros.org/wiki/tf2. Last visited 2012.

[for Operations Research, 2005] for Operations Research, T. C. I. (2005). Interior Point Optimizer. https://projects.coin-or.org/Ipopt. Last visited 2014.

[Foundation, 2015] Foundation, E. (2015). Xtext. http://www.eclipse.org/Xtext/. Last visited July 2013.

[Fowler, 2005] Fowler, M. (2005). Language workbenches: The killer-app for domain specific languages? http://martinfowler.com/articles/languageWorkbench.html.

[Fowler, 2010] Fowler, M. (2010). *Domain Specific Languages*. Addison-Wesley Professional.

[Fragniere and Gondzio, 2002] Fragniere, E. and Gondzio, J. (2002). Optimization modeling languages. *Pardalos, P., Resende, M. Handbook of Applied Optimization*, pages 993–1007.

[Fraunhofer IPA, 2015] Fraunhofer IPA (2015). Care-o-bot. http://www.care-o-bot-4.de/.

[Frigerio et al., 2011] Frigerio, M., Buchli, J., and Caldwell, D. G. (2011). A Domain Specific Language for kinematic models and fast implementations of robot dynamics algorithms. In *Workshop on Domain Specific Languages for Robotics*.

[Frigerio et al., 2012] Frigerio, M., Buchli, J., and Caldwell, D. G. (2012). Code generation of algebraic quantities for robot controllers. In *Proceedings of the 2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2346–2351, Vilamoura, Portugal.

[Fuchs et al., 2009] Fuchs, M., Borst, C., Robuffo, P., Baumann, A., Kraemer, E., Langwald, J., Gruber, R., Seitz, N., Plank, G., Kunze, K., Burger, R., Schmidt, F., Wimboeck, T., and Hirzinger, G. (2009). Rollin' Justin — design considerations and realization of a mobile platform for a humanoid upper body. In *Proceedings of the 2009 IEEE International Conference on Robotics and Automation*, Kobe, Japan.

[Gantmacher, 1975] Gantmacher, F. (1975). *Lectures in Analytical Mechanics*. MIR, Moscow.

[Gauss, 1829] Gauss, K. F. (1829). Über ein neues allgemeines Grundgesetz der Mechanik. *Journal fü die reine und angewandte Mathematik*, 4:232–235.

[Gherardi, 2013] Gherardi, L. (2013). *Variability Modeling and Resolution in Component-based Robotics Systems*. PhD thesis, University of Bergamo.

[Ghosh, 2010] Ghosh, D. (2010). *DSL in Action*. Manning Publications.

[Gibbs, 1879] Gibbs, J. W. (1879). On the fundamental formulae of dynamics. *American Journal of Mathematics*, 2:49–64.

[Goessler and Sifakis, 2005] Goessler, G. and Sifakis, J. (2005). Composition for Component-Based Modeling. *Science of Computer Programming*, 55(1-3):161–183.

[Hägele, 2011] Hägele, M. (2011). Robot Standards. http://www.robot-standards.org.

[Hägele and Hans, 2005] Hägele, M. and Hans, M. (2005). SMErobot. http://www.smerobot.org.

[Heath and Bizer, 2011] Heath, T. and Bizer, C. (2011). Linked data: Evolving the web into a global data space. *Synthesis lectures on the semantic web: theory and technology*, 1(1):1–136.

[Hedin, 2011] Hedin, G. (2011). An introductory tutorial on jastadd attribute grammars. In *Generative and Transformational Techniques in Software Engineering III*, pages 166–200. Springer.

[Hertz, 1894] Hertz, H. (1894). *Die Prinzipien der Mechanik in neuem Zusammenhange dargestellt*. Barth, Leipzig, Germany. Reprinted in "The principles of mechanics," Dover, 1956.

[Hertz, 1984] Hertz, H. (1984). *Die Prinzipien der Mechanik. Einleiting*. Geest und Portig, Leipzig, Germany. With annotations by Josef Kuczera.

[Hewitt and Baker, 1978] Hewitt, C. and Baker, H. G. (1978). Actors and Continuous Functionals. Technical report, Massachusetts Institute of Technology.

[Hochgeschwender et al., 2013] Hochgeschwender, N., Gherardi, L., Shakhirmardanov, A., Kraetzschmar, G., Brugali, D., and Bruyninckx, H. (2013). A Model-based Approach to Software Deployment in Robotics. In *26th IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2013)*, Tokyo, Japan. IEEE/RJS.

[Hogan, 1985] Hogan, N. (1985). Impedance control: An approach to manipulation. Parts I-III. *Transactions of the ASME, Journal of Dynamic Systems, Measurement, and Control*, 107:1–24.

[Hollerbach and Suh, 1987] Hollerbach, J. M. and Suh, K. C. (1987). Redundancy resolution of manipulators through torque optimization. *IEEE Journal of Robotics and Automation*, 3(4):308–316.

[Hooker and Margulies, 1965] Hooker, W. W. and Margulies, G. (1965). The dynamical attitude equations for an arbitrary n-body satellite having r rotational degrees of freedom. *J. Astronautical Sciences*, 12(4):123–128.

[Houska et al., 2009] Houska, B., Ferreau, H. J., and Diehl, M. (2009). ACADO—An open-source toolkit for automatic control and dynamic optimization. In *Proceedings of the 2009 Belgian-French-German Conference on Optimization*, page 167, Leuven, Belgium.

[Hudak, 1996] Hudak, P. (1996). Building domain-specific embedded languages. *ACM Computing Surveys (CSUR)*, 28(4es):196.

[Humanoid Research Group, AIST, 2009] Humanoid Research Group, AIST (2009). HRP-4C humanoid robot. http://www.aist.go.jp/aist_e/latest_research/2009/20090513/20090513.html/.

[Hutchinson et al., 1996] Hutchinson, S., Hager, G. D., and Corke, P. I. (1996). A tutorial on visual servo control. *Robotics and Automation, IEEE Transactions on*, 12(5):651–670.

[Inglés-Romero et al., 2012] Inglés-Romero, J. F., Lotz, A., Vicente-Chicote, C., and Schlegel, C. (2012). Dealing with run-time variability in service robotics: towards a DSL for non-functional properties. In *3rd International Workshop on Domain-Specific Languages and Models for Robotic Systems (DSLRob)*, Tsukuba, Japan.

[JetBRAINS, 2015] JetBRAINS (2015). Meta Programming System. https://www.jetbrains.com/mps/.

[Kalaba et al., 2004] Kalaba, R. E., Natsuyama, H., and Udwadia, F. E. (2004). An extension of Gauss's principle of least constraint. *Int. J. of General Systems*, 33(1):63–69.

[Kallrath, 2004] Kallrath, J. (2004). *Modeling languages in mathematical optimization*, volume 88. Springer Science & Business Media.

[Kang et al., 1990] Kang, K. C., Cohen, S. G., Hess, J. A., Novak, W. E., and Peterson, A. S. (1990). Feature-oriented domain analysis (foda) feasibility study. Technical report, DTIC Document.

[Kent, 2002] Kent, S. (2002). Model driven engineering. In Butler, M., Petre, L., and Sere, K., editors, *Integrated Formal Methods*, volume 2335 of *Lecture Notes in Computer Science*, pages 286–298. Springer Berlin Heidelberg.

[Khatib, 1983] Khatib, O. (1983). Dynamic control of manipulators in operational space. In *6th IFtoMM Congr. on Theory of Machines and Mechanisms*, pages 15–20.

[Khatib, 1985] Khatib, O. (1985). The operational space formulation in robot manipulator control. In *Proceedings of the 15th International Symposium on Industrial Robots*, pages 165–172, Tokyo, Japan.

[Khatib et al., 2002] Khatib, O., Brock, O., Chang, K.-S., Ruspini, D., Sentis, L., Conti, F., and Viji, S. (2002). Efficient algorithms for robots with human-like structures and interactive haptic simulation. In Lenarčič, J. and Husty, M., editors, *8th International Symposium on Advances in Robot Kinematics*, pages 89–98, Caldes de Malavella, Spain.

[Khatib et al., 2003] Khatib, O., Brock, O., Chang, K.-S., Ruspini, D., Sentis, L., and Viji, S. (2003). Robots for the human and interactive simulations. In Huang, T., editor, *Proceedings of the 11th World Congress in Mechanism and Machine Science*, pages 1572–1576, Tianjin, China. China Machinery Press.

[Khatib et al., 2004a] Khatib, O., Brock, O., Chang, K.-S., Ruspini, D., Sentis, L., and Viji, S. (2004a). Human-centered robotics and interactive haptic simulation. *The International Journal of Robotics Research*, 23(2):167–178.

[Khatib et al., 2008] Khatib, O., Sentis, L., and Park, J.-H. (2008). A unified framework for whole-body humanoid robot control with multiple constraints and contacts. In *European Robotics Symposium*, pages 303–312, Prague, Czech Republic.

[Khatib et al., 2004b] Khatib, O., Sentis, L., Park, J.-H., and Warren, J. (2004b). Whole-body dynamic behavior and control of human-like robots. *International Journal of Humanoid Robotics*, 1(1):29–43.

[Khatib et al., 1999] Khatib, O., Yokoi, K., Brock, O., Chang, K., and Casal, A. (1999). Robots in human environments: Basic autonomous capabilities. *The International Journal of Robotics Research*, 18(7):684–696.

[Kleppe et al., 2003] Kleppe, A., Warmer, J., and Bast, W. (2003). *MDA Explained*. Addison-Wesley.

[Klotzbücher and Bruyninckx, 2012] Klotzbücher, M. and Bruyninckx, H. (2012). Coordinating robotic tasks and systems with rFSM Statecharts. *Journal of Software Engineering in Robotics*, 3(1):28–56.

[Klotzbücher et al., 2010] Klotzbücher, M., Soetens, P., and Bruyninckx, H. (2010). OROCOS RTT-Lua: an Execution Environment for building Real-time Robotic Domain Specific Languages. In *International Workshop on Dynamic languages for RObotic and Sensors*, pages 284–289.

[Klotzbüecher, 2013] Klotzbüecher, M. (2013). *Domain specific languages for hard real-time safe coordination of robot and machine tool systems.* PhD thesis, KU Leuven, Department of Mechanical Engineering.

[KUKA, 2010] KUKA (2010). Youbot store. http://www.youbot-store.com. Accessed online 1 August 2012.

[Kunze et al., 2011] Kunze, L., Roehm, T., and Beetz, M. (2011). Towards semantic robot description languages. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 5589–5595, Shangai, China.

[Lagrange, 1867] Lagrange, J. L. (1867). Mécanique analytique. In Serret, J.-A., editor, *Oeuvres*. Gauthier-Villars, Paris, France.

[Lange, 2013] Lange, C. (2013). Ontologies and languages for representing mathematical knowledge on the Semantic Web. *Semantic Web — Interoperability, Usability, Applicability*, 4(2):119–151.

[Lau et al., 2006] Lau, K.-K., Ling, L., and Wang, Z. (2006). Composing components in design phase using exogenous connectors. In *EUROMICRO '06:*

Proceedings of the 32nd EUROMICRO Conference on Software Engineering and Advanced Applications, pages 12–19. IEEE Computer Society.

[Lau and Wang, 2007] Lau, K.-K. and Wang, Z. (2007). Software component models. *IEEE Transactions on Software Engineering*, 33(10):709–724.

[Laulusa and Bauchau, 2008] Laulusa, A. and Bauchau, O. A. (2008). Review of classical approaches for constraint enforcement in multibody systems. *Transactions of the ASME, Journal of Computational and Nonlinear Dynamics*, 3(1):011004–1–8.

[Lee, 2004] Lee, E. A. (2004). Actor-oriented design: A focus on domain-specific languages for embedded systems. http://chess.eecs.berkeley.edu/pubs/362.html.

[Lee, 2006] Lee, E. A. (2006). Advanced tool architectures. http://chess.eecs.berkeley.edu/pubs/143.html.

[Lee et al., 2003] Lee, E. A., Neuendorffer, S., and Wirthlin, M. J. (2003). Actor-oriented design of embedded hardware and software systems. *Journal of Circuits, Systems, and Computers*, 12(3):231–260.

[Levine, 2009] Levine, J. (2009). *Flex & Bison: Text Processing Tools*. O'Reilly Media, Inc.

[Liegeois, 1977] Liegeois, A. (1977). Automatic supervisory control of the configuration and behavior of multibody mechanisms. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-7(12):868–871.

[Lilov and Wittenburg, 1977] Lilov, L. and Wittenburg, J. (1977). Bewegungsgleichungen für Systeme starrer Körper mit Gelenken beliebiger Eigenschaften. *Zeitschrift für Angewandte Mathematik und Mechanik*, 57(3):137–152.

[Lin and Chen, 2011] Lin, S.-T. and Chen, M.-W. (2011). A PID type constraint stabilization method for numerical integration of multibody systems. *Transactions of the ASME, Journal of Computational and Nonlinear Dynamics*, 6(1):044501–1–6.

[Luh et al., 1980a] Luh, J. Y. S., Walker, M. W., and Paul, R. P. C. (1980a). On-line computational scheme for mechanical manipulators. *Transactions of the ASME, Journal of Dynamic Systems, Measurement, and Control*, 102(2):69–76.

[Luh et al., 1980b] Luh, J. Y. S., Walker, M. W., and Paul, R. P. C. (1980b). Resolved-acceleration control of mechanical manipulators. *IEEE Transactions on Automatic Control*, 25(3):468–474.

[Manola et al., 2004] Manola, F., Miller, E., McBride, B., et al. (2004). Rdf primer. *W3C recommendation*, 10(1-107):6.

[Mansard and Chaumette, 2007] Mansard, N. and Chaumette, F. (2007). Task sequencing for sensor-based control. *IEEE Transactions on Robotics*, 23(1):60–72.

[Mansard et al., 2009] Mansard, N., Stasse, O., Evrard, P., and Kheddar, A. (2009). A versatile generalized inverted kinematics implementation for collaborative working humanoid robots: The Stack of Tasks. In *Proceedings of the 2009 International Conference on Advanced Robotics*, Munich, Germany.

[Martin and Bobrow, 1995] Martin, B. J. and Bobrow, J. E. (1995). Determination of minimum-effort motions for general open chain. In *Proceedings of the 1995 IEEE International Conference on Robotics and Automation*, pages 1160–1165, Nagoya, Japan.

[Masarati, 2011] Masarati, P. (2011). Constraint stabilization of mechanical systems in ordinary differential equations form. *Proceedings of the Institution of Mechanical Engineers, Part K: Journal of Multi-body Dynamics*, 225(1):12–33.

[Matic, 2008] Matic, S. (2008). *Compositionality in Deterministic Real-Time Embedded Systems*. PhD thesis, University of California, Berkeley.

[Medvidovic and Taylor, 1997] Medvidovic, N. and Taylor, R. N. (1997). A Framework for Classifying and Comparing Architecture Description Languages. In *Proceedings of the 6th European conference held jointly with the 5th ACM SIGSOFT international symposium on Foundations of software engineering*, pages 60–76.

[Mehta et al., 2000] Mehta, N. R., Medvidovic, N., and Phadke, S. (2000). Towards a taxonomy of software connectors. In *ICSE '00: Proceedings of the 22nd international conference on Software engineering*, pages 178–187. ACM.

[Merkle, 2010] Merkle, B. (2010). Textual modeling tools: overview and comparison of language workbenches. In *Proceedings of the ACM international conference companion on Object oriented programming systems languages and applications companion*, SPLASH '10, pages 139–148, New York, NY, USA. ACM.

[Modelica Association, 2015] Modelica Association (2015). Modelica: Language design for multi-domain modeling. http://www.modelica.org/. Last visited September 2014.

[Nakamura et al., 1987] Nakamura, Y., Hanafusa, H., and Yoshikawa, T. (1987). Task-priority based redundancy control of robot manipulators. *The International Journal of Robotics Research*, 6(2):3–15.

[Nakanishi et al., 2008] Nakanishi, J., Cory, R., Mistry, M., Peters, J., and Schaal, S. (2008). Operational space control: A theoretical and empirical comparison. *The International Journal of Robotics Research*, 27(6):737–757.

[Nakanishi et al., 2007] Nakanishi, J., Mistry, M., and Schaal, S. (2007). Inverse dynamics control with floating base and constraints. In *Proceedings of the 2007 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1942–1947, San Diego, California.

[Newton, 1871] Newton, I. (1871). *Philosophiae naturalis principia mathematica*. Maclehose, Glasgow, Scotland, reprinted edition. Eds. W. Thomson and H. Blackburn.

[Nierstrasz and Meijler, 1995] Nierstrasz, O. and Meijler, T. D. (1995). Research directions in software composition. *ACM Computing Surveys*, 27(2):262–264.

[Nordmann et al., 2014] Nordmann, A., Hochgeschwender, N., and Wrede, S. (2014). A survey on Domain-Specific Languages in robotics. In *International Conference on Simulation, Modeling, and Programming for Autonomous Robots*, volume 8810 of *Springer Lecture Notes in Computer Science*, pages 195–206.

[Ott, 2008] Ott, C. (2008). *Cartesian impedance control of redundant and flexible-joint robots*, volume 49 of *Springer Tracts in Advanced Robotics*. Springer Verlag.

[Parr, 2007] Parr, T. (2007). *The definitive ANTLR reference: building domain-specific languages*. Pragmatic Bookshelf Raleigh.

[Peters et al., 2008] Peters, J., Mistry, M., Udwadia, F., Nakanishi, J., and Schaal, S. (2008). A unifying framework for robot control with redundant DOFs. *Autonomous Robots*, 24(1):1–12.

[Philippsen et al., 2011] Philippsen, R., Sentis, L., and Khatib, O. (2011). An open source extensible software package to create whole-body compliant skills in personal mobile manipulators. In *Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on*, pages 1036–1041.

[Popov, 1974] Popov, E. P. (1974). Control of robots-manipulators. *Engineering Cybernetics*, 6.

[Popov et al., 1978] Popov, E. P., Vereshchagin, A. F., and Zenkevich, S. L. (1978). *Manipulyatsionnye roboty: dinamika i algoritmy*. Nauka, Moscow.

[Quigley et al., 2009] Quigley, M., Conley, K., Gerkey, B., Faust, J., Foote, T. B., Leibs, J., Wheeler, R., and Ng, A. Y. (2009). ROS: an open-source Robot Operating System. In *ICRA Workshop on Open Source Software*.

[Radestock and Eisenbach, 1996] Radestock, M. and Eisenbach, S. (1996). Coordination in evolving systems. In *Trends in Distributed Systems. CORBA and Beyond*, pages 162–176. Springer-Verlag.

[Radestock and Eisenbach, 2003] Radestock, M. and Eisenbach, S. (2003). Coordinating components in middleware systems. *Concurrency and Computation: Practice and Experience*, 15(13):1205–1231.

[Ragan-Kelley et al., 2013] Ragan-Kelley, J., Barnes, C., Adams, A., Paris, S., Durand, F., and Amarasinghe, S. (2013). Halide: a language and compiler for optimizing parallelism, locality, and recomputation in image processing pipelines. *ACM SIGPLAN Notices*, 48(6):519–530.

[Reiser, 2014] Reiser, U. (2014). *Eine webbasierte Integrations- und Testplattform zur Unterstützung des verteilten Entwicklungsprozesses von komplexen Serviceroboter-Applikationen*. PhD thesis, Universität Stuttgart, Holzgartenstr. 16, 70174 Stuttgart.

[Rethink Robotics, 2012] Rethink Robotics (2012). Rethink Robotics online store. http://www.rethinkrobotics.com/products/baxter/. Accessed online 1 October 2013.

[Richalet et al., 1978] Richalet, J., Rault, A., Testud, J. L., and Papon, J. (1978). Model predictive heuristic control: Applications to industrial processes. *Automatica*, 14:413–428.

[Rickert, 2015] Rickert, M. (2015). Robotics library. http://sourceforge.net/apps/mediawiki/roblib.

[Roberson and Schwertassek, 1988] Roberson, R. E. and Schwertassek, R. (1988). *Dynamics of multibody systems*. Springer-Verlag.

[Roberson and Wittenburg, 1966] Roberson, R. E. and Wittenburg, J. (1966). A dynamical formalism for an arbitrary number of interconnected rigid bodies, with reference to the problem of satellite attitude control. In *Third Congress International Federation of Automatic Control*, pages 46D.1–46.D9, Londen, U.K.

[RoboCup, 2015a] RoboCup (2015a). Robocup@home. http://www.robocupathome.org/. Last visited January 2015.

[RoboCup, 2015b] RoboCup (2015b). Robocup@work. http://www.robocupatwork.org/. Last visited January 2015.

[RoboHow, 2015] RoboHow (2015). The RoboHow Project. http://robohow.eu/.

[ROSETTA, 2015] ROSETTA (2015). Robot control for skilled execution of tasks in natural interaction with humans; based on autonomy, cumulative knowledge and learning. http://www.fp7rosetta.eu/.

[Rutgeerts, 2007] Rutgeerts, J. (2007). *Constraint-based task specification and estimation for sensor-based robot tasks in the presence of geometric uncertainty.* PhD thesis, Department of Mechanical Engineering, Katholieke Universiteit Leuven, Belgium.

[Saab et al., 2013] Saab, L., Ramos, O. E., Keith, F., Mansard, N., Souères, P., and Fourquet, J.-Y. (2013). Dynamic whole-body motion generation under rigid contacts and other unilateral constraints. *IEEE Transactions on Robotics*, 29(2):346–362.

[Saint Germain, 1900] Saint Germain, A. d. (1900). Sur la fonction *s* introduite par M. Appell dans les équations de la dynamique. *Comptes Rendus de l'Académie des Sciences de Paris*, 130:1174.

[Samson et al., 1991] Samson, C., Le Borgne, M., and Espiau, B. (1991). *Robot Control, the Task Function Approach.* Clarendon Press, Oxford, England.

[Sanatnama et al., 2008] Sanatnama, H., Ghani, A. A. A., Yap, N. K., and Selamat, M. H. (2008). Mediator Connector for Composition of Loosely Coupled Software Components. *Journal of Applied Sciences*, 8(18):3139–3147.

[Schlegel et al., 2012] Schlegel, C., Steck, A., and Lotz, A. (2012). Robotic software systems: From code-driven to model-driven software development. In Dutta, A., editor, *Robotic Systems–Applications, Control and Programming*, pages 473–502. INTECH Open Access Publisher.

[Scott, 2009] Scott, M. L. (2009). Chapter 4 - Semantic Analysis. In Scott, M. L., editor, *Programming Language Pragmatics (Third Edition)*, pages 175 – 211. Morgan Kaufmann, Boston, third edition edition.

[Sentana, 1996] Sentana, E. (1996). Econometric applications of positive rank-one modifications of the symmetric factorization of a positive semi-definite matrix. *Spanish Economic Review*, 1(1):79–90.

[Sentis, 2007] Sentis, L. (2007). *Synthesis and control of whole-body behaviors in humanoid systems.* PhD thesis, Stanford University, U.S.A.

[Shakhimardanov and Bruyninckx, 2014a] Shakhimardanov, A. and Bruyninckx, H. (2014a). JSON schema C++ parser for the semantic models of the kinematic chains. https://github.com/shakhimardanov/orocos_kdl_electric_with_extensions. Last visited September 2015.

[Shakhimardanov and Bruyninckx, 2014b] Shakhimardanov, A. and Bruyninckx, H. (2014b). JSON schema for the semantic models of the kinematic chains and geometric relations. https://gitlab.mech.kuleuven.be/rob-dsl/dsl-storage. Last visited September 2015.

[Shakhimardanov and Bruyninckx, 2015] Shakhimardanov, A. and Bruyninckx, H. (2015). A DSL for composable robot dynamics with formal semantics. submitted to Journal of Software Engineering for Robotics.

[Shakhimardanov et al., 2010] Shakhimardanov, A., Paulus, J., Hochgeschwender, N., Reckhaus, M., and Kraetzschmar, G. (2010). BRICS Deliverable-2.1:Best Practice Assessment of Software Technologies for Robotics. Technical report, University of Applied Sciences Bonn-Rhein-Sieg.

[Smits, 2010] Smits, R. (2010). *Robot skills: design of a constraint-based methodology and software support*. PhD thesis, Department of Mechanical Engineering, Katholieke Universiteit Leuven, Belgium.

[Smits et al., 2001] Smits, R., Bruyninckx, H., and Aertbeliën, E. (2001). KDL: Kinematics and Dynamics Library. http://www.orocos.org/kdl. Last visited August 2012.

[Smits et al., 2008] Smits, R., De Laet, T., Claes, K., Bruyninckx, H., and De Schutter, J. (2008). iTaSC: a tool for multi-sensor integration in robot manipulation. In *IEEE International Conference on Multisensor Fusion and Integration for Intelligent Systems*, pages 426–433, Seoul, South-Korea. MFI2008.

[Soetens, 2006] Soetens, P. (2006). *A Software Framework for Real-Time and Distributed Robot and Machine Control*. PhD thesis, Department of Mechanical Engineering, Katholieke Universiteit Leuven, Belgium. http://www.mech.kuleuven.be/dept/resources/docs/soetens.pdf.

[Sohl and Bobrow, 2001] Sohl, G. A. and Bobrow, J. E. (2001). A recursive multibody dynamics and sensitivity algorithm for branched kinematic chains. *Journal of Dynamic Systems, Measurement, and Control*, 123(3):391–399.

[Sporny et al., 2013] Sporny, M., Kellogg, G., and Lanthaler, M. (2013). Json-ld 1.0-a json-based serialization for linked data. *W3C Working Draft*.

[Stanford Robotics and AI Lab, 2011] Stanford Robotics and AI Lab (2011). Stanford whole-body control framework. https://github.com/poftwaresatent/stanford_wbc. Last visited January 2015.

[Steck and Schlegel, 2010] Steck, A. and Schlegel, C. (2010). Towards quality of service and resource aware robotic systems through model-driven software development. In *1st International Workshop on Domain-Specific Languages and models for ROBotic systems*.

[Stramigioli and Bruyninckx, 2001] Stramigioli, S. and Bruyninckx, H. (2001). Geometry of dynamic and higher-order kinematic screws. In *Proceedings of the 2001 IEEE International Conference on Robotics and Automation*, pages 3344–3349, Seoul, Korea.

[Strauss and Carey, 1992] Strauss, P. S. and Carey, R. (1992). An object-oriented 3D graphics toolkit. *ACM SIGGRAPH Computer Graphics*, 26(2):341–349.

[Taylor et al., 2009] Taylor, R. N., Medvidovic, N., and Dashofy, E. M. (2009). *Software Architecture: Foundations, Theory, and Practice*. Wiley Publishing.

[Udwadia and Kalaba, 2002] Udwadia, F. E. and Kalaba, R. E. (2002). On the foundations of analytical dynamics. *International Journal of non-linear mechanics*, 37(6):1079–1090.

[Universal Robot, 2015] Universal Robot (2015). UR5 technical specification. http://www.universal-robots.com/.

[Van den Broeck et al., 2011] Van den Broeck, L., Diehl, M., and Swevers, J. (2011). Experimental validation of time optimal mpc on a flexible motion system. In *American Control Conference (ACC), 2011*, pages 4749–4754.

[van Deursen et al., 2000] van Deursen, A., Klint, P., and Visser, J. (2000). Domain-specific languages: an annotated bibliography. *SIGPLAN Not.*, 35(6):26–36.

[Vanthienen, 2015] Vanthienen, D. (2015). *Composition Pattern for Constraint-based Programming with Application to Force-sensorless*. PhD thesis, KU Leuven, Department of Mechanical Engineering.

[Vanthienen et al., 2011a] Vanthienen, D., De Laet, T., Decré, W., Smits, R., Klotzbücher, M., Buys, K., Bellens, S., Gherardi, L., Bruyninckx, H., and De Schutter, J. (2011a). itasc as a unified framework for task specification, control, and coordination, demonstrated on the pr2. demonstration IEEE/RSJ International Conference on Intelligent Robots and Systems.

[Vanthienen et al., 2011b] Vanthienen, D., De Laet, T., Smits, R., and Bruyninckx, H. (2011b). iTaSC Software. http://www.orocos.org/itasc. Last visited May 2014.

[Vanthienen et al., 2014] Vanthienen, D., Klotzbücher, M., and Bruyninckx, H. (2014). The 5C-based architectural Composition Pattern: lessons learned from re-developing the iTaSC framework for constraint-based robot programming. *Journal of Software Engineering in Robotics*, 5(1):17–35.

[Vereshchagin, 1974] Vereshchagin, A. F. (1974). Computer simulation of the dynamics of complicated mechanisms of robot-manipulators. *Engineering Cybernetics*, 12(6):65–70.

[Vereshchagin, 1989] Vereshchagin, A. F. (1989). Modelling and control of motion of manipulational robots. *Soviet J. of Computer and Systems Sciences*, 27(5):29–38. Originally published in Izvestiia Akademii nauk SSSR, Tekhnicheskaya Kibernetika, No. 1, pp. 125–134, 1989.

[Verscheure, 2009] Verscheure, D. (2009). *Contributions to Contact Modeling and Identification, and Optimal Robot Motion Planning.* PhD thesis, Department of Mechanical Engineering, Katholieke Universiteit Leuven, Belgium.

[Voelter et al., 2013] Voelter, M., Benz, S., Dietrich, C., Engelmann, B., Helander, M., Kats, L. C., Visser, E., and Wachsmuth, G. (2013). *DSL engineering: Designing, implementing and using domain-specific languages.* dslbook. org.

[Voelter and Groher, 2007] Voelter, M. and Groher, I. (2007). Handling variability in model transformations and generators. In *7th OOPSLA Workshop on Domain-Specific Modeling.*

[Voelter et al., 2007] Voelter, M., Groher, I., and Kolb, B. (2007). Mechanisms for expressing variability in models and mdd tool chains. *MDSD in Embedded Systems.*

[Web3D Consortium, 2008] Web3D Consortium (2008). X3D — Extensible 3D (ISO/IEC FDIS 19775-1:2008). http://www.web3d.org/x3d/specifications/ISO-IEC-19775-1.2-X3D-AbstractSpecification/.

[Whitney, 1969] Whitney, D. E. (1969). Resolved motion rate control of manipulators and human prostheses. *IEEE Transactions on Man-Machine Systems*, 10(2):47–53.

[Willow Garage, 2009] Willow Garage (2009). Universal Robot Description Format (URDF). http://www.ros.org/wiki/urdf.

[Wittenburg, 1977] Wittenburg, J. (1977). *Dynamics of systems of rigid bodies*. Teubner, Stuttgart, Germany.

[Wyrobek et al., 2008] Wyrobek, K. A., Berger, E. H., Van der Loos, H. F. M., and Salisbury, J. K. (2008). Towards a personal robotics development platform: Rationale and design of an intrinsically safe personal robot. In *Proceedings of the 2008 IEEE International Conference on Robotics and Automation*, pages 2165–2170, Pasadena, California, U.S.A.

[Ziena, 2015] Ziena (2015). KNITRO nonlinear optimization solver. http://www.ziena.com/knitro.htm.

[Zyp et al., 2013] Zyp, K., Galiegue, F., et al. (2013). Json schema: core definitions and terminology. http://json-schema.org/latest/json-schema-core.html.

Publications

- D. Brugali and A. Shakhimardanov. Component-based Robotic Engineering. Part II: Models and systems. *In IEEE Robotics and Automation Magazine*, volume 17, pages 100-112, March 2010.
- A. Shakhimardanov, N. Hochgeschwender, and G. Kraetzschmar. Component Models in Robotics Software. *In Proceedings of the Workshop on Performance Metrics for Intelligent Systems (PerMIS)*, September 2010, Baltimore, USA.
- A. Shakhimardanov, N. Hochgeschwender, M, Reckhaus and G. K. Kraetzschmar. Analysis of software connectors in robotics. *In Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, September 2011, San Francisco, CA, USA.
- N. Hochgeschwender, L. Gherardi, A. Shakhirmardanov, G. K. Kraetzschmar, and H. Bruyninckx. A Model-based Approach to Software Deployment in robotics, *In Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, November 2013, Tokyo, Japan.
- A. Shakhimardanov, H. Bruyninckx. Vereshchagin's algorithm for the linear-time hybrid dynamics and control with weighted or prioritized partial motion constraints in tree-structured kinematic chains, *Humanoids Workshop on Torque-Controlled*, October 2013, Atlanta, GA, USA.
- Markus Klotzbuecher, Nico Hochgeschwender, Luca Gherardi, Herman Bruyninckx, Gerhard Kraetzschmar, Davide Brugali, Azamat Shakhimardanov, Jan Paulus, Michael Reckhaus, Hugo Garcia, Davide Faconti and Peter Soetens. The BRICS Component Model: A Model-Based Development Paradigm For Complex Robotics Software Systems. *In Proceedings of the 28th ACM Symposium on Applied Computing (SAC). Track on Software Architecture: Theory, Technology, and Applications (SA-TTA).*, 2013, Coimbra, Portugal.

- A. Shakhimardanov, H. Bruyninckx. Design and development of a composable DSL for robot kinematics and dynamics conforming to formal semantic models: lessons learned, submitted to *Journal of Software Engineering for Robotics (JOSER)*, 2015.

- Enea Scioni, Herman Bruyninckx, Azamat Shakhimardanov, Markus Klotzbucher, Hugo Garcia, Sebastian Blumenthal. Hierarchical Hypergraphs for Knowledge-centric Robot Systems: a Composable Structural Meta Model and its Domain-Specific Language NPC4, submitted to *Journal of Software Engineering for Robotics (JOSER)*, 2015.

- Michael Reckhaus, Nico Hochgeschwender, Jan Paulus, Azamat Shakhimardanov and Gerhard K. Kraetzschmar. An Overview about Simulation and Emulation in Robotics. In Proceedings of the Workshop on Simulation Technologies in the Robot Development Process. *In Proceedings of the Workshop on Simulation Technologies in the Robot Development Process, 2nd International Conference on Simulation, Modeling, and Programming for Autonomous Robots (SIMPAR)* November 2010, Darmstadt, Germany.

- Gerhard K. Kraetzschmar, Azamat Shakhimardanov, Michael Reckhaus, Jan Paulus and Nico Hochgeschwender. On the role of simulation in the robot development process. *In Proceedings of the Workshop on Simulation Technologies in the Robot Development Process, 2nd International Conference on Simulation, Modeling, and Programming for Autonomous Robots (SIMPAR)*, November 2010, Darmstadt, Germany.

- N. Hochgeschwender and A. Shakhimardanov. Component-Based Robotics Middleware. *Invited talk at SDIR Tutorial on Component-Based Robotics Engineering, IEEE/RAS International Conference on Robotics and Automation (ICRA)*, May 2010, Anchorage, USA.

FACULTY OF ENGINEERING SCIENCE
DEPARTMENT OF MECHANICAL ENGINEERING
PRODUCTION ENGINEERING, MACHINE DESIGN AND AUTOMATION DIVISION
Celestijnenlaan 300 box 2420
B-3001 Heverlee
info@mech.kuleuven.be
http://www.mech.kuleuven.be