

KATHOLIEKE UNIVERSITEIT
LEUVEN

Arenberg Doctoraatsschool Wetenschap & Technologie
Faculteit Ingenieurswetenschappen
Departement Werktuigkunde

ROBOT SKILLS

Design of a constraint-based methodology and software support

Ruben Smits

Proefschrift voorgedragen
tot het behalen van de
graad van Doctor in de
Ingenieurswetenschappen

Mei 2010

ROBOT SKILLS

Design of a constraint-based methodology and software support

Ruben Smits

Jury:
Prof. dr. ir. Ann Haegemans, voorzitter preliminaire
Prof. dr. ir. Adhemar Bultheel, voorzitter publieke
Prof. dr. ir. Joris De Schutter, promotor
Prof. dr. ir. Herman Bruyninckx, promotor
Prof. dr. ir. Joost Duflou, assessor
Prof. dr. ir. Jos Vander Sloten, assessor
Prof. dr. ir. Johan Baeten
(Katholieke Hogeschool Limburg)
Prof. dr. Michael Beetz
(Technische Universität München)
Prof. dr. Klas Nilsson
(Lunds Tekniska Högskola)

Proefschrift voorgedragen
tot het behalen van de
graad van Doctor in de
Ingenieurswetenschappen

Mei 2010

© Katholieke Universiteit Leuven – Faculty of Engineering
Kasteelpark Arenberg 1, B-3001 Leuven (Belgium)

Alle rechten voorbehouden. Niets uit deze uitgave mag worden vermenigvuldigd en/of openbaar gemaakt worden door middel van druk, fotokopie, microfilm, elektronisch of op welke andere wijze ook zonder voorafgaande schriftelijke toestemming van de uitgever.

All rights reserved. No part of the publication may be reproduced in any form by print, photoprint, microfilm or any other means without written permission from the publisher.

D/2010/7515/43
ISBN 978-94-6018-204-4

Bedankt!

If we knew what it was we were doing, it would not be called research, would it?

Albert Einstein (1879–1955)

In 2002 had ik de kandidaturen van mijn ingenieurstudies afgerond. Ik moest toen kiezen welke richting ik zou uitgaan, en ik twijfelde hard tussen werktuigkunde, elektrotechniek en computerwetenschappen. Uiteindelijk koos ik voor werktuigkunde, ik kan mij echter niet meer herinneren waarom. In 2004, na vier jaar ingenierstudies had ik nog geen enkel idee wat ik het jaar erna, na mijn vijfde en laatste jaar aan de unief, ging doen. Als bij wonder viel alles op zijn plaats gedurende dat laatste jaar. Ik kwam, samen met mijn thesispartner Jan Mathijssen, gedurende mijn Masterthesis in contact met robotica. Jan en ik moesten één van de oudste en grootste nog werkende robots van het departement "hacken" zodat we hem konden aansturen met een gewone computer. Gedurende dat jaar werd mij duidelijk dat robotica een zeer veelzijdig domein is. Robotica brengt werktuigkunde, elektrotechniek en computerwetenschappen tesamen. Ik was direct verkocht, als de kans zich voordeed om iets met robotica te kunnen blijven doen zou ik geen seconde twijfelen.

En zo geschiedde, de promotoren van mijn masterthesis, Joris De Schutter en Herman Bruyninckx stelden me voor om een doctoraat te beginnen. Ik ben ze

ontzettend dankbaar voor deze kans en voor de vrijheid bij het invullen van die kans.

Er zijn uiteraard nog hoop andere mensen die ik wil bedanken:

Mijn juryleden voor de kritische en constructieve opmerkingen bij het lezen van mijn tekst en tijdens de preliminaire verdediging.

Johan Rutgeerts, Wim Meeussen, Klaas Gadeyne en Peter Soetens om me onder te dompelen in de wondere wereld van Linux, programmeren en C++. Ik heb gigantisch veel geleerd van hen.

Mijn directe collega's voor de toffe sfeer in onze onderzoeksgroep en bureau, Tinne, om altijd eens een kritische blik te werpen op mijn papers en me te helpen bij het "debuggen" van men software; Wilm, die altijd zijn "closed-source" software ter beschikking kon stellen op die zeldzame momenten dat ik die nodig had en natuurlijk me altijd te corrigeren als ik weer maar eens een overduidelijke spellingsfout maak; Diederik, om af en toe eens bij te praten over de koers; Hans, die zijn hand niet omdraait om een hardware probleempje te fixen; Markus, to help me out with all my weird linux problems en natuurlijk ook de nieuwkomers, Steven, Koen en Nick om de sfeer in onze groep nog wat op te fleuren.

Michaël en Tinne, Ilse en Tine voor de Herentse (spelletjes-)avonden.

"Onze Pa" en "ons Ma" voor de kansen die ze me altijd gegeven hebben. En natuurlijk ook onze Maarten, onze Joris en ons Hannelore.

Dit werk is opgedragen aan Eef, Lotte en Dries. Zij zijn degene die ervoor zorgen dat ik thuis de nodige ontspanning heb, en me de drive bezorgen om er elke dag weer keihard tegenaan te gaan.

Ruben Smits
Leuven, 17 Mei 2010

Abstract

The days when robots were only used to move their endpoint from point A to B, passing through point C, safely hidden behind a fence, are over. Nowadays robots have to grasp objects, move them around, manipulate them using two arms, while keeping controlled distances from humans and fragile objects or even during physical interaction with humans. Furthermore, all these tasks have to be executed as optimally as possible. Six degrees-of-freedom industrial arms are being replaced by mobile platforms with two redundant arms, both equipped with force sensors, three-fingered grippers with touch sensors, a head with cameras, laser scanners, inertial sensors, etc. In the coming years the habitat of robotic systems will evolve from the industrial work cells to domestic, cluttered and populated environments. To cope with the increasing complexity, not only a highly modular methodology for motion specification is needed. Also the online motion coordination becomes very important since it has to cope with the dynamically changing environment in which the robotic system works.

This thesis presents a highly modular control hierarchy for the coordination of constraint-based motion specification that can be used with a high variety of sensors, robotic systems and tasks. To create a highly modular coordination system, a modular motion specification methodology needs to be used as the underlying framework. iTaSC, or instantaneous Task Specification using Constraints, is such a modular framework that allows us to specify motion by imposing those constraints on the interaction between the robot and the environment that are important for the task at hand. This thesis extends the iTaSC methodology to not only allow the specification of constraints in feature spaces, which are built up from exactly six independent geometric coordinates, but to allow the specification of constraints in any feature space for which an interaction model can be found. Calculating the robot motion from the specified constraints can be very computationally expensive in systems that include a lot of actuated joints, such as humanoid robots. Therefore the quest for efficient algorithms is still ongoing. This thesis discusses one of the most ground-breaking algorithms for acceleration-based constrained hybrid (combination of forward and inverse) dynamics, and explains how it can be

extended for tree-like kinematic robot structures and how it can be used to solve an iTaSC motion specification.

Building software support for the new generation of robot applications from scratch for each new robotic system, or each new application is a very elaborate task. Therefore reusable software components and formal representations of both the robotic system and the task to be executed are mandatory. This thesis presents a component-based modular architecture for the next generation of robot controllers based on the extended iTaSC motion specification and coordination. A formal representation for robotic systems and their motion specification is presented to facilitate the creation of interchangeable file formats for the specification of robot skills.

Beknopte samenvatting

De tijd waarin robots enkel gebruikt werden om hun eindpunt te bewegen van punt A naar punt B, via punt C, veilig achter een hekwerk, is voorgoed voorbij. Tegenwoording moeten robots objecten grijpen, verplaatsen en manipuleren met twee armen. Tegelijkertijd moeten de robots op een gecontroleerde en veilige afstand blijven van mensen en kwetsbare objecten in hun omgeving, of bewegen tijdens fysieke interactie met mensen. Deze taken moeten bovendien allemaal zo optimaal mogelijk uitgevoerd worden. Industriële robotarmen met zes vrijheidsgraden worden vervangen door mobiele platformen met twee redundante armen, beide uitgerust met krachtsensoren, grijpers met drie vingers en contactsensoren, een hoofd met cameras, laserscanners, inertiële sensoren, etc. In de komende jaren zal de werkomgeving van robotsystemen evolueren van de industriële werkcel naar huiselijke, ongestructureerde en bevolkte omgevingen. Om het hoofd te bieden aan deze toenemende complexiteit is niet alleen een zeer modulaire methodologie voor bewegingsspecificatie nodig, ook de online bewegingscoördinatie wordt zeer belangrijk om met de dynamisch veranderende omgeving waarin het robotsysteem moet werken om te gaan.

Dit proefschrift presenteert een zeer modulaire controlehiërarchie voor de coördinatie van beperkingsgebaseerde bewegingsspecificatie, wat we robotvaardigheden noemen. De gepresenteerde controlehiërarchie is geschikt voor een grote verscheidenheid aan robotsystemen, sensoren en taken. Om modulaire robotvaardigheden mogelijk te maken, moet onderliggend een modulaire bewegingsspecificatiemethodologie gebruikt worden. iTaSC, of *instantaneous Task Specification using Constraints* (ogenblikkelijke taakspecificatie door middel van beperkingen), is zo'n modulaire methodologie die ons in staat stelt om gewenste beweging te specifiëren door enkel die beperkingen op te leggen op de interactie tussen de robot en zijn omgeving, die belangrijk zijn voor de huidige taak.

Dit proefschrift breidt de iTaSC methodologie uit zodat niet alleen geometrische beperkingen in kenmerkruimtes, opgebouwd uit exact zes onafhankelijke geometrische coördinaten, opgelegd kunnen worden, maar ook beperkingen

gespecifieerd kunnen worden in eender welke kenmerkruimte waarvoor een interactiemodel opgesteld kan worden.

De berekening van een robotbeweging uit de gespecifieerde beperkingen kan veel rekentijd vragen voor robotsystemen die veel bekrachtigde gewrichten bevatten, zoals humanoïde robots. Daarom is de zoektocht naar efficiënte algoritmen nog steeds aan de gang. Dit proefschrift behandelt één van de meest baanbrekende algoritmen voor versnellings- en beperkingsgebaseerde hybride dynamica, en legt uit hoe het algoritme uitgebreid kan worden voor kinematische boomstructuren, en hoe het gebruikt kan worden om een iTaSC-bewegingspecificatie op te lossen.

Steeds van nul beginnen om software te ontwerpen voor elke nieuwe (complexe) toepassing of voor elk nieuw (complex) robotsysteem, is een zeer omvangrijke taak. Daarom zijn herbruikbare softwarecomponenten en formele representaties nodig voor zowel de robotsystemen als de taken die uitgevoerd moeten worden. Dit proefschrift introduceert een componenten-gebaseerde, modulaire architectuur voor de volgende generatie van robotcontrolemodules. De voorgestelde architectuur is gebaseerd op de uitgebreide iTaSC-bewegingsspecificatie en -coördinatie. Dit proefschrift bevat tevens een formele representatie voor robotsystemen en hun bewegingsspecificatie om de creatie van uitwisselbare bestandsformaten voor de specificatie van robotvaardigheden te vergemakkelijken.

Symbols, definitions and abbreviations

General abbreviations

1D, 2D, 2.5D, 3D, 6D	: 1-, 2-, 2.5-, 3- or 6 - dimensional
API	: application programming interface
COLLADA	: collaborative design activity
CPU	: central processing unit
DAG	: directed acyclic graph
DOF	: degree of freedom
DLS	: damped least squares
EE	: robot end effector
FPK	: forward position kinematics
FSM	: finite state machine
GUI	: graphical user interface
IBVS	: image based visual servoing
ID	: inverse dynamics
IPK	: inverse position kinematics
iTaSC	: instantaneous task specification using constraints
KDL	: kinematics and dynamics library
LGPL	: lesser general public license
MathML	: mathematical markup language
OMG	: object managment group
OROCOS	: open robot control software
PD	: proportional derivative
ROS	: robot operating system
RTT	: realtime toolkit
SDLS	: selectively damped least squares
SRI	: singularity robust inverse
STL	: standard template library
SVD	: singular value decomposition
UML	: unified modeling language

URDF	: unified robot description format
URI	: universal resource identifier
XML	: extensible markup language

General symbols and definitions

$\boldsymbol{T}_b^a$	:	homogeneous transformation between frame a and frame b
${}_c\mathbf{t}_b^a$	:	velocity of a with respect to b expressed in reference frame c

iTaSC

$\boldsymbol{q}$	:	robot manipulator joint coordinates
$\boldsymbol{q}_I$	:	joint coordinates of the robot manipulator to which the first object is attached
$\boldsymbol{q}_{II}$	:	joint coordinates of the robot manipulator to which the second object is attached
$\boldsymbol{\chi_f}$	:	auxiliary coordinates introduced by a spatial relation between two objects
$\boldsymbol{\chi_{fI}}$	:	auxiliary coordinates between first object and its feature
$\boldsymbol{\chi_{fII}}$	:	auxiliary coordinates between features of different objects
$\boldsymbol{\chi_{fIII}}$	:	auxiliary coordinates between second object and its feature
$\boldsymbol{\chi_u}$	:	uncertainty coordinates
$\boldsymbol{\chi_{uI}}$	:	position uncertainty coordinates of first object
$\boldsymbol{\chi_{uII}}$	:	shape uncertainty coordinates between first object and its feature
$\boldsymbol{\chi_{uIII}}$	:	shape uncertainty coordinates between second object and its feature
$\boldsymbol{\chi_{uIV}}$	:	position uncertainty coordinates of second object
$\boldsymbol{l}$	:	loop equation
$\boldsymbol{J_f}$	:	feature jacobian
$\boldsymbol{J_q}$	:	robot jacobian
$\boldsymbol{J_u}$	:	uncertainty jacobian
$\boldsymbol{y}$	:	output variables
$\boldsymbol{f}$	:	output equation
$\boldsymbol{C_f}$	:	auxiliary coordinate selection matrix
$\boldsymbol{C_q}$	:	robot manipulator joint coordinate selection matrix
$\boldsymbol{y}_d$	:	desired output values
$\boldsymbol{A}$	:	task specification constraint matrix
$\boldsymbol{B}$	:	task specification uncertainty matrix
$\boldsymbol{P}, \boldsymbol{Q}$	:	extended linearized system model
$\boldsymbol{P}_{robot}$	:	linearized robot process model
$\boldsymbol{P}_{uncert}$	:	linearized uncertainty process model

$\boldsymbol{h}$	:	measurement equation
$\boldsymbol{M}_q$	:	joint space minimization norm
$\boldsymbol{M}_c$	:	constraint space minimization norm
$\boldsymbol{u}$	:	robot manipulator control input

Extending iTaSC with constraints in any feature space

$\boldsymbol{h}$	:	interaction model
$\boldsymbol{H}$	:	interaction matrix
$\boldsymbol{p}$	:	3D coordinates of a point in the camera frame
$\boldsymbol{m}$	:	normalized coordinates of a point in the camera frame
$\boldsymbol{K}$	:	intrinsic camera parameter matrix

Skills

c_i	:	$\langle \texttt{name}; \boldsymbol{f}; \boldsymbol{K}; \boldsymbol{\gamma}_i \rangle$, constraint
$\boldsymbol{K}$	:	constraint controller
$\boldsymbol{\gamma}$	:	constraint controller parameters
p_i	:	priority of constraint i
w_i	:	weight of constraint i
P_j	:	priority of skill j
W_j	:	weight of skill j
σ	:	solver parameters

Linear-time solver for acceleration-based constraints

$\dot{\boldsymbol{X}}_i$	:	velocity of segment i
$\ddot{\boldsymbol{X}}_i$	:	acceleration of segment i
$\ddot{\boldsymbol{X}}_i^b$	:	bias acceleration of segment i
$\boldsymbol{Z}_i$	:	unit velocity along the joint axis of segment i
$\boldsymbol{F}_i$	:	force on segment i
$\boldsymbol{F}_i^b$	:	bias force on segment i
$\boldsymbol{F}_i^e$	:	external force on segment i
$\boldsymbol{M}_i$	:	rigid-body inertia matrix of segment i
$\boldsymbol{M}_i^a$	:	articulated-body inertia matrix of segment i
τ_i	:	applied joint torque at the joint of segment i
$\boldsymbol{A}_i$	:	unit constraint force matrix at segment i
$\boldsymbol{\beta}_i, \boldsymbol{b}_N$	:	constraint acceleration energy at segment i
$\boldsymbol{\nu}$	:	constraint force magnitudes
$\boldsymbol{\mathcal{Z}}_i$	:	acceleration constraint coupling matrix

Contents

Bedankt! i

Abstract iii

Beknopte samenvatting v

Symbols, definitions and abbreviations vii

Contents xi

List of Figures xv

1 Introduction 1

1.1 Motivation 2

1.2 Background 3

1.3 Contributions 9

1.4 Overview of the thesis 12

2 Literature survey 15

2.1 Robot motion specification 16

2.2 Robot motion coordination 19

2.3 Conclusion 21

3 Instantaneous constraint-based motion specification 23

3.1 Example application 24
3.2 Modeling spatial relations between objects 27
3.3 Configuring the desired motion and estimation 36
3.4 Calculating the robot motion from the constraint definition . . 40
3.5 Conclusion 43

4 Extending iTaSC with constraints in any feature space 45

4.1 Definition of a constraint in a feature space 45
4.2 Example: Constraint-based image-based visual servoing 47
4.3 Conclusion 52

5 Skills 55

5.1 Hierarchical control structure 56
5.2 Skill controller view on the iTaSC motion specification 60
5.3 iTaSC configuration interface 62
5.4 Skill representation and specification 65
5.5 Example application 69
5.6 Task controller 78
5.7 Conclusion 79

6 Linear-time solver for acceleration-based constraints 81

6.1 Introduction 81
6.2 Segment-to-segment recursions 83
6.3 Dynamics algorithms 87
6.4 Acceleration constraints 87
6.5 Examples of different use-cases 95
6.6 Using togheter with iTaSC 96

6.7 Comparison with priority-based approach with full acceleration constraints . . . 100
6.8 Conclusion . . . 101

7 Software architecture for robot skills 103
7.1 Introduction . . . 104
7.2 Component-based software architecture for iTaSC . . . 104
7.3 Deployment of the iTaSC Scene . . . 112
7.4 Motion Coordination . . . 114
7.5 Example . . . 115
7.6 Orocos RTT-based implementation . . . 118
7.7 Conclusion . . . 119

8 Formal representation of the Robotics Scene Graph 121
8.1 Introduction . . . 122
8.2 The Robotics Scene Graph Standard . . . 123
8.3 COLLADA . . . 131
8.4 Extensions and other standards . . . 134
8.5 Conclusion . . . 137

9 Kinematics and dynamics software library for robotics 139
9.1 Introduction . . . 140
9.2 Three levels of use . . . 141
9.3 Four levels of representation . . . 142
9.4 The natural classes in KDL . . . 145
9.5 Conclusion . . . 148

10 General conclusions 151
10.1 Main contributions . . . 151
10.2 Outlook . . . 154

References **157**

Curriculum vitae **169**

List of Figures

1.1 In every robotic application there is an action to be executed by a robot in a given environment. 4

1.2 Graphical overview of the different chapters of the thesis, grouped by theme. 12

3.1 Overview of the experimental setup for the human-aware force-controlled manipulation task showing both robots a human and the laser scanner 24

3.2 Detail of the experimental setup for the human-aware force-controlled manipulation task. The detail shows all the sensors that are used in the application, the force sensor and the laser distance sensor and the camera that is used to point to the closest human in the environment. 25

3.3 The "scene graph" of the example application: both robots, the probe, all the sensors and objects are visualized and how they are connected to each other. Black solid lines are fixed transformations, blue dotted lines are partially uncertain transformations and red dashed lines are transformations controlled by the robotic manipulators control input. 26

3.4 Resulting virtual kinematic loop created by a spatial relation between object and feature frames. 27

3.5 The resulting graph for a modeled virtual kinematic loop between objects $\{o1\}$ and $\{o2\}$, including the robot manipulator coordinates, $\boldsymbol{q}$, the spatial relation coordinates, $\boldsymbol{\chi_f}$ and the modeled uncertainty coordinates $\boldsymbol{\chi_u}$. 28

3.6 Overview of the most important auxiliary coordinates and the uncertainty coordinates modeled for the force-controlled manipulation between a probe ($o2$) and an unknown 2.5D object ($o1$). x_c, y_c and θ_c depend on the model of the contour. 30

3.7 The "scene graph" of the example application including the spatial relation (green dash-dotted line) introduced by the unknown contour tracking skill. 31

3.8 Object and feature frames modeled for the spatial relation between a laser and a plane. 32

3.9 The "scene graph" of the example application including the spatial relation (green dash-dotted line) introduced by the plane estimating skill. 33

3.10 Spatial relation introduced to point a camera ($o1$) to a person ($o2$). 34

3.11 The "scene graph" of the example application including the spatial relation (green dash-dotted line) introduced by the camera pointing skill. 35

3.12 The spatial relation introduced between the $\{contact\}$ frame ($o1$) and the $\{person\}$ frame ($o2$) to model the projected distance. . 36

3.13 The "scene graph" of the example application including the virtual manipulator (green dash-dotted line) introduced by the keep distance skill. 37

3.14 The "scene graph" of the example application including the virtual manipulators (green dash-dotted lines) introduced by the different skills. 38

4.1 Overview of the experiment: the camera is attached to the robot end effector $\{ee\}$; target points t_i are projected by the camera in the *image* plane. The desired positions are represented by the red dots. The dotted line represents the vertical wall, which is located at $\{wall\}$. $\{project\}$ represents the projection of the robot end effector on the wall. The dashed line represents the security border b, which triggers the distance constraint. 48

4.2 X-position of the robot end effector. It is clear that the constraint on the distance from the wall only becomes active after 90 seconds and successfully keeps the robot end effector at a safe distance of $0.05m$ from the wall which is located at $-1m$. . 51

4.3 Normalized position of the tracked points in the normalized camera-image, the crosses are the desired positions, the rectangle describes the image edge boundary for the activation/deactivation of the tracking constraints. 52

4.4 The camera image with the four tracked points, in green the actual points and in orange the edge boundary which activates the tracking constraints. 53

5.1 Skill-based control system, with decreasing intelligence and increasing control bandwidth from left to right. 56

5.2 Skill-based control system, with decreasing intelligence and increasing control bandwidth from left to right, adapted to include an extra layer to enable full configuration and coordination of existing industrial robot systems. 57

5.3 A UML state machine representation for the unknown object's contour following skill. 72

5.4 A UML state machine representation for the plane estimating skill. 74

5.5 The UML state machine off the camera pointing skill. 76

6.1 Reference frames and notation for segments in a tree-structured kinematic chain. "p_i" is the "proximal" frame of segment i, and "d_i" is the segment's distal frame. The frames' $\boldsymbol{Z}_i$ and $\boldsymbol{Z}_{i+1}$ axes lie along the (single degree of freedom) axes of the joints connected to the segment. 84

6.2 Overview of the input and output of the presented hybrid dynamic algorithm. 88

6.3 The two-link mechanism used in the presented experiments in its initial position. 95

6.4 Torques to be applied to satisfy the constraints in the four different simulation experiments: (i) both X and Y are constrained, Y is controlled to a new desired position, (ii) only X is constrained and a virtual spring is added on the first joint, (iii) only X is constrained and an external force is applied on the second segment, (iv) only Y is constrained and controlled to a new desired position. Notice that in the simulation experiments without the external force the torques all go back to zero, but in the case of the external force a remaining torque is needed to satisfy the constraint. 97

6.5 Resulting joint positions in the four different experiments: (i) both X and Y are constrained, Y is controlled to a new desired position, (ii) only X is constrained and a virtual spring is added on the first joint, notice how q_1 goes to π (iii) only X is constrained and an external force is applied on the second segment, (iv) only Y is constrained and controlled to a new desired position. Notice that in spite of the similar resulting EE motion of the first three experiments the joint motion is different. 98

6.6 The resulting motion of the end effector of the two link mechanism in the four different simulation experiments: (i) both X and Y are constrained, Y is controlled to a new desired position, (ii) only X is constrained and a virtual spring is added on the first joint, (iii) only X is constrained and an external force is applied on the second segment, (iv) only Y is constrained and controlled to a new desired position. It is clearly shown that only in the last experiment where no constraint is active on the X-direction the EE is moving in the X-direction. 99

7.1 Overview of the iTaSC component structure, it shows the communication data-flow between all the components in case of a velocity resolved control scheme. 108

7.2 The different components and their data flow connections for the "human aware force controlled manipulation using two robots" example. Only the components inside the dashed iTaSC block are iTaSC related. The blue boxed components are *Robot/Object* components, the green boxed components are *Virtual Manipulator* components and the red boxed components are the *Constraint* components. The external components take care of the hardware interface. 116

7.3 The RealTime Toolkit Application Stack: an Orocos component is built upon the RealTime Toolkit (RTT) library. 119

7.4 Orocos TaskContext: Data flows through ports and is manipulated by algorithms in the component. 120

Chapter 1

Introduction

The great liability of the engineer compared to men of other professions is that his works are out in the open where all can see them. His acts, step by step, are in hard substance. He cannot bury his mistakes in the grave like the doctors. He cannot argue them into thin air or blame the judge like the lawyers. He cannot, like the architects, cover his failures with trees and vines. He cannot, like the politicians, screen his sort-comings by blaming his opponents and hope the people will forget. The engineer simply cannot deny he did it. If his works do not work, he is damned.

Herbert Hoover (1874 - 1964)

1.1 Motivation

Until recently the main working environment of a robot was an industrial work cell. Complete automation of the robot task is the main design goal for robot tasks in this industrial environment. This automation is usually achieved by adapting the working environment to the robot and make it deterministic so that the robot control system does not have to deal with uncertainties.

Robotic applications however are emerging from their industrial environment towards domestic environments. In these domestic environments the robot has to be able to perform dozens of different tasks in a dynamically changing and unknown environment. This leads to an increasing kinematic complexity of the robot platforms, to maximize the number of tasks they are able to perform. The robot platforms are made mobile using legs or a wheeled base, and usually consist of multiple redundant human-sized robot-arms equipped with multi-fingered hands. To cope with the dynamically changing and unknown environment, the robots are equipped with a variety of sensors: laser scanners, cameras, force sensors, inertial sensors, etc. The variety and number of sensors increases the complexity of the robot applications, since the sensors have to be used to *understand* and *learn* what is going on in the environment.

The objective of this thesis is to design a methodology that can be used to specify complex robot tasks for the new generation of robots. The methodology takes advantage of the kinematic redundancy of the robots and the variety of sensors. Furthermore, the methodology should enable the design of modular and reusable subtasks each coordinating a specific set of desired motions between a specific set of objects.

Specifying the desired motion of these new kinds of robotic applications differs a lot from the motion specification of a robot application in the traditional industrial setting. We want to maximize the reusability of the developed functionality and the motion specification design. To make this possible, a modular motion specification methodology has to be used. Firstly, this methodology must be able to specify the desired motion in terms of the objects and the actions between these objects that are related to a specific part of the task. As such, we can reuse the motion specification on different robotic platforms. Secondly, the motion specification methodology must allow a minimal specification of the desired motion, this means we do not want to over-specify the desired motion and hence enable exploitation of the kinematic redundancy of the robot in view of optimizing the motion or executing another simultaneous motion that is related to the overall task.

For instance, a force-controlled manipulation task between two objects only constrains six degrees of freedom. If this task is performed by two six degree of

freedom robots, each holding one of the objects, another six degrees of freedom are left unspecified. These remaining degrees of freedom can for instance be used for a second task in which a camera, held by one of the robots, has to be pointed towards a person in the environment. This second task only constrains three degrees of freedom between the camera and the person. Therefore, a third task can be added that keeps the manipulation at a certain minimal safe distance from a person in the environment. This third task only specifies one degree of freedom.

The example application described above illustrates the need for and the possibilities of the motion specification framework we have in mind. Tasks are designed without knowing which type of robot will execute them. Complete applications can be designed by gluing multiple subtasks together that can be executed simultaneously on a specific robot. A standardized file format for the specification of the desired motion will even increase the interchangeability of the motion specifications which are designed for the execution of a specific skill.

The coordination of the desired motion in such a modular motion specification framework is very important. Especially when multiple motion specifications are active in parallel, the difference between the desired motion and the actual motion for each specification has to be monitored and appropriate action has to be taken if necessary. This coordination is essential in robotic applications and offers a discrete symbolic interface for the continuous low-level motion specification.

1.2 Background

1.2.1 Motion specification

The goal of part of the research within the Robotics Group of the Mechanical Department at the Katholieke Universiteit Leuven is to develop *programming support* for the implementation of complex, sensor-based robotic tasks in the presence of geometric uncertainty. Examples of complex tasks include sensor-based navigation and 3D manipulation in partially or completely unknown environments, using redundant robotic systems such as mobile manipulator arms, cooperating robots, robotic hands or humanoid robots, and using multiple sensors such as vision, force, torque, tactile and distance sensors.

Within the context of this thesis, the traditional *robot-centric* control paradigm will have to be replaced by an *action-centric* paradigm, e.g., (Ambler and Popplestone 1975; Khatib 1987; Samson et al. 1991; De Schutter et al. 2007).

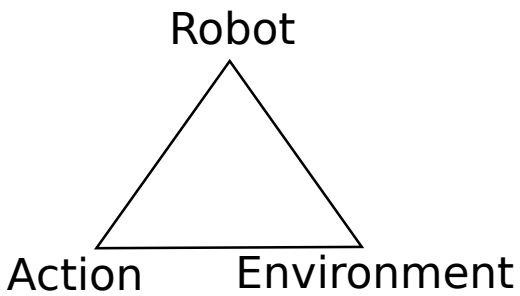

Figure 1.1: In every robotic application there is an action to be executed by a robot in a given environment.

Approaches for motion specification

Every robotic application tries to impose a desired action that has to be executed by a robotic system in a given environment. This results in an area of tension between the *robot*, the *action* and the *environment*, see Figure 1.1.

This area of tension is reflected in the different approaches for the specification of the desired motion in robotic applications. The classification presented below does not induce rigid boundaries in the area of tension between robot, action and environment; usually a mixture of all three is used to specify a robotic application. Hence, a well-designed motion specification methodology supports the three types of motion specification presented below.

(i) **Robot-centric motion specification.**
Recent industrial robot programming languages such as Kuka's Kuka Robot Language, ABB's RAPID, or Fanuc's KAREL are all based on the robot-centric motion specification, which are typical of the robot programming languages developed in the 60's, 70's, and early 80's, like (Ernst 1962; Paul 1977; Silver 1973; Mujtaba and Goldman 1981; Taylor et al. 1982; Shimano 1979; Takase et al. 1981; Ruoff 1979). The common concept in all those languages is that the motion of the *robot*, usually the robot end effector or an attached tool, is specified. When the task, or more importantly the environment changes, the program has to be redefined.

(ii) **Object-centric motion specification.**
Motion specification at the level of the objects that are involved in the action was the subject of robot programming research in the 70's and 80's. Spatial relations between objects are used to specify the desired goal state of the objects. In some cases even the uncertainty in the

geometric model of the environment is included in the specification. These spatial relations are independent from the robotic system, therefore object-centric motion specification can be reused for different robotic systems, and in different environment-robot settings. For instance, if a peg-in-hole task is specified using an object-centric motion specification, this specification is independent from the manipulated object, i.e. the specification does not change whether the peg, the hole or even both objects are manipulated. This kind of specification is limited since (i) the desired goal state of the objects cannot always be specified completely because of the existing uncertainty in the geometric model of the environment, and (ii) the spatial relations are usually specified using position representations, which can lead to overspecifying the objects desired goal state.

(iii) **Action-centric motion specification.**
Another approach is to specify the sequence of actions instead of the sequence of desired object states. Thus instead of building a geometric model of the objects in their desired goal state, the operations through which this state can be achieved are described. The actions include a goal statement involving spatial relationships between objects. The desired object goal state does not specify actual positions, instead it indicates the physical relationships between the objects that should be achieved by the action. This motion specification approach does not have the tendency to overspecify the desired goal state of the objects.

iTaSC: instantaneous Task Specification using Constraints

iTaSC is a motion specification framework introduced by De Schutter et al. 2007, which is primarily based on the work of Ambler and Popplestone 1975. It presents a methodology for specifying constraints on spatial relationships between objects and their features, not only in the desired goal state but at all instants during the execution of a motion. These features are usually physical properties of their corresponding objects, such as the bottom of a box or the symmetry axis of a cylinder. Furthermore, the methodology enables the explicit introduction of geometric uncertainties. From this specification, an instantaneous motion control law is automatically derived to calculate the robot motion control input, which can be joint torque, acceleration or velocity, from the desired constraint input. The methodology is especially designed for the action-centric motion specification approach but can be used for all three of the presented approaches for motion specification. The presented methodology allows us to specify the desired motion specification as a constrained optimization problem. Since most of the work presented in this

thesis is based on the iTaSC methodology, Chapter 3 will explain in more detail the iTaSC methodology.

1.2.2 Motion coordination

A typical robot *task* consists of a sequence of motions, each with its own formal specification of the desired instantaneous motion of the robot, its own motion controller and its own online parameter estimators as described in Chapter 3. The configuration of the motion specification in terms of the desired motion and parameters of the motion controllers and the coordination of different configurations based on measurements and the actual motion is called a *skill*. Since the term *skill* is given different meanings in literature, this text will use the following definition for *skill*.

Definition 1 (Skill) A skill is the component in a robot control system that is responsible for the *coordinated execution* and *parameter configuration* of the set of available *instantaneous robot motion controllers*, such that, together, multiple motion controllers let the robot system realize a certain *task* between a set of modeled objects in the *environment*.

For example pointing a camera to an object in space is a task that is locally defined between the camera and the object. The configuration of the desired motion between the camera and the object, the monitoring of the actual motion, and the coordination of the resulting motion is what we call a skill.

The reason why the skill is so important in all robotic domains is the increasing complexity of the coordination of a robotic system. This increasing complexity is due to the fact that robotic tasks involve more sensors and more uncertainty every day, and programmers want their robots to use more and more knowledge online, to guide the sensor processing towards real scene and task *understanding*. In addition, interaction with the human(s) in the environment becomes mandatory: the robot should "understand" what the human is doing, and why he is doing it, and vice versa.

Coordination and configuration of desired motion is found in all robotic domains, from industrial robotics, over behavior-based robotics to multi-agent robot systems. The introduction of our definition of a skill can be seen as a *unification* of all configuration and coordination methodologies that are introduced in literature. Therefore we can rightfully call the skill a *best practice in robotics*.

A large part of the recent research about skills is focused on learning approaches, where the robot control system has to learn a skill or motion, as fast and as good as possible from a human demonstration (Schaal 1999; Billard and

Siegwart 2004; Schraft and Meyer 2006; Argall et al. 2009). All these learning approaches boil down to learning motion configuration parameters and motion coordination strategies. Almost all existing learning methodologies are however based on the full 6D motion specification of objects, which in most cases overspecifies the motion specification. Chapter 5 will present a coordination methodology based on the constraint-based motion specification presented in Chapter 3, which should enable learning of only those aspects of the motion that are important for the execution of a task between two objects.

1.2.3 Software support

Obtaining an actual execution on a robotic system of a given task from its motion specification is not always straightforward, and usually the implementation is robot system or even task specific. Software support is needed to convert the mathematical task description into the control-platform-specific control signals.

Ideally, generic robot programming systems or generic programming tools are used from the mathematical description of the task down to the execution of the task on a specific robotic platform. Such a generic robot programming system does not exist, and the search for such a system is still ongoing. The problem with proposals for new robot programming systems is that they not always address the key requirements of a *robot* programming system.

Lozano-Pérez 1983 describes the key requirements for a robot programming system in the areas of sensing, world modeling, motion specification, flow control and programming support. He also gives a representative overview of the existing systems at that time and how they meet these requirements. He describes three different categories of programming systems:

(i) *guiding systems* in which the user has to define the entire motion of the robot in terms of joint positions. These systems do not have the capability of incorporating external sensors. The entire motion is specified as a single execution sequence without loops, conditionals or computations. Guiding systems are the most used programming system in industry, some example applications are spot welding, painting and simple material handling.

(ii) *robot-level programming systems* enable the usage of external sensors, such as vision and force. Using this extra information, robot applications can cope with a greater degree of uncertainty in the environment. The key drawback of these systems is that they require the robot programmer to be an expert in computer programming and in the design of sensor-based

motion strategies. Hence, these systems are unsuitable for the typical worker on the factory floor.

(iii) *task-level programming* requires specifying goals for the positions of objects, rather than the motions of the robot needed to achieve those goals. This approach however requires the geometric models of the environment and the robot.

It is the latter category that interests us the most, since it allows us to create modular building blocks that are independent from the robot system and can be reused to create complex robot applications.

A very important issue raised by Lozano-Pérez 1983 is that the range of computer knowledge of robot users is large, from factory personnel with no knowledge about robotics and programming, to Ph.D's in computer science. Reducing the basic functionality of robot programming systems to the least sophisticated user is a fatal mistake. He explicitly argues that:

> *robot programming languages should support the functional requirements of its most sophisticated users and allow sophisticated users to implement special-purpose interfaces,* in the language itself*, for the less experienced users.*

Lozano-Pérez 1983 also discusses the requirements for robot programming *languages*. All of these requirements are still important and most of them are still not met in modern robot programming languages available for industrial robots. Therefore they are listed here again:

(i) sensing: *a vast majority of current industrial robot applications are performed using position control alone without significant external sensing. Instead, the environment is engineered to eliminate all significant sources of uncertainty. Sensing enables robots to perform tasks in the presence of significant environmental uncertainties without special-purpose tooling.* There is a need to specify parameters for sensor-based motions and to specify alternate actions based on sensory conditions. *This is the primary motivation for using sophisticated robot programming languages.*

(ii) world modeling: *When the environment is not known a priori some mechanism must be provided for representing the positions of objects and their features and the uncertainty of these positions.*

(iii) motion specification: Specify the motion by a set of constraints between features of the robot and features of objects in the environment. *The execution system can then choose the "best" motion that satisfies these*

constraints by supplying a desired optimization criterion. This general capability was beyond the state of the art in trajectory planning and still is in some form. The programming system should either provide a well-defined semantics for treating inconsistent constraints or make it impossible to specify inconsistent constraints.

(iv) flow of control: *In general, the program for a sensor-based robot must choose among alternative actions on the basis of its internal model of the task and the data from its sensors. In each state, the task program must specify the appropriate actions for the robot.* The programming system must provide the capabilities for making these control decisions. *Robot programs often must interact with external processes which are being executed in parallel and asynchronously. Coordination between cooperating and competing asynchronous processes must be possible.*

(v) programming support: *A sophisticated robot programming system must first be a sophisticated programming system. Robot program development is often ignored in the design of robot systems, and consequently, complex robot programs can be very difficult to debug. That's why programming systems should allow programs to be modified on-line and immediately restarted. Also the ability to record sensor output and program traces, together with visualizations play an important role in debugging. Most existing robot systems are stand-alone. Today this makes little sense. A robot should support a high-speed command interface to other computers.*

Lozano-Pérez 1983 ends with the hope that commercial robots of the future will be designed with a view towards interfacing to other computers, rather than as a stand-alone systems. Today there is still no decent interoperability possible between commercial robots, even from the same manufacturer.

The issues raised by Lozano-Pérez 1983 are still very important, and by far still not addressed by the current industrial robot programming systems. Even current experimental robot programming systems do not address all of the issues.

1.3 Contributions

This thesis contains two types of contributions. First, this thesis presents methodologies and algorithms that enable the design of modular and reusable skills which can be used on the new generation of robots. The thesis also contains various software tools to create practical applications on real robots using the presented methodologies and enabling the maximal reuse and distribution of robot skills.

1.3.1 Extension of constraint-based motion specification

Chapter 3 presents and illustrates the constraint-based motion specification of De Schutter et al. 2007 and Rutgeerts 2007. The work presented by De Schutter et al. 2007 only allows geometric constraints in the robot space: constraints can only be defined on the robot joint coordinates or on the auxiliary coordinates from modeled spatial relations. Therefore, Chapter 4 also explains how to include constraints between two objects from any kind of feature space, as long as an *interaction model* for the output can be found. As an example, image-based visual servoing (IBVS) (Espiau et al. 1992) is reformulated in the iTaSC framework in order to integrate it seamlessly with other task constraints, in image space, in Cartesian space, in the joint space of the robot, or in the "image space" of any other sensor (e.g. force, distance).

1.3.2 Coordination of constraint-based motion specification

The constraint-based motion specification from Chapter 3 only allows us to specify *instantaneous* motion control and estimation; no mechanism to control the overall *non-instantaneous* motion is provided. Chapter 5 explains how to specify a skill based on the constraint-based motion specification methodology and how this motion specification can be coordinated and configured. Chapter 5 also shows how the use of iTaSC results in highly modular and reusable robot skills.

1.3.3 A recursive acceleration-based constraint solver for branching kinematic structures.

Solving the control input for humanoids and other highly redundant systems from motion specification is still a computationally intensive procedure. A recursive acceleration-based constraint solver based on Gauss's principle of least constraint (Vereshchagin 1989) was proposed in the late eighties, but never got the attention it deserved. Chapter 6 explains in detail how this solver works, and extends it for branching kinematic structures to make it useful for humanoids and other highly redundant robotic systems. Chapter 6 also explains how this solver can be used in combination with the constraint-based motion specification.

1.3.4 Formal representation for constraint-based motion specifications

To allow maximal reuse of motion specifications and the skills that are built on top of them, the formal representation of a constraint-based motion specification is important. Chapter 8 explains how a constraint-based motion specification can be formalized and represented in an existing standard file format that is commonly used in the 3D interactive industry. This formal representation will allow us to easily share motion specifications and make motion specification easily interchangeable between various implementations and control systems.

1.3.5 A software framework for solving constraint-based motion specification

Chapter 7 presents a software framework that addresses all of the problems stated in Section 1.2.3. It uses the constraint-based motion specification methodology to address the issues in motion specification, sensing and world modeling. The skills from Chapter 5 are used to address the issues in flow of control. The actual implementation is based on the OROCOS project (Bruyninckx 2001), which is a very powerful programming framework for the execution of hard realtime control tasks, and which allows the programmer to define application-specific interfaces for less experienced users.

1.3.6 A software library for kinematic and dynamic algorithms of kinematic structures.

The software framework presented in Chapter 7 as well as many other robotic control software frameworks needs powerful kinematic and dynamic algorithms that can be reused for different models of kinematic structures. Chapter 9 presents a software library that offers a generic implementation of various kinematic and dynamic algorithms. For each of these implementations special care is taken (i) to use a generic API, so that using different algorithm implementations for the same purpose only changes the configuration procedure but not the usage of the algorithm in the running state of a robot control system; and (ii) to make all algorithm implementations safe to use in a hard realtime control setting.

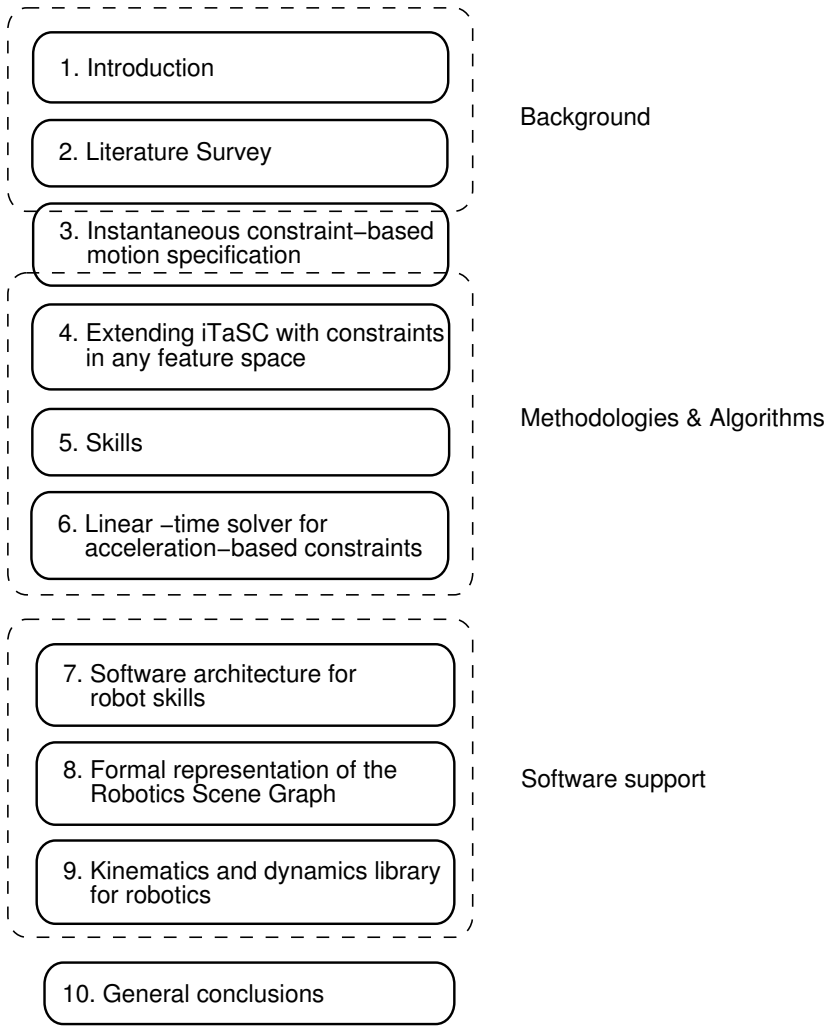

Figure 1.2: Graphical overview of the different chapters of the thesis, grouped by theme.

1.4 Overview of the thesis

Figure 1.2 gives a graphical overview of the different chapters of the thesis. Like the contributions, the chapters can be grouped in "Methodologies & Algorithms" and "Software Support".

The main objective of this thesis, introduced in Chapter 1, is the design of highly modular and reusable tasks for robots. These tasks are what we call skills and are presented in Chapter 5. Different approaches for motion specification and motion coordination are introduced in Chapter 2. Chapter 3 introduces the underlying instantaneous constraint-based motion specification framework and makes the transition from the background of this thesis to the developed methodologies and algorithms. Chapter 4 extends this motion specification framework to include constraints from any feature space. A specific solver for acceleration-based motion specification, which is particulary of interest for highly redundant robots, is presented in Chapter 6.

A software architecture that inherits the modularity and reusability of the presented methodologies, is presented in Chapter 7. To increase the reusability and distribution of the tasks, Chapter 8 presents a proposal for a standardized motion specification file format. Chapter 9 presents a software library that allows realtime execution of kinematics and dynamics algorithms, needed in robotic systems.

Chapter 2

Literature survey

I almost wish I hadn't gone down that rabbit-hole, and yet, and yet–it's rather curious, you know, this sort of life.

Lewis Carroll (Alice in Wonderland)

This chapter provides a short overview of the research and different approaches for robot motion specification. Research in the domain of robot motion specification was very active in the seventies and eighties but mostly ended in the early nineties after the introduction of the Operational Space (Khatib 1987) and the Task Function Approach (Samson et al. 1991). De Schutter et al. 2007 obtain the most recent result of the research in the robot motion specification domain and is the base for this thesis. A short history about robot motion coordination introduces the most common approaches for coordination in the different robotic domains without giving an elaborate overview.

2.1 Robot motion specification

Task specification for robot applications became very popular in the beginning of the 1970's when industry/researchers started to use robots for automated mechanical assembly. The main difficulty was how to introduce external systems to cope with uncertainties due to the robot manipulation accuracy, object positioning and environment modeling.

Feldman et al. 1969 and Ejiri et al. 1971 are a few of the first attempts to include a camera as an external sensor in an automated mechanical assembly system. Both systems focus on automatic object recognition, which is necessary to determine the appropriate manipulation movements.

Bolles and Paul 1973 present one of the first successful attempts to incorporate a camera system, force feedback and touch sensor to assemble a water pump for a car. As future work they state that more work has to be done on the automatic specification of position and *motion constraints* and on a more powerful control language.

Ambler et al. 1975 describe a system that is able to "learn" to recognize new objects based on a camera system. The assembly motion programming however was still quite tedious, involving a lot of numerical parameters. As future work they point out the replacement of these parameters by instructions based on geometric relations.

Task-level motion specification has already been studied for a long time. The goal of this research is to specify the desired robot motion in terms of *feature frames* of the objects to be manipulated instead of manipulator coordinates. These feature frames indicate *physical entities* on the objects, such as vertices, edges, faces, holes, etc.

Ambler and Popplestone 1975 present the first attempt towards a more object-oriented task specification. They provide a "much less painful" method to specify the robot motion through symbolic spatial relations that are to be established between the features of objects during the different steps in an assembly process. Quoting Ambler and Popplestone 1975:

> *The goal is to infer the desired relative pose of these objects from the specified geometric relations between the features. In turn this desired relative pose is used to determine the goal pose of a robot who has to assemble both objects.*

Popplestone et al. 1978 describe an implementation of a system that transforms the symbolic specification of geometric goals together with a program that specifies the directions of the motions into a sequence of end-effector positions.

The emphasis of this system is primarily on the task specification; it does not cover sensor feedback.

Taylor 1976 also introduces constraints in terms of goal positions on the *free* variables aside the constraints introduced by contact. For all constraints a mathematical model is constructed. The *contact* and *free* constraints are merged together and reduced. Afterwards, a linear programming technique is used to solve for the intermediate manipulator motions. Taylor 1976 introduces a methodology to make planning decisions by manipulating constraints on the positions of objects which explicitly model their error. Brooks 1982 significantly extends the work of Taylor 1976. The principal extension was the use of symbolic constraints not only forward to get error bounds but also backward to restrict the choices on plan variables. A second extension was the introduction of appropriate sensing into the program, if the error bounds due to the uncertainty are too big, a sensing action, which reduces this uncertainty, is performed.

The Task Frame Formalism: Mason 1981 introduces the *Task Frame* formalism, which is now widely used for specifying hybrid force position/velocity control operations. This formalism was first successfully implemented by Raibert and Craig 1981 using a torque resolved control scheme. In this formalism the force and velocity controlled directions coincide with the axes of a single frame, the *Task Frame* or *Compliance Frame*. This work however neglects the robot dynamics and does not track how the *Task Frame* evolves during the motion execution. De Schutter and Van Brussel 1988a and De Schutter and Van Brussel 1988b introduce a general way to track the *Task Frame*. They present a methodology to update the position of the *Task Frame* based on the observation of the natural constraints. This allows easy specification of for instance force-controlled contour following. Others like Yoshikawa and Sudou 1993 use ad hoc solutions. Bruyninckx and De Schutter 1996 present an overview of *Task Frame* based motion specifications for different setups.

The Operational Space Approach: Khatib 1987 introduces the dynamic model of the robot into the task frame formalism. He also generalizes the formalism to the *Operational Space*. Khatib 1987 however does not offer a general way to construct such an *Operational Space*. De Sapio et al. 2006 extend this work for constrained multi-body systems and highlight the resemblance between the natural constraints and the *Operational Space* motion specification.

The Task Function Approach: Samson et al. 1991 introduce the *Task Function* concept. A task function is defined as an output function which

depends on the robot joint state and time. The task function can take any form, although Samson et al. 1991 give various examples, they do not give any hints on how to create this task function in a general way. Samson et al. 1991 also explain how to explicitly include the sensor variables into the task function to model the interaction with the environment. They state:

> *we try to derive simple tools which facilitate the design of sensor-based tasks: location of sensors, choice of the task function and models of the interaction screws. A natural method consists of proposing a set of basic 'canonical' structures corresponding to the most usual tasks, which may be combined and adapted to each specific application.*

By applying constraints on the sensor measurements in the task function they create a *virtual linkage* between the sensor and the measured object, which resembles the linkage created by contact.

Bruyninckx et al. 1995 introduce a *virtual contact manipulator* to link the manipulated object with the environment using models based on the first and second order geometric parameters of the contact surfaces. The motion constraints are described by the *loop closure equations* for the virtual contact manipulators for every contact between the manipulated object and the environment. Motion specification is done by choosing the right joint velocities of the free joints in the virtual contact manipulators. For geometric uncertainty estimation, *uncertainty joints* are introduced in the virtual contact manipulators. Identification of these uncertainties boils down to closing the gaps in the loops, introduced by geometric uncertainties in the model.

The methodology introduced by De Schutter et al. 2007, which was explained in more detail by Rutgeerts 2007, extends this work to use generic 6D *virtual manipulators* to describe the degrees of freedom between two objects. This approach is based on Ambler and Popplestone 1975. De Schutter et al. 2007 explain how to create the virtual manipulator for the spatial relations between the features of objects. The task specification is done by specifying constraints on the joints of the virtual manipulator. De Schutter et al. 2007 also explain how the robot joint commands can be *instantaneously* derived from this task specification for a velocity-resolved control scheme. As in Bruyninckx et al. 1995, geometric uncertainties are modeled as *uncertainty joints* in the virtual manipulators. De Schutter et al. 2007's main contributions are:

- the systematic approach to specify the instantaneous motions, involving multiple task constraints, multiple robots and multiple sensors,
- the integration of both task specification and estimation of uncertain geometric parameters in a unified framework,

- the ability to specify constraints not only in one frame but in arbitrary frames, and
- the ability to automatically derive controllers and estimators from the specification.

The methodology introduced by De Schutter et al. 2007 is referred to as iTaSC, instantaneous Task Specification using Constraints. Other applications of iTaSC were presented by De Laet et al. 2007; Smits et al. 2008; Smits et al. 2008; Decré et al. 2007. Other control schemes are introduced by De Laet and De Schutter 2007.

2.2 Robot motion coordination

Task skeletons are the first primitives that come close to the concept of robot motion *coordination*. Task skeletons were primarily used at the planning level and most of the time they were only sequential, i.e. once the execution on the robot started, all robot actions take place in a fixed sequential order. Lozano-Pérez 1976 describes how to automatically transform an assembly plan into a manipulator program. Lozano-Pérez 1976 uses goal position and orientation constraints for (i) the part description itself, (ii) the description of objects before and after each assembly operation, (iii) the indication of the prerequisites of assembly operations, and (iv) to describe the result of the operations in the assembly strategies. The assembly strategies are formulated as *skeleton programs*, they specify the motions of objects relative to local coordinate systems and are therefore independent of how the objects are held, which of the objects is manipulated, and the manipulator. The planner determines the parameters for the skeleton programs from the knowledge of the environment present in the system's world model. He also extends the constraint definitions of Ambler and Popplestone 1975 to include appropriate ranges of the relative position and orientation between the objects where the constraints between the objects are valid.

Taylor 1976 introduces prototypical motion strategies for particular tasks as parameterized robot programs, known as *procedure skeletons*. A skeleton has all the motions, error tests, and computations needed to carry out a task, but many of the parameters needed to specify motions and tests remain to be specified.

Lozano-Pérez and Brooks 1985 introduce an architecture for a new task-level system, TWAIN. This architecture is based on a two-level approach. It starts by transforming the task-level description into a skeleton program. This skeleton program introduces different variables for input, output and errors. In a

second step the constraints, errors and uncertainties on these variables are propagated forward and backward (Brooks 1982). If the uncertainty is too high to successfully execute the program, a sensing action is executed to reduce this uncertainty. Instead of using a sequence of model states as a task specification, TWAIN uses a sequence of operations, describing the operations through which the desired state can be achieved. These operations should still be object-oriented, and contain the goal spatial relationship between the manipulated objects. The interest in the methodology of propagating the uncertainty and error back and forward to come up with a successful plan decreased since the observation of small variations can lead to drastic changes of the plan leading to an exploding problem complexity.

Coordination in industrial robot applications:

Saridis and Stephanou 1977 already distinguished different control levels for controlling a prosthetic arm. They call them *organization*, *coordination*, and *direct control*. The skill-like *coordination* level is responsible for improving the overall dynamic performance of the system by changing some control law parameters based on a minimization of the overall energy needed to perform a motion.

McCarragher et al. 1997 present a skill-like hybrid dynamic framework for the monitoring and control of robotic manipulation in a mechanical assembly setting. Their work is based on discrete events and process monitoring. They combine rule-based logic with Hidden Markov models that lets the controller decide autonomously how the robotic task has to evolve. In this work the coordination is based on stochastic data. Despite the fact their framework can deal very well with uncertainty in the environment model, it lacks support for the *estimation* of the uncertain parameters, it does not include the support for gradual controller state changes, and it is primarily focused on assembly tasks.

Finkemeyer et al. 2005 and Finkemeyer 2004 present a skill-like hybrid control architecture that can switch between different controllers and that can handle real sensors with finite measurement ranges in a stable and safe way, based on an adaptive selection matrix. The authors use this architecture for executing assembly tasks. They tackle the problem of uncertainty in the environment through extensive robust action coordination, but the architecture lacks support for the estimation of the uncertain parameters and for decisions based on stochastic data.

Coordination in behavior-based robot applications

Brooks 1986 introduces the skill-like subsumption architecture. Instead of dividing the overall robotic application in series of functional blocks, he proposes to divide a robotic application into *task-achieving modules*. Each of these modules is a finite state machine. Every module adds a new layer of competence to the already existing modules.

Mataric 1997 explains the differences between skill-like behavior-based control and purely reactive control. Reactive control uses no internal models or internal state and relies on a direct coupling between sensors and actions. Behavior-based control relies on the subsumption architecture (Brooks 1986) and includes an internal representation and computation in order to *decide* what action to take. The action selection or arbitration level (Maes 1989; Brooks 1991) is usually built in the behavior design without an explicit control level. Since it is very hard to prove a working system without explicit coordination, most behavior-based control systems follow a hybrid approach were the behaviors are controlled by a *supervisor*.

Coordination in multi-agent robot applications

Multi-agent systems can be controlled in different ways: (i) centralized, by a special coordination agent; (ii) distributed, through negotiation between the different agents; and (iii) decentralized, where every agent decides on its own how to fulfill its individual goals. Friedrich et al. 1998 propose a special *manager* agent to activate all other agents. Every agent is responsible for a specific *skill* which use *elementary operations*(EOs) as a representation. These EOs provide the interface needed for task negotiation or action selection by the manager. Joyeux et al. 2009 also propose a *manager* that makes sure the overall plan is coordinated using the different agents in the system. Laengle et al. 1995 also propose a *special agent* to coordinate the actions of different agents that need to be synchronized. Kaiser et al. 1996 present a framework to *learn* a *coordination skill*. In this work the coordination of other basic skills, like sensing, navigation and manipulation, is learned from a human demonstration.

2.3 Conclusion

Although task-oriented motion specification based on constraint specifications has been a research topic for a long time, it has only extensively being used in the planning phase of the robot program, see (Latombe 1991; Latombe 1999; LaValle 2006) for a review of planning algorithms. The operational space

specification (Khatib 1987) is one of the most popular methodologies for task-oriented motion specification, but until the iTaSC methodology (De Schutter et al. 2007) no generic way for constructing an operational space existed.

Coordination of motion is addressed in many robotic domains. All of them use some form of state machine logic for the coordination. The coordination can be applied to simple reactive controllers, Task Frame Formalism controllers or Operational Space controllers. The drawback of these controllers is that they over-specify the desired motion, which decreases the composability of the motion controllers. Hence, the coordination methodologies built on top of these controllers will also not be optimized for composability of different coordination strategies.

Chapter 3

Instantaneous constraint-based motion specification

Never mistake motion for action

Ernest Hemingway (1899 – 1961)

A task specification methodology based on the specification of constraints between objects and their features was developed by De Schutter et al. 2007 and described in more detail by Rutgeerts 2007. This chapter will explain in detail how the methodology works and illustrates the methodology with a force-controlled manipulation task executed by two robots that are aware of humans moving in their environment.

3.1 Example application: Human-aware force-controlled manipulation task using two robots

In this application two robots are used to:

- track the contour of an unknown 2.5D object,
- while keeping a safe distance from, and
- pointing a camera towards the person closest to the robots' workspace.

Figure 3.1 gives an overview of the actual experimental setup. Figure 3.2 shows a detail of the sensors that are used in the application.

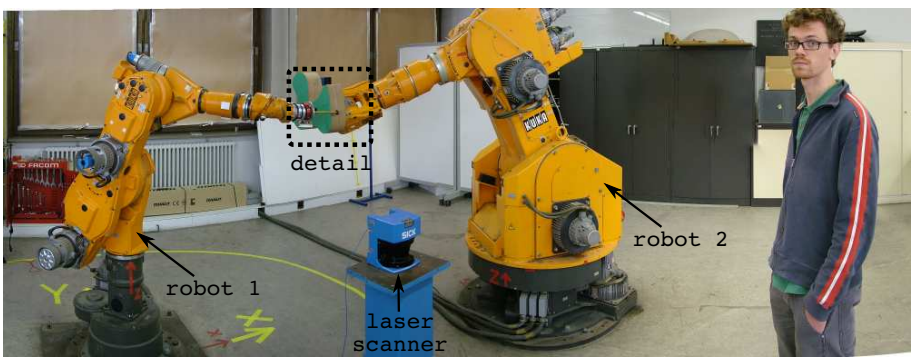

Figure 3.1: Overview of the experimental setup for the human-aware force-controlled manipulation task showing both robots a human and the laser scanner

In every application multiple *object frames*, these are frames attached to the objects of interest, are connected to each other in a *scene graph*. Hence, the scene graph contains the information on which object frames are connected to each other and how they are connected to each other. The scene graph of this application is shown in Figure 3.3 and contains the following object frames and models:

- Robot 1 with its base frame $\{r1\}$ attached to the world reference frame $\{w\}$ with a fixed offset pose $\boldsymbol{T}_{w}^{r1}$. The model of Robot 1 only defines one object frame, its end effector frame $\{e1\}$. The pose of the end effector is represented by $\boldsymbol{T}_{r1}^{e1}(\boldsymbol{q}_1)$, in terms of its joint coordinates $\boldsymbol{q}_1$;

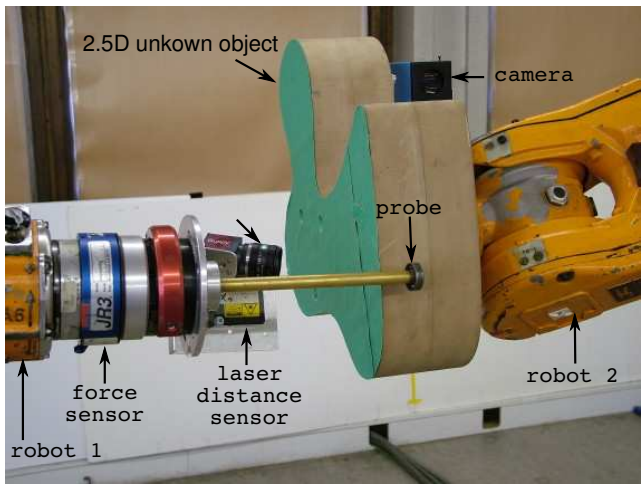

Figure 3.2: Detail of the experimental setup for the human-aware force-controlled manipulation task. The detail shows all the sensors that are used in the application, the force sensor and the laser distance sensor and the camera that is used to point to the closest human in the environment.

- Robot 2 with its base frame $\{r2\}$ attached to the world reference frame $\{w\}$ with a fixed offset pose $\boldsymbol{T}_{w}^{r2}$. The model of Robot 2 only defines one end effector frame $\{e2\}$. The pose of the end effector is represented by $\boldsymbol{T}_{r2}^{e2}(\boldsymbol{q}_2)$, in terms of its joint coordinates $\boldsymbol{q}_2$;
- A 6D force sensor with its object frame $\{force\}$ attached to the object frame $\{e1\}$ with a fixed offset pose $\boldsymbol{T}_{e1}^{force}$;
- A 1D laser distance sensor with its object frame $\{laser\}$ attached to the object frame $\{e1\}$ with a fixed offset pose $\boldsymbol{T}_{e1}^{laser}$;
- A probe with its object frame $\{probe\}$ defined on its tip, attached to the object frame $\{force\}$ with a fixed offset pose $\boldsymbol{T}_{force}^{probe}$;
- A camera with its object frame $\{camera\}$ attached to the object frame $\{e2\}$ with a fixed offset pose $\boldsymbol{T}_{e2}^{camera}$;
- An unknown 2.5D object (it is an unknown 2D shape extruded over an unknown distance) with its object frame $\{object\}$ attached to the object frame $\{e2\}$ with a fixed offset pose $\boldsymbol{T}_{e2}^{object}$, and a partially unknown object frame $\{front\}$ with a partially unknown offset pose $\boldsymbol{T}_{object}^{front}(\boldsymbol{\chi_u}_{front})$. The modeled uncertainty contains the extrusion distance of the unknown 2D shape, h_u and the orientation of the front-plane of the object, ϕ_u and ψ_u;

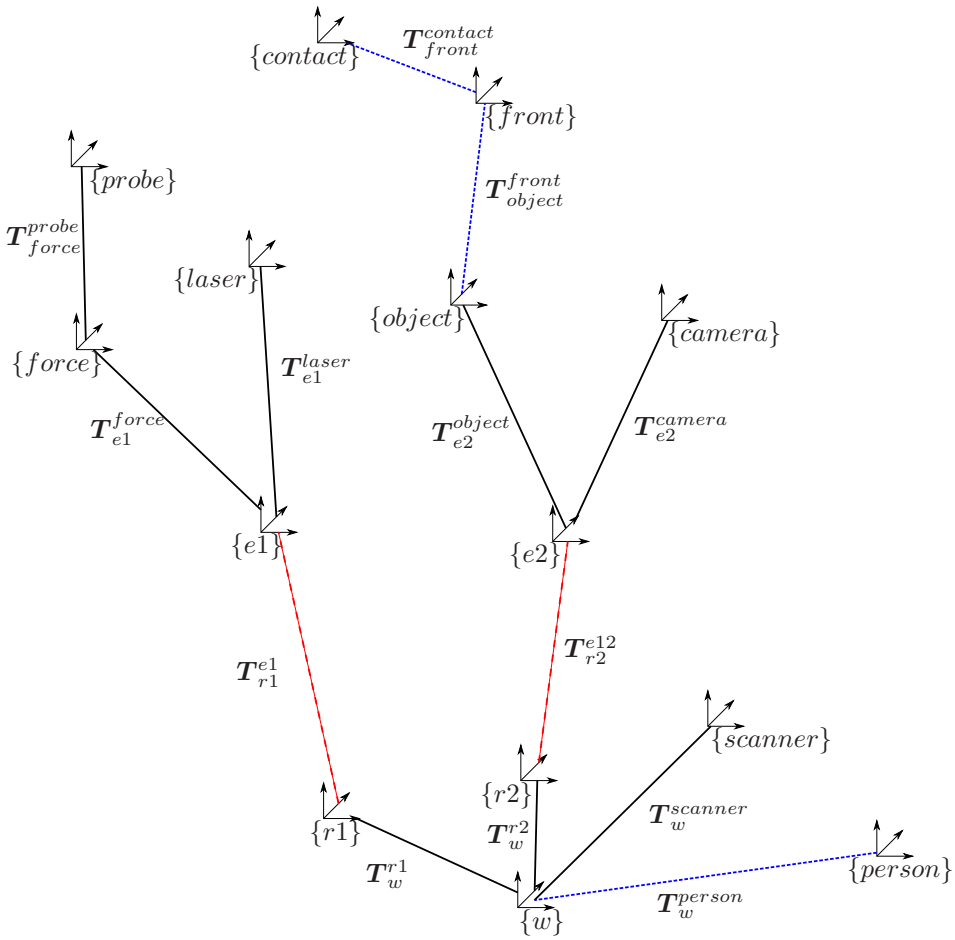

Figure 3.3: The "scene graph" of the example application: both robots, the probe, all the sensors and objects are visualized and how they are connected to each other. Black solid lines are fixed transformations, blue dotted lines are partially uncertain transformations and red dashed lines are transformations controlled by the robotic manipulators control input.

- The contact point of the probe on the object, represented by the feature frame $\{contact\}$ attached to the object frame $\{front\}$ by a partially unknown offset pose $\boldsymbol{T}^{contact}_{front}(\boldsymbol{\chi}_{\boldsymbol{u}contact})$. The modeled uncertainty contains the unknown 2D shape information, which are the tangential and normal direction of the contour;
- The 2D laser range scanner represented by the object frame $\{scanner\}$

which is attached to the environment object frame $\{w\}$ with a fixed offset pose $\boldsymbol{T}_w^{scanner}$;

- The closest person in the environment represented by the object frame $\{person\}$ attached to the environment object frame $\{w\}$ with a partially unknown offset pose $\boldsymbol{T}_w^{person}(\boldsymbol{\chi_{uperson}})$. The modeled uncertainty contains the 2D location of the person in the $X-Y$-plane of the world reference frame.

3.2 Modeling spatial relations between objects

In a first step the iTaSC methodology models different relationships that introduce auxiliary coordinates that will facilitate the specification of the desired motion and the estimation procedure.

3.2.1 Introducing auxiliary coordinates

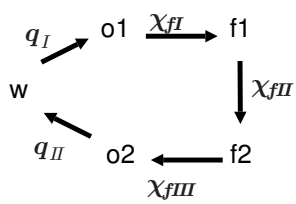

Figure 3.4: Resulting virtual kinematic loop created by a spatial relation between object and feature frames.

A spatial relationship connects the object frames $\{o1\}$ and $\{o2\}$, through their feature frames $\{f1\}$ and $\{f2\}$ as is shown in Figure 3.4. Every spatial relationship between two objects contains six independent degrees of freedom (DOF), which are characterized by six independent auxiliary coordinates, $\boldsymbol{\chi_f}$. These coordinates are divided into $\boldsymbol{\chi_{fI}}$, the coordinates which model the motion of $\{f1\}$ with respect to $\{o1\}$, $\boldsymbol{\chi_{fII}}$, the coordinates which model the motion between $\{f1\}$ and $\{f1\}$, and $\boldsymbol{\chi_{fIII}}$, the coordinates which model the motion of $\{f2\}$ with respect to $\{o2\}$. The spatial relation creates a virtual kinematic loop between two objects which, in the general case, both can be connected to a robotic manipulator. The resulting kinematic position loop l, includes both the robotic manipulator and the spatial relation and hence depends on the robot

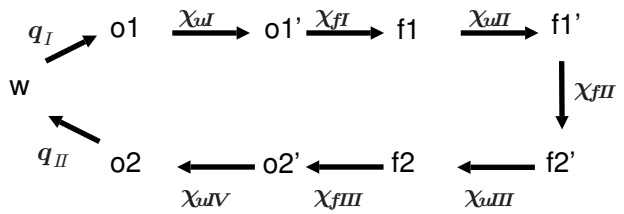

Figure 3.5: The resulting graph for a modeled virtual kinematic loop between objects $\{o1\}$ and $\{o2\}$, including the robot manipulator coordinates, $\boldsymbol{q}$, the spatial relation coordinates, $\boldsymbol{\chi_f}$ and the modeled uncertainty coordinates $\boldsymbol{\chi_u}$.

coordinates $\boldsymbol{q} = [\boldsymbol{q}_I, \boldsymbol{q}_{II}]^T$ and the auxiliary coordinates $\boldsymbol{\chi_f} = [\boldsymbol{\chi_{fI}}, \boldsymbol{\chi_{fII}}, \boldsymbol{\chi_{fIII}}]^T$:

$$l(\boldsymbol{q}, \boldsymbol{\chi_f}) = \boldsymbol{0}, \tag{3.1}$$

$$\boldsymbol{T}^w_{o1}(\boldsymbol{q}_I)\boldsymbol{T}^{o1}_{f1}(\boldsymbol{\chi_{fI}})\boldsymbol{T}^{f1}_{f2}(\boldsymbol{\chi_{fII}})\boldsymbol{T}^{f2}_{o2}(\boldsymbol{\chi_{fIII}})\boldsymbol{T}^{o2}_{w}(\boldsymbol{q}_{II}) = \boldsymbol{I}, \tag{3.2}$$

$$\boldsymbol{J_f}\dot{\boldsymbol{\chi}}_{\boldsymbol{f}} + \boldsymbol{J_q}\dot{\boldsymbol{q}} = \boldsymbol{0}, \text{ and} \tag{3.3}$$

$$\boldsymbol{J_f}\ddot{\boldsymbol{\chi}}_{\boldsymbol{f}} + \dot{\boldsymbol{J}}_{\boldsymbol{f}}\dot{\boldsymbol{\chi}}_{\boldsymbol{f}} + \boldsymbol{J_q}\ddot{\boldsymbol{q}} + \dot{\boldsymbol{J}}_{\boldsymbol{q}}\dot{\boldsymbol{q}} = \boldsymbol{0}. \tag{3.4}$$

Equation (3.1) is the analytic form of the resulting virtual kinematic loop which is represented in Eq. (3.2) using homogeneous transformation matrices. Eqs. (3.3) and (3.4) represent the first and second derivative, respectively, of Eq. (3.1) with the *feature jacobian* $\boldsymbol{J_f} = \frac{\partial l}{\partial \boldsymbol{\chi_f}}$, the *robot jacobian* $\boldsymbol{J_q} = \frac{\partial l}{\partial \boldsymbol{q}}$, and their derivatives $\dot{\boldsymbol{J}}_{\boldsymbol{f}} = \frac{d\boldsymbol{J_f}}{dt}$ and $\dot{\boldsymbol{J}}_{\boldsymbol{q}} = \frac{d\boldsymbol{J_q}}{dt}$.

3.2.2 Introducing geometric uncertainties

Any geometric uncertainty in the spatial relationship is modeled by a corresponding uncertainty coordinate $\chi_{\boldsymbol{u}}$. The uncertainty coordinates connect the nominally modeled frames $\{t\}$ with the estimated frames $\{t'\}$. Figure 3.5 shows the resulting graph for the virtual kinematic loop between two objects, which includes the uncertainty coordinates $\boldsymbol{\chi_u}$.

Two types of uncertainty models are considered:

(i) Models that describe the uncertainty of the position of objects. These models connect the modeled *object* frames $\{o\}$ with the estimated *object* frames $\{o'\}$; and

(ii) Models that describe the uncertainty of a feature with respect to the object. These models describe the uncertainty about the *shape* of the objects and connect the modeled feature frames $\{f\}$ to the estimated feature frames $\{f'\}$.

Hence $\boldsymbol{\chi_u}$ is divided into $\boldsymbol{\chi_{uI}}$, which models the "position" uncertainty between $\{o1\}$ and $\{o1'\}$; $\boldsymbol{\chi_{uII}}$, which models the "shape" uncertainty between $\{f1\}$ and $\{f1'\}$; $\boldsymbol{\chi_{uIII}}$, which models the "shape" uncertainty between $\{f2\}$ and $\{f2'\}$; and $\boldsymbol{\chi_{uIV}}$, which models the "position" uncertainty between $\{o2\}$ and $\{o2'\}$.

Equations (3.1)-(3.4) are extended to include the influence of the uncertainty coordinates $\boldsymbol{\chi_u}$:

$$l(\boldsymbol{q}, \boldsymbol{\chi_f}, \boldsymbol{\chi_u}) = \boldsymbol{0}, \tag{3.5}$$

$$\boldsymbol{T}^{w}_{o1}(\boldsymbol{q}_I)\boldsymbol{T}^{o1}_{o1'}(\boldsymbol{\chi_{uI}})\boldsymbol{T}^{o1'}_{f1}(\boldsymbol{\chi_{fI}})\boldsymbol{T}^{f1}_{f1'}(\boldsymbol{\chi_{uII}})\boldsymbol{T}^{f1'}_{f2'}(\boldsymbol{\chi_{fII}})\ldots$$

$$\boldsymbol{T}^{f2'}_{f2}(\boldsymbol{\chi_{uIII}})\boldsymbol{T}^{f2}_{o2'}(\boldsymbol{\chi_{fIII}})\boldsymbol{T}^{o2'}_{o2}(\boldsymbol{\chi_{uIV}})\boldsymbol{T}^{o2'}_{w}(\boldsymbol{q}_{II}) = \boldsymbol{I}, \tag{3.6}$$

$$\boldsymbol{J_f}\dot{\boldsymbol{\chi}}_{\boldsymbol{f}} + \boldsymbol{J_u}\dot{\boldsymbol{\chi}}_{\boldsymbol{u}} + \boldsymbol{J_q}\dot{\boldsymbol{q}} = \boldsymbol{0}, \text{ and} \tag{3.7}$$

$$\boldsymbol{J_f}\ddot{\boldsymbol{\chi}}_{\boldsymbol{f}} + \dot{\boldsymbol{J}}_{\boldsymbol{f}}\dot{\boldsymbol{\chi}}_{\boldsymbol{f}} + \boldsymbol{J_u}\ddot{\boldsymbol{\chi}}_{\boldsymbol{u}} + \dot{\boldsymbol{J}}_{\boldsymbol{u}}\dot{\boldsymbol{\chi}}_{\boldsymbol{u}} + \boldsymbol{J_q}\ddot{\boldsymbol{q}} + \dot{\boldsymbol{J}}_{\boldsymbol{q}}\dot{\boldsymbol{q}} = \boldsymbol{0}, \tag{3.8}$$

with the uncertainty jacobian $\boldsymbol{J_u} = \frac{\partial l}{\partial \boldsymbol{\chi_u}}$, and its derivative $\dot{\boldsymbol{J}}_{\boldsymbol{u}} = \frac{d\boldsymbol{J_u}}{dt}$.

3.2.3 Modeling the example application

This section illustrates the different spatial relations needed for the example application.

Force-controlled motion of a probe in contact with the contour of a 2.5D unknown object

A spatial relation is introduced between the $\{probe\}$ frame and the $\{front\}$ frame, modeling the following auxiliary coordinates, $\boldsymbol{\chi_{f1}}$, which are shown in Figure 3.6:

- z: the distance from the front plane of the object to the actual contact point,
- s: the arc length along the contour,

- y: the distance of the probe to the contour perpendicular to the contour, and
- θ, ϕ and ψ: ZYX-Euler angles representing the orientation of the probe with respect to the contour.

This results in $\boldsymbol{\chi_{f1}} = (z, s, y, \theta, \phi, \psi)$. Since we are dealing with an *unknown* contour, this limited set of coordinates cannot be used at the position level to model the relative position between the $\{probe\}$ frame and the $\{front\}$ frame. At the velocity level this minimal set however does exist (De Schutter et al. 2007, Section 5.3), hence the relative position has to be updated each time step by integrating the coordinate velocities. The following uncertainty coordinates, $\boldsymbol{\chi_{u}}_{contact}$, are added to model the unknown contour:

- y_u: the distance between the modeled and the real contour and
- θ_u: the orientation error between the tangents of the modeled and the real contour.

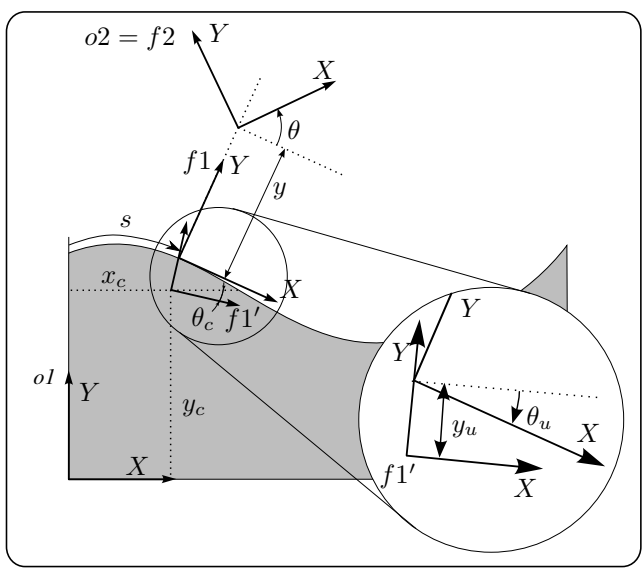

Figure 3.6: Overview of the most important auxiliary coordinates and the uncertainty coordinates modeled for the force-controlled manipulation between a probe ($o2$) and an unknown 2.5D object ($o1$). x_c, y_c and θ_c depend on the model of the contour.

The resulting scene graph including the introduced spatial relation is shown in Figure 3.7.

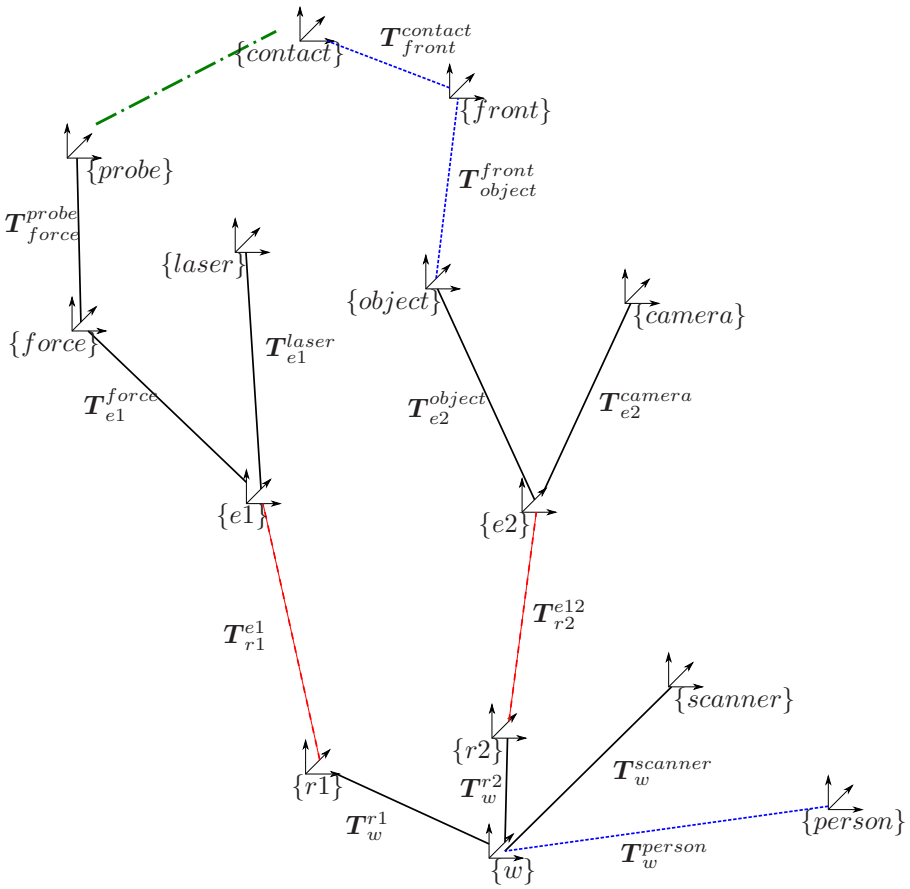

Figure 3.7: The "scene graph" of the example application including the spatial relation (green dash-dotted line) introduced by the unknown contour tracking skill.

Estimating the orientation and height of a plane

A spatial relation is introduced for the estimation of the orientation and height of the object's frontal plane with respect to the {*object*} frame using a laser distance measurement. Figure 3.8 shows the modeled frames. The unknown

orientation and height are modelled as uncertainty coordinates $\boldsymbol{\chi_{u}}_{front} = (h_u, \phi_u, \psi_u)$ between $\{object\}$ and $\{front\}$. A spatial relation between the $\{laser\}$ frame and the $\{front\}$ frame is introduced containing the following auxiliary coordinates $\boldsymbol{\chi_{f}}_{2}$:

- z: the distance along the laser beam axis from the sensor to the front plane,
- θ: the orientation about the laser beam axis,
- ϕ and ψ: the orientation between the laser beam axis and the normal of the front plane, and
- x and y: the translation on the front plane from the center of the plane to the projection of the laser beam on the front plane.

The most important coordinate in this spatial relation is z, which directly maps onto the laser distance measurement.

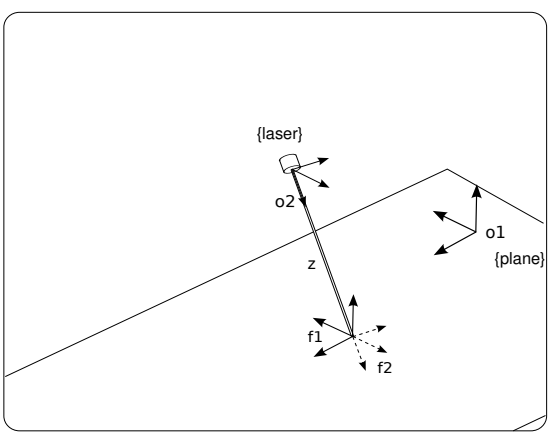

Figure 3.8: Object and feature frames modeled for the spatial relation between a laser and a plane.

The resulting scene graph including the introduced spatial relation is shown in Figure 3.9.

Pointing a camera to a person

To point a camera to the person which is closest to the robotic system the following spatial relation is introduced between the $\{camera\}$ frame and the

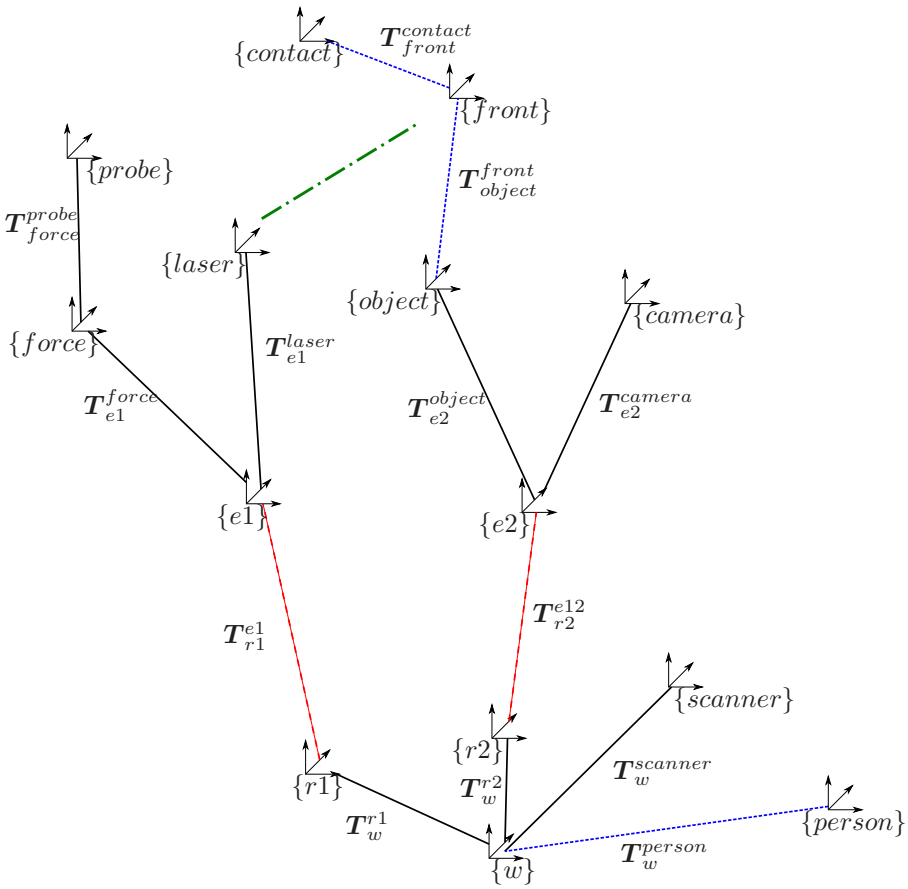

Figure 3.9: The "scene graph" of the example application including the spatial relation (green dash-dotted line) introduced by the plane estimating skill.

$\{person\}$ frame which are shown in Figure 3.10, with the following auxiliary coordinates:

- x and y: this is the position of the projection of the tracked person in the $X-Y$ plane of the $\{camera\}$ frame
- z: the projection distance of the person to the $\{camera\}$ frame
- θ: the orientation about the projection axis

- ϕ and ψ: the orientation of the projection axis with respect to the $\{person\}$ frame.

This results in $\boldsymbol{\chi_{f3}} = (x, y, z, \theta, \phi, \psi)$.

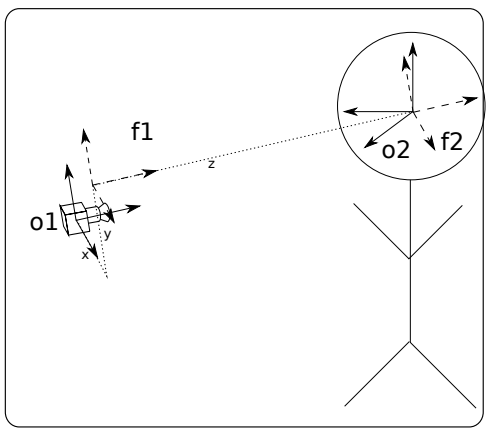

Figure 3.10: Spatial relation introduced to point a camera ($o1$) to a person ($o2$).

Distance between two objects frames

To keep a safe distance between the force-controlled manipulation and the closest person moving in the environment a spatial relation between the $\{contact\}$ frame and the $\{person\}$ frame, shown in Figure 3.13, is introduced. It contains the following auxiliary coordinates $\boldsymbol{\chi_{f4}}$:

- θ_1: the orientation along the Z-axis of the $\{person\}$ frame (which has the same orientation as the world frame $\{w\}$) to point the X-axis of the $\{person\}$ frame to the projection of the $\{contact\}$ frame in the X-Y plane of the $\{person\}$ frame,
- x: the distance between the origin of the $\{person\}$ frame and the projection of the $\{contact\}$ frame in the X-Y plane of the $\{person\}$ frame,
- z: the distance between the projection of the $\{contact\}$ frame in the X-Y plane of the $\{contact\}$ frame,
- ϕ and ψ: the orientation needed to align the $X-Y$ plane of the $\{contact\}$ frame with the $X-Y$ plane of the $\{person\}$ frame, and

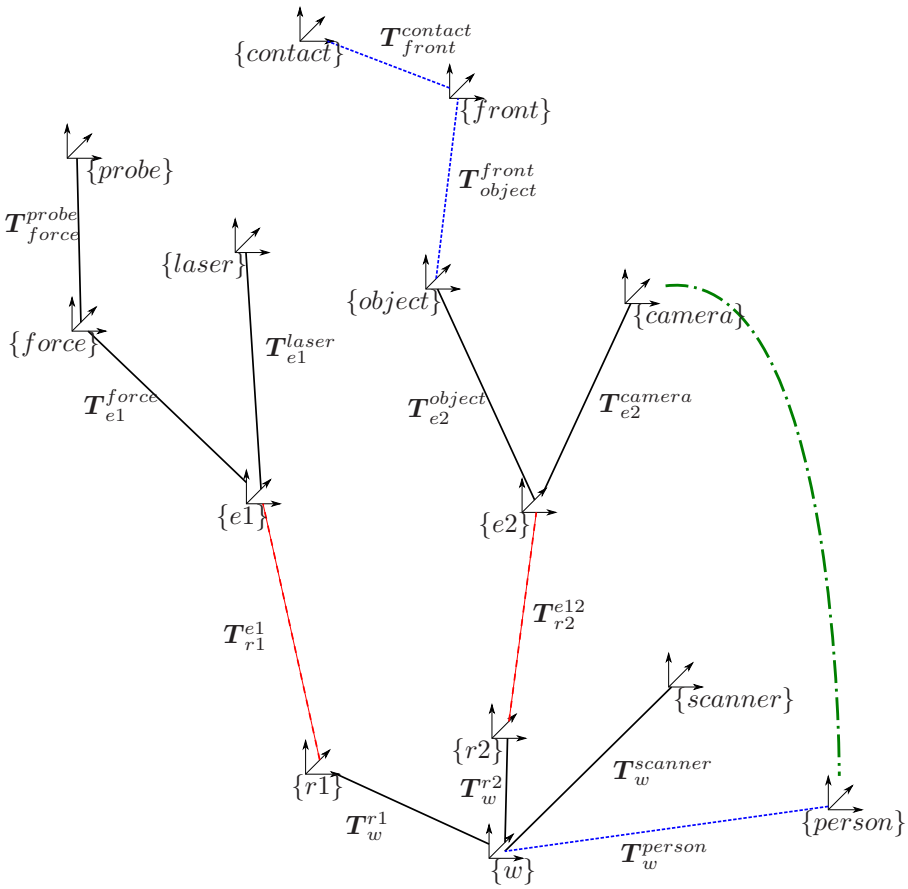

Figure 3.11: The "scene graph" of the example application including the spatial relation (green dash-dotted line) introduced by the camera pointing skill.

- θ_2: the orientation along the Z-axis of the previously aligned $\{contact\}$ frame to the actual $\{contact\}$ frame.

This results in $\boldsymbol{\chi}_{\boldsymbol{f}\,4} = (\theta_1, x, y, \phi, \psi, \theta_2)$.

The resulting scene graph including all modeled spatial relations is shown in Figure 3.14.

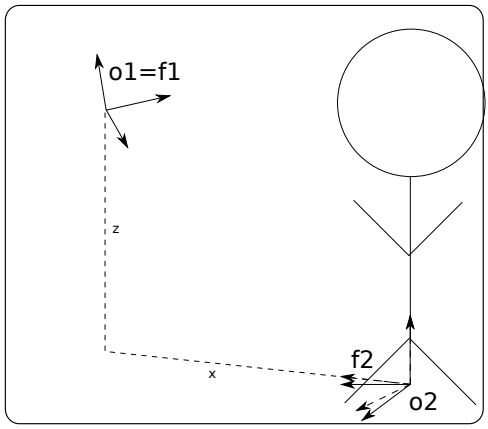

Figure 3.12: The spatial relation introduced between the $\{contact\}$ frame ($o1$) and the $\{person\}$ frame ($o2$) to model the projected distance.

3.3 Configuring the desired motion and estimation

3.3.1 Specifying motion by imposing constraints

The spatial relations are modeled such that the specification of the constraints, between two objects and the measurement equations for the uncertainty estimation is as intuitive and straightforward as possible.

To specify the actual motion the output variables $\boldsymbol{y}$ have to be *selected* from the available controllable coordinates:

$$\boldsymbol{y} = \boldsymbol{f}(\boldsymbol{q}, \boldsymbol{\chi_f}) \tag{3.9}$$

The available controllable coordinates are the robot manipulator joint coordinates, $\boldsymbol{q}$, and the spatial relations auxiliary coordinates, $\boldsymbol{\chi_f}$.

The goal of modeling the spatial relations is to make the relation $\boldsymbol{f}$ between the output $\boldsymbol{y}$, on which we want to impose constraints, and the auxiliary coordinates $\boldsymbol{\chi_f}$ as simple as possible. In general $\boldsymbol{f}$ can be any function but in most cases $\boldsymbol{f}$ can be modeled by selection matrices $\boldsymbol{C_f}$ and $\boldsymbol{C_q}$, which select a single auxiliary coordinate $\chi_{\boldsymbol{f}}$ or a single robot joint coordinate q as output:

$$\boldsymbol{y} = \boldsymbol{C_q q} + \boldsymbol{C_f \chi_f}. \tag{3.10}$$

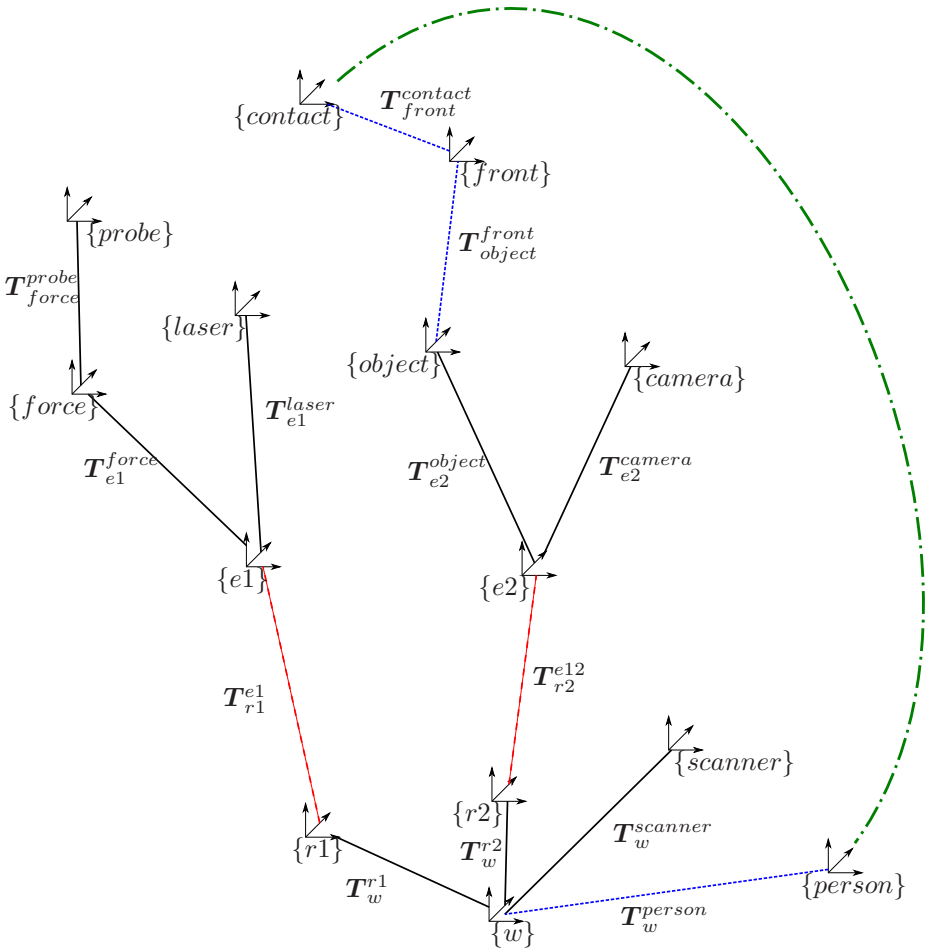

Figure 3.13: The "scene graph" of the example application including the virtual manipulator (green dash-dotted line) introduced by the keep distance skill.

Constraints are defined on the output variables $\boldsymbol{y}$.

$$\boldsymbol{y} = \boldsymbol{y}_d, \tag{3.11}$$

where the subscript $_d$ denotes a desired quantity. Inequality relations can also be used if the constraint optimization solver that will be used to solve the desired robot motion can deal with these. Decré et al. 2009 explains how this can be done.

How the constraints are defined mathematically depends on the desired motion

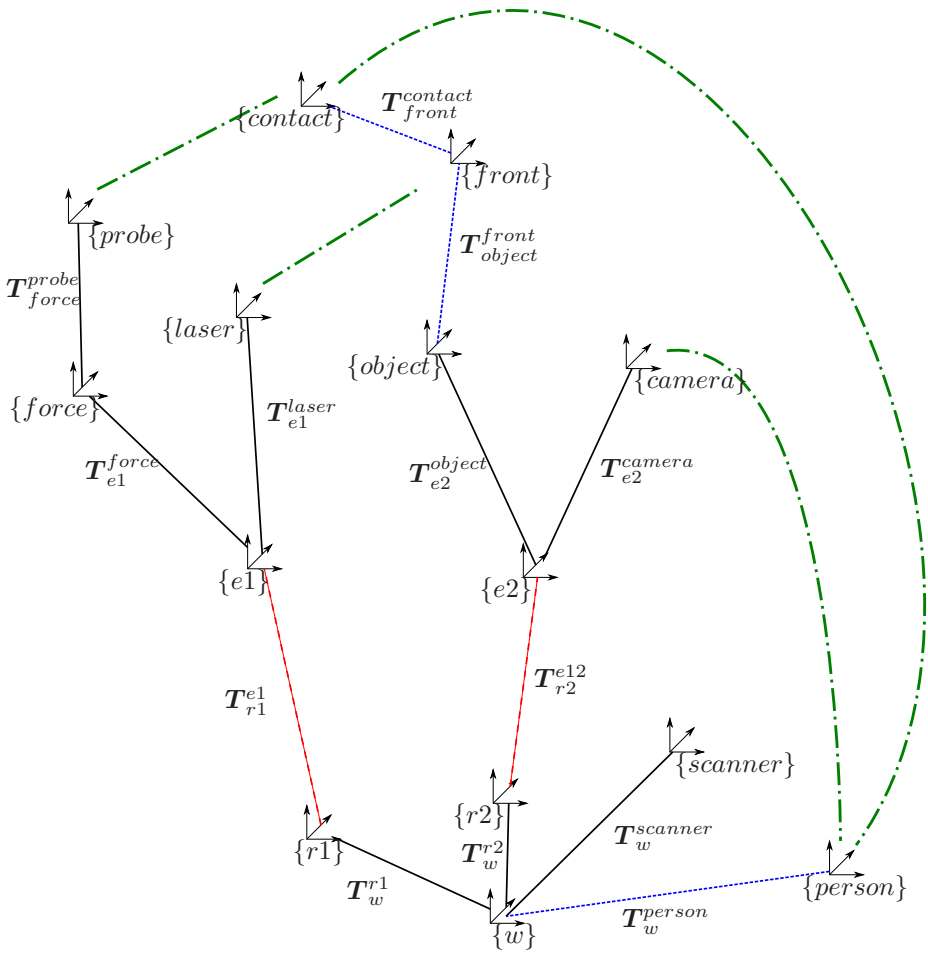

Figure 3.14: The "scene graph" of the example application including the virtual manipulators (green dash-dotted lines) introduced by the different skills.

and on the solution domain, being velocity, acceleration or torque. For example in a velocity-resolved scheme a fixed-value constraint can look like:

$$\dot{y}_d = K_p(y_d - y_m), \tag{3.12}$$

with K_p, a feedback gain to compensate for modeling and integration errors, y_d the desired value for y and y_m the measured value for y.

3.3.2 Specifying the estimation problem

The goal of the estimation problem is (i) to provide an estimate for the outputs $\boldsymbol{y}$ to be used in the constraint definition (3.15) if the output cannot be measured directly, (ii) to provide and estimate for the uncertainty coordinates $\boldsymbol{\chi_u}$ and their derivatives and (iii) to maintain the consistency between the auxiliary coordinates $\boldsymbol{\chi_f}$ and the other coordinates $\boldsymbol{q}$ and $\boldsymbol{\chi_u}$.

The estimation is based on a prediction-correction procedure. The prediction step is based on the *extended system model*.

The extended system model $\boldsymbol{P}$, $\boldsymbol{Q}$ relates the change of the entire system state, $\left(\boldsymbol{q}^i, \boldsymbol{\chi_f}^i, \boldsymbol{\chi_u}^i\right)$ (where a^i is the vector of a and its derivatives up to order i)[1], to the system state itself, and the control input of the robot manipulators $\boldsymbol{u}$:

$$\frac{d}{dt}\begin{pmatrix}\boldsymbol{q}^i\\ \boldsymbol{\chi_f}^i\\ \boldsymbol{\chi_u}^i\end{pmatrix} = \begin{pmatrix}\boldsymbol{P}_{robot} & \boldsymbol{0} & \boldsymbol{0}\\ \boldsymbol{P}^{\boldsymbol{q}}_{loop} & \boldsymbol{0} & \boldsymbol{P}^{\boldsymbol{\chi_u}}_{loop}\\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{P}_{uncert}\end{pmatrix}\begin{pmatrix}\boldsymbol{q}^i\\ \boldsymbol{\chi_f}^i\\ \boldsymbol{\chi_u}^i\end{pmatrix} + \begin{pmatrix}\boldsymbol{Q}_{robot}\\ \boldsymbol{Q}_{loop}\\ \boldsymbol{0}\end{pmatrix}\boldsymbol{u}. \tag{3.13}$$

The extended system model contains three different parts:

- The linearized process model of the robotic manipulator $\boldsymbol{P}_{robot}$, $\boldsymbol{Q}_{robot}$. In case of an ideal velocity-controlled manipulator, $\dot{\boldsymbol{q}} = \boldsymbol{u} = \dot{\boldsymbol{q}}_d$, $\boldsymbol{P}_{robot} = \boldsymbol{0}$ and $\boldsymbol{Q}_{robot} = \boldsymbol{1}$;
- The loop closure equations (3.7) and (3.8) solved for the auxiliary coordinates $\boldsymbol{P}^{\boldsymbol{q}}_{loop}$, $\boldsymbol{P}^{\boldsymbol{\chi_u}}_{loop}$, $\boldsymbol{Q}_{loop}$. In case of an ideally velocity-resolved control scheme, $\boldsymbol{P}^{\boldsymbol{q}}_{loop} = \boldsymbol{0}$, $\boldsymbol{P}^{\boldsymbol{\chi_u}}_{loop} = \boldsymbol{J}_{\boldsymbol{f}}^{-1}\boldsymbol{J}_{\boldsymbol{u}}$ and $\boldsymbol{Q}_{loop} = \boldsymbol{J}_{\boldsymbol{f}}^{-1}\boldsymbol{J}_{\boldsymbol{q}}$; and
- The linearized process model of the modelled uncertainties $\boldsymbol{P}_{uncert}$.

To perform the correction step the measurement equation

$$\boldsymbol{h}(\boldsymbol{q}, \boldsymbol{\chi_f}, \boldsymbol{\chi_u}) = \boldsymbol{z}, \tag{3.14}$$

must be provided, this equation relates the measurement to the available coordinates $\boldsymbol{q}, \boldsymbol{\chi_f} and \boldsymbol{\chi_u}$.

To make the definition of $\boldsymbol{h}$ as simple as possible, the estimation usually introduces a dedicated spatial relation. This relation is modelled so that the auxiliary coordinates introduced by the relation, directly map on the

[1] The differentiation order for the different coordinates is the same, except for $\boldsymbol{\chi_u}$, which can have higher-order derivatives in the system state if the process model $\boldsymbol{P}_{uncert}$ contains these higher-order derivatives.

measurements of the used sensor. The loop closure equations (3.5) (which are automatically derived) will link the new auxiliary coordinates to the uncertainty coordinates in the virtual kinematic loop.

3.4 Calculating the robot motion from the constraint definition

How to calculate the robot motion from the constraint definition depends on the control input of the robotic system. This section presents the equations for velocity-resolved and acceleration-resolved robot manipulators.

3.4.1 Velocity-resolved robot manipulator

In the case of a velocity-resolved robot manipulator, for which the control input are the joint velocities $\dot{\boldsymbol{q}}$, Eqs. (3.9) and (3.11) are derived and combined into:

$$\dot{\boldsymbol{y}}_d = \boldsymbol{C_q}\dot{\boldsymbol{q}} + \boldsymbol{C_f}\dot{\boldsymbol{\chi}}_{\boldsymbol{f}}. \tag{3.15}$$

In general $\boldsymbol{C_q} = \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{q}}$ and $\boldsymbol{C_f} = \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{\chi_f}}$. In most cases $\boldsymbol{C_q}$ and $\boldsymbol{C_f}$ are just selection matrices as already highlighted in Section 3.3.1.

$\dot{\boldsymbol{\chi}}_{\boldsymbol{f}}$ is calculated from Eq.(3.7):

$$\dot{\boldsymbol{\chi}}_{\boldsymbol{f}} = -\boldsymbol{J_f}^{-1}\boldsymbol{J_q}\dot{\boldsymbol{q}} - \boldsymbol{J_f}^{-1}\boldsymbol{J_u}\dot{\boldsymbol{\chi}}_{\boldsymbol{u}}, \tag{3.16}$$

this expression combined with Eq. (3.15) results in:

$$\begin{aligned} \left(\boldsymbol{C_q} - \boldsymbol{C_f}\boldsymbol{J_f}^{-1}\boldsymbol{J_q}\right)\dot{\boldsymbol{q}} &= \dot{\boldsymbol{y}}_d + \boldsymbol{C_f}\boldsymbol{J_f}^{-1}\boldsymbol{J_u}\dot{\boldsymbol{\chi}}_{\boldsymbol{u}}, \\ \boldsymbol{A}\dot{\boldsymbol{q}}_d &= \dot{\boldsymbol{y}}_d + \boldsymbol{B}\dot{\boldsymbol{\chi}}_{\boldsymbol{u}}, \text{ and} \\ \boldsymbol{A}\dot{\boldsymbol{q}}_d &= \dot{\boldsymbol{y}}_d^{\circ}, \end{aligned} \tag{3.17}$$

with $\boldsymbol{A} = \boldsymbol{C_q} - \boldsymbol{C_f}\boldsymbol{J_f}^{-1}\boldsymbol{J_q}$, $\boldsymbol{B} = \boldsymbol{C_f}\boldsymbol{J_f}^{-1}\boldsymbol{J_u}$ and $\dot{\boldsymbol{y}}_d^{\circ} = \dot{\boldsymbol{y}}_d + \boldsymbol{B}\dot{\boldsymbol{\chi}}_{\boldsymbol{u}}$. $\boldsymbol{A}$ is a $(n_c \times n_q)$-matrix, with n_c the total number of constraints and n_q the total number of joints in the robotic system.

Equation (3.17) can be solved for $\dot{\boldsymbol{q}}_d$ in various ways, depending on the *constrainedness* of the motion specification. The constrainedness of a motion specification depends on the relation between n_c and n_q:

The fully-constrained case: $n_c = n_q$

If the number of constraints and the number of robot joints are equal and the constraint equations are independent, the motion specification is called fully constrained. This means only one solution for $\dot{\boldsymbol{q}}_d$ exists. In this case $\boldsymbol{A}$ is square and the inverse $\boldsymbol{A}^{-1}$ can be used to calculate $\dot{\boldsymbol{q}}_d$:

$$\dot{\boldsymbol{q}}_d = \boldsymbol{A}^{-1}\dot{\boldsymbol{y}}_d^{\circ} \tag{3.18}$$

The under-constrained case: $n_c < n_q$

If the number of independent constraints is smaller than the number of robot joints the motion specification is called under-constrained, the robotic system is redundant for the motion specification. This means an unlimited number of solutions for $\dot{\boldsymbol{q}}_d$ exists which all satisfy $\dot{\boldsymbol{y}}_d^{\circ}$. A weighted generalized inverse (Doty et al. 1993) $\boldsymbol{A}^{\#}$:

$$\boldsymbol{A}^{\#} = \boldsymbol{M}_q^{-1}\boldsymbol{A}^T\left(\boldsymbol{A}\boldsymbol{M}_q^{-1}\boldsymbol{A}^T\right)^{-1}, \tag{3.19}$$

is used to calculate a unique solution for $\dot{\boldsymbol{q}}_d$, which minimizes the $\boldsymbol{M}_q$-norm:

$$\dot{\boldsymbol{q}}_d = \boldsymbol{A}^{\#}\dot{\boldsymbol{y}}_d^{\circ}. \tag{3.20}$$

A well-known choice for $\boldsymbol{M}_q$ is the joint-space mass matrix of the robotic system. The corresponding solution minimizes the kinetic energy in the system.

The over-constrained case: $n_c > n_q$

If the number of independent constraints is greater than the number of robot joints the motion specification is over-constrained, the robotic system is not able to satisfy all constraints at the same time if the constraint equations are independent. Two approaches exist to solve an over-constrained motion specification:

- *weighting* between the constraints: in this approach a weighted generalized inverse (Doty et al. 1993) $\boldsymbol{A}^{\#}$:

  $$\boldsymbol{A}^{\#} = \left(\boldsymbol{A}^T\boldsymbol{M}_c\boldsymbol{A}\right)^{-1}\boldsymbol{A}^T\boldsymbol{M}_c, \tag{3.21}$$

  is used to calculate the $\boldsymbol{M}_c$-least-squares solution for $\dot{\boldsymbol{q}}_d$. If $\boldsymbol{M}_c = \mathrm{diag}(w_i^2)$, w_i defines the importance/weight of the constraint i relative to the other constraints; and

- *prioritizing* between the constraints: in this approach the constraints are divided into priorities. With the task priority strategy (Maciejewski and Klein 1985; Hanafusa et al. 1981; Nakamura et al. 1987; Siciliano and Slotine 1991), according to the division of the constraints in priorities, the lower-priority constraints only produce a self-motion which does not interfere with the higher-priority constraints. $\boldsymbol{A}$ and $\dot{\boldsymbol{y}}_d^\circ$ get divided into the priorities:

$$\boldsymbol{A} = (\boldsymbol{A}_1, \boldsymbol{A}_2, \ldots, \boldsymbol{A}_N)^T, \text{ and} \tag{3.22}$$

$$\dot{\boldsymbol{y}}_d^\circ = (\dot{\boldsymbol{y}}_{d\,1}^\circ, \dot{\boldsymbol{y}}_{d\,2}^\circ, \ldots, \dot{\boldsymbol{y}}_{d\,N}^\circ)^T. \tag{3.23}$$

For each priority space

$$\boldsymbol{P}_i = \boldsymbol{I} - \boldsymbol{A}_{1,i}^{\#} \boldsymbol{A}_{1,i}, \tag{3.24}$$

is the projector onto the null-space of the augmented task space

$$\boldsymbol{A}_{1,i} = (\boldsymbol{A}_1, \boldsymbol{A}_2, \ldots, \boldsymbol{A}_i)^T, \tag{3.25}$$

and the joint velocity solution is:

$$\dot{\boldsymbol{q}}_{di} = \dot{\boldsymbol{q}}_{di-1} + \bar{\boldsymbol{A}}_i^{\#} \left(\dot{\boldsymbol{y}}_{d\,i}^\circ - \boldsymbol{A}_i \dot{\boldsymbol{q}}_{di-1}\right), \quad \dot{\boldsymbol{q}}_1 = \boldsymbol{A}_1^{\#} \dot{\boldsymbol{y}}_{d\,1}^\circ, \tag{3.26}$$

with $\bar{\boldsymbol{A}}_i = \boldsymbol{A}_i \boldsymbol{P}_{i-1}$. In each priority space Eq. (3.19) can be used to calculate the weighted generalized inverse, which results in a combination of both approaches. It should be noted that a set of lower-priority constraints can only be satisfied if there are still DOF left in the robotic system to satisfy them.

3.4.2 Acceleration-resolved robot manipulator

A similar derivation can be made at the acceleration level (De Laet and De Schutter 2007) by differentiating Eq. (3.15)

$$\ddot{\boldsymbol{y}}_d = \dot{\boldsymbol{C}}_{\boldsymbol{q}} \dot{\boldsymbol{q}} + \boldsymbol{C}_{\boldsymbol{q}} \ddot{\boldsymbol{q}}_d + \dot{\boldsymbol{C}}_{\boldsymbol{f}} \dot{\boldsymbol{\chi}}_{\boldsymbol{f}} + \boldsymbol{C}_{\boldsymbol{f}} \ddot{\boldsymbol{\chi}}_{\boldsymbol{f}}, \tag{3.27}$$

solving Eq. (3.8) for $\ddot{\boldsymbol{\chi}}_{\boldsymbol{f}}$ and combining this result with Eq. (3.27) leads to:

$$\begin{aligned} \left(\boldsymbol{C}_{\boldsymbol{q}} - \boldsymbol{C}_{\boldsymbol{f}} \boldsymbol{J}_{\boldsymbol{f}}^{-1} \boldsymbol{J}_{\boldsymbol{q}}\right) \ddot{\boldsymbol{q}}_d = \ddot{\boldsymbol{y}}_d &- \left(\dot{\boldsymbol{C}}_{\boldsymbol{q}} - \boldsymbol{C}_{\boldsymbol{f}} \boldsymbol{J}_{\boldsymbol{f}}^{-1} \dot{\boldsymbol{J}}_{\boldsymbol{q}}\right) \dot{\boldsymbol{q}} \\ &- \left(\dot{\boldsymbol{C}}_{\boldsymbol{f}} - \boldsymbol{C}_{\boldsymbol{f}} \boldsymbol{J}_{\boldsymbol{f}}^{-1} \dot{\boldsymbol{J}}_{\boldsymbol{f}}\right) \dot{\boldsymbol{\chi}}_{\boldsymbol{f}} \\ &- \boldsymbol{C}_{\boldsymbol{f}} \boldsymbol{J}_{\boldsymbol{f}}^{-1} \left(\dot{\boldsymbol{J}}_{\boldsymbol{u}} \dot{\boldsymbol{\chi}}_{\boldsymbol{u}} + \boldsymbol{J}_{\boldsymbol{u}} \ddot{\boldsymbol{\chi}}_{\boldsymbol{u}}\right), \text{ and} \\ \boldsymbol{A} \ddot{\boldsymbol{q}}_d = \ddot{\boldsymbol{y}}_d^\circ, & \end{aligned} \tag{3.28}$$

with
$$\ddot{\boldsymbol{y}}_d^\circ = \ddot{\boldsymbol{y}}_d - \left(\dot{\boldsymbol{C}}_{\boldsymbol{q}} - \boldsymbol{C}_{\boldsymbol{f}}\boldsymbol{J}_{\boldsymbol{f}}^{-1}\dot{\boldsymbol{J}}_{\boldsymbol{q}}\right)\dot{\boldsymbol{q}} - \left(\dot{\boldsymbol{C}}_{\boldsymbol{f}} - \boldsymbol{C}_{\boldsymbol{f}}\boldsymbol{J}_{\boldsymbol{f}}^{-1}\dot{\boldsymbol{J}}_{\boldsymbol{f}}\right)\dot{\boldsymbol{\chi}}_{\boldsymbol{f}} - \boldsymbol{C}_{\boldsymbol{f}}\boldsymbol{J}_{\boldsymbol{f}}^{-1}\left(\dot{\boldsymbol{J}}_{\boldsymbol{u}}\dot{\boldsymbol{\chi}}_{\boldsymbol{u}} + \boldsymbol{J}_{\boldsymbol{u}}\ddot{\boldsymbol{\chi}}_{\boldsymbol{u}}\right).$$

Equation (3.28) can be solved for $\ddot{\boldsymbol{q}}_d$ in the same way as in the velocity-resolved case, depending on the contrainedness of the motion specification.

3.4.3 Remarks

Natural constraints and artificial constraints

Two different kinds of constraints can be identified, *natural* constraints imposed by physical contact with the environment and *artificial* constraints imposed by the programmer to specify the desired motion. Natural constraints are considered hard constraints which cannot be violated. All artificial constraints are considered to be soft constraints which can be violated, and are controlled with a specific configured control equation. Of course, when dealing with both natural and artificial constraints in the same motion specification, the priority-based approach has to be used with the natural constraints in the highest priority space and the artificial constraints in the lower-priority spaces, to make sure no motion is applied in the directions of the natural constraints.

Dealing with singularities

In robot singularities, the proposed solutions will still satisfy the feasible constraints, but excessive joint velocities may occur during the transition into and out of the singularities. To deal with this problem, the generalized inverse shall rather be a damped least-squares inverse (Nakamura and Hanafusa 1986; Wampler 1986).

3.5 Conclusion

The iTaSC methodology allows us to derive the robot manipulator control input from the specification of constraints. The methodology includes the estimation procedure for geometric uncertainty derived from the specification of the measurement equations and the process model of the uncertainty coordinates.

Chapter 4

Extending iTaSC with constraints in any feature space

This chapter explains how constraints can be defined not only on the geometric auxiliary coordinates of the spatial relations defined by De Schutter et al. 2007, but also directly on coordinates from any feature space when instead of a spatial relation for the feature coordinates an *interaction model* can be defined. The methodology presented in this chapter is illustrated with a visual servoing task, which combines constraints on features in the image-space with other task-related constraints in the cartesian and joint space.

4.1 Definition of a constraint in a feature space

The iTaSC methodology as explained in Sect. 1.2.1 and in (De Schutter et al. 2007) always explicitly models six independent auxiliary coordinates $\boldsymbol{\chi_f}$ between two object frames $\{o1\}$ and $\{o2\}$. These coordinates $\boldsymbol{\chi_f}$ model a controllable virtual manipulator between the two object frames. In some cases it is not possible or necessary to model six independent auxiliary coordinates. In this case an *interaction model* is used to specify the relation between a (set of) auxiliary or *feature* coordinate(s) $\boldsymbol{\chi_f}$ and the pose between both objects:

$$\boldsymbol{\chi_f} = \boldsymbol{h}(\boldsymbol{T}_{o1}^{o2}), \tag{4.1}$$

with $\boldsymbol{h}$ the interaction model. At the velocity level, $\dot{\boldsymbol{\chi}}_{\boldsymbol{f}}$ depends on the twist between both objects:

$$\dot{\boldsymbol{\chi}}_{\boldsymbol{f}} = \boldsymbol{H}_{o1}\mathbf{t}_{o1}^{o2}, \tag{4.2}$$

with $\boldsymbol{H}$ the *Interaction Matrix* (Samson et al. 1991; Espiau et al. 1992).

In the original iTaSC methodology $\boldsymbol{h}$ can be seen as the the inverse position kinematics (IPK) of the modeled virtual manipulator. In most cases the analytical closed form solution IPK of a 6D virtual manipulator does not exist. The inverse relation, or forward position kinematics (FPK) is usually a lot easier:

$$\boldsymbol{h}^{-1}(\boldsymbol{\chi}_{\boldsymbol{f}}) = \boldsymbol{T}_{o1}^{o2}, \tag{4.3}$$

or in differential form:

$$\boldsymbol{H}^{-1}\dot{\chi}_{\boldsymbol{f}=o1}\mathbf{t}_{o1}^{o2}. \tag{4.4}$$

One of the main reasons for the need to model the six independent auxiliary coordinates between two objects in the original iTaSC methodology is the fact that the jacobian $\boldsymbol{J}_{\boldsymbol{f}}$ needs to be invertible in Eqs. (3.15) and (3.16).

Using Eq. (4.4) in the velocity loop (3.7) results in:

$$\boldsymbol{J}_{\boldsymbol{q}}\dot{\boldsymbol{q}} + \boldsymbol{J}_{\boldsymbol{u}}\dot{\boldsymbol{\chi}}_{\boldsymbol{u}} + \boldsymbol{H}^{-1}\dot{\boldsymbol{\chi}}_{\boldsymbol{f}} = 0, \tag{4.5}$$

and Eq. (3.16) becomes:

$$\dot{\boldsymbol{\chi}}_{\boldsymbol{f}} = -\boldsymbol{H}\boldsymbol{J}_{\boldsymbol{q}}\dot{\boldsymbol{q}} - \boldsymbol{H}\boldsymbol{J}_{\boldsymbol{u}}\boldsymbol{\chi}_{\boldsymbol{u}}. \tag{4.6}$$

Notice the resemblance of Eq. (4.6) with Eq. (3.16), $\boldsymbol{J}_{\boldsymbol{f}}^{-1}$ is replaced by $\boldsymbol{H}$. In the original iTaSC methodology $\boldsymbol{J}_{\boldsymbol{f}}$ had to be a 6×6 invertible matrix. In the presented extended methodology however, $\boldsymbol{H}$ does not have to be a 6×6 matrix but can be any $n_f \times 6$ matrix with n_f the number of feature coordinates, nor $\boldsymbol{H}$ has to be invertible. Hence in the presented extended methodology there is no need for six independent coordinates $\boldsymbol{\chi}_{\boldsymbol{f}}$.

Combining Eq. (4.6) and the constraint definition on the outputs (3.15) results in:

$$\begin{aligned} \left(\boldsymbol{C}_{\boldsymbol{q}} - \boldsymbol{C}_{\boldsymbol{f}}\boldsymbol{H}\boldsymbol{J}_{\boldsymbol{q}}\right)\dot{\boldsymbol{q}} &= \dot{\boldsymbol{y}}_d + \boldsymbol{C}_{\boldsymbol{f}}\boldsymbol{H}\boldsymbol{J}_{\boldsymbol{u}}\dot{\boldsymbol{\chi}}_{\boldsymbol{u}}, \text{ and} \\ \boldsymbol{A}_{ext}\dot{\boldsymbol{q}} &= \dot{\boldsymbol{y}}_{d\,ext}^{\circ}, \end{aligned} \tag{4.7}$$

with $\boldsymbol{A}_{ext} = \boldsymbol{C}_{\boldsymbol{q}} - \boldsymbol{C}_{\boldsymbol{f}}\boldsymbol{H}\boldsymbol{J}_{\boldsymbol{q}}$ and $\dot{\boldsymbol{y}}_{d\,ext}^{\circ} = \dot{\boldsymbol{y}}_d + \boldsymbol{C}_{\boldsymbol{f}}\boldsymbol{H}\boldsymbol{J}_{\boldsymbol{u}}\dot{\boldsymbol{\chi}}_{\boldsymbol{u}}$.

Equation (4.7) can be solved using the procedure described in Sect. 3.3.1.

4.2 Example: Constraint-based image-based visual servoing

This section reformulates image-based visual servoing (IBVS) in the extended iTaSC motion specification and illustrates how easily extra task-specific constraints from different task spaces can be combined in the generic framework for sensor-based robot systems presented by De Schutter et al. 2007. Real world experimental results are presented for IBVS constraints with extra cartesian-space constraints.

The goal of the application is to track an object in the image by specifying constraints on the measured projected features t_i (image points) of the object in the image-plane of the camera, $\{image\}$. The camera is attached to the robot end effector $\{ee\}$ (eye in hand configuration). The object is moved around by a human and the constraints on the projected features use a constant feedback control law to keep the feature at the desired position in the image.

Extra task-related constraints include a minimum distance between the robot end effector and a vertical wall. The constraint is defined and coordinated as explained in Sect. 5.5.4. In the joint space constraints are added on the joint values of joint 3 and 5 of our robot to keep it away from singular positions. An overview of the experimental setup is shown in Fig. 4.1.

Notice that although the image, distance and joint constraints are defined in different operational spaces they can be easily combined, as shown in the next sections.

4.2.1 The Interaction model of the camera

In this application the projection of a certain point in the camera image is controlled to stay at a desired position in the camera image.

As for classical IBVS applications, the camera is modeled by its intrinsic matrix $\boldsymbol{K}$ (an upper triangular 3×3 matrix containing focal length, principal point and skew parameter information) and a frame attached to the sensor itself defined as *camera frame* (having its z axis coincident with the focal axis and origin in the projection center) (Hartley and Zisserman 2004). In this application $\boldsymbol{K}$ is supposed to be known and assumed to be constant (no zoom). Let $\boldsymbol{p} = (x_p, y_p, z_p)^T$ be the $3D$ coordinates of a generic target point expressed in the camera frame; in the *normalized image* the corresponding *normalized image point* homogeneous coordinates $\boldsymbol{m}$ are defined by:

$$\bar{\boldsymbol{m}} = \left(\boldsymbol{m}^T, 1\right)^T = (m_x, m_y, 1)^T = \frac{1}{z_p}\boldsymbol{p}. \tag{4.8}$$

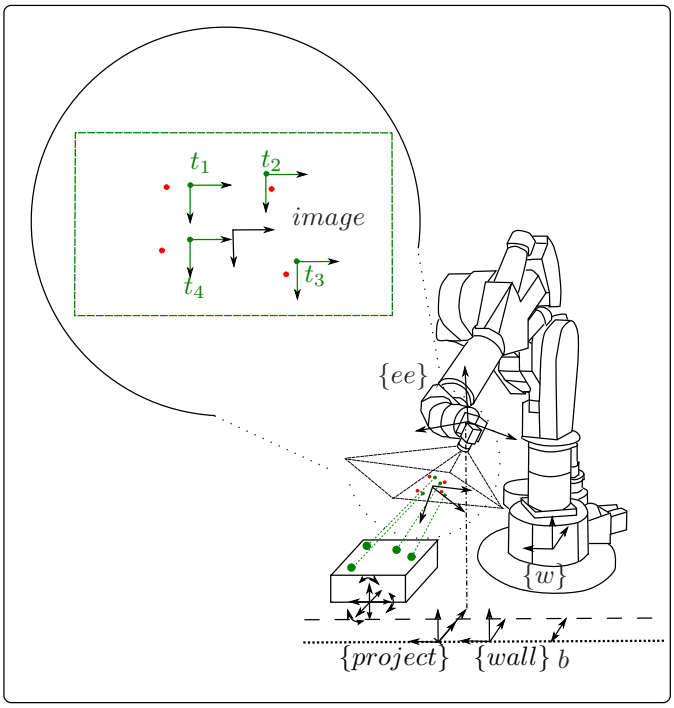

Figure 4.1: Overview of the experiment: the camera is attached to the robot end effector $\{ee\}$; target points t_i are projected by the camera in the *image* plane. The desired positions are represented by the red dots. The dotted line represents the vertical wall, which is located at $\{wall\}$. $\{project\}$ represents the projection of the robot end effector on the wall. The dashed line represents the security border b, which triggers the distance constraint.

These coordinates will correspond to the feature coordinates $\boldsymbol{\chi_f} = \boldsymbol{m}$. The pose between the $\{camera\}$ frame and the target point is unknown but the points can be easily measured in the camera image. If pixel coordinates are used, the same point is defined by the following equation:

$$\bar{\boldsymbol{p}} = (u, v, 1)^T = \boldsymbol{K}\bar{\boldsymbol{m}}. \tag{4.9}$$

Since $\boldsymbol{K}$ is known, it is straightforward to find the normalized coordinates $\boldsymbol{m}$ from the measured pixel coordinates $\bar{\boldsymbol{p}}$ by inverting (4.9) for each image target feature of interest.

The interaction matrix (4.2) in the IBVS case is denoted as $\boldsymbol{H}_{\text{ibvs}}$ and is obtained from the known vision theory (Espiau et al. 1992). Since k features

are measured, $\boldsymbol{H}_{\text{ibvs}}$ will be a $2k \times 6$ matrix structured as follows:

$$\boldsymbol{H}_{\text{ibvs}} = \left(\boldsymbol{H}_{\text{ibvs},1}^T, \ldots, \boldsymbol{H}_{\text{ibvs},i}^T, \ldots, \boldsymbol{H}_{\text{ibvs},k}^T \right)^T, \tag{4.10}$$

where the generic entry $\boldsymbol{H}_{\text{ibvs},i}$ $(i = 1 \ldots k)$ for each measured image *point* feature has the form:

$$\boldsymbol{H}_{\text{ibvs},i} = \begin{pmatrix} -\frac{1}{z_i} & 0 & \frac{m_{xi}}{z_i} & m_{xi}m_{yi} & -(1+m_{xi}^2) & m_{yi} \\ 0 & -\frac{1}{z_i} & \frac{m_{yi}}{z_i} & (1+m_{yi}^2) & -m_{xi}m_{yi} & -m_{xi} \end{pmatrix} \tag{4.11}$$

$\boldsymbol{H}_{\text{ibvs},i}$ is both function of the measured point coordinates in the image $\boldsymbol{m}$ and of the corresponding cartesian $3D$ point depth z_i of the actual feature on the object with respect to the $\{camera\}$ frame. Notice that in this application the point depths are unknown: for this reason approximate fixed values or on-line updated estimates (Conticelli and Allotta 2001) or Sect. 3.3.2 must be used for z_i to build $\boldsymbol{H}_{\text{ibvs}}$.

4.2.2 Configuring the constraints in the image space

The feature coordinates $\boldsymbol{\chi_f}_i$ of the position of a tracked point in the camera image are the normalized image coordinates $\boldsymbol{m}$. For each image point i, a constraint c_i (5.1) is defined in the image space:

- `name`: feature tracking,
- $\boldsymbol{f}$:

$$\boldsymbol{y}_i = \boldsymbol{\chi_f}_i, \tag{4.12}$$

- $\boldsymbol{K}$: at the velocity level:

$$\dot{\boldsymbol{y}}_i = \boldsymbol{k}_p(\boldsymbol{y}_{di} - \boldsymbol{y}_i), \text{ and} \tag{4.13}$$

- $\boldsymbol{\gamma}_i$: the desired value $\boldsymbol{y}_{di}$ and the position feedback constant $\boldsymbol{k}_p$.

The current value for $\boldsymbol{\chi_f}_i$ is determined using the pixel value measurements $\bar{\boldsymbol{p}}$ for each point:

$$\bar{\boldsymbol{m}} = \boldsymbol{K}^{-1}\bar{\boldsymbol{p}}. \tag{4.14}$$

4.2.3 Coordinating image-based visual servoing

The IBVS tracking can be coordinated in various ways.

Object tracking

In this skill at least 3 non-collinear points that are rigidly attached to the same object are constrained to be kept at a constant desired position in the image: $\boldsymbol{C}_s = \boldsymbol{c}_{object}$, $\boldsymbol{w}_{object} = \boldsymbol{1}$, $\boldsymbol{\gamma}$: $\boldsymbol{y}_{dobject} = y_{desired}$, with $\boldsymbol{c}_{object}$ constraints on measured image points of the same *object* and $y_{desired}$ the desired position of the measured image points in the image.

The constraint configuration results in keeping the exact pose between the $\{camera\}$ and the $\{object\}$ frame. If more than 3 non-collinear points are constrained the result will be the $\boldsymbol{M}_c$-least-squares solution with $\boldsymbol{M}_c = \text{diag}(\boldsymbol{w}_{object})$ constructed using the weights defined for each measured image point.

Keep the features in the field of view

In this skill the desired behavior is not to keep the tracked features at their exact desired position but just to keep them in the image view. A boundary is defined at the image plane edges. For each tracked point a constraint is defined to push the image point away from the boundary, towards the center of the image and coordinated as described in Sect. 5.5.4.

Tracking multiple objects simultaneously

In this skill the points do not have to be on the same object. Different objects could be tracked, the constraints however should be defined in such a way that the task still makes sense. Tracking independently moving objects only makes sense if the constraints are not to keep the points at the exact desired position in the image, but rather to just keep them in the field of view as explained in the previous paragraph, or if an appropriate weighting scheme between the different constraints is applied.

4.2.4 Results

The example task is executed on a velocity-controlled industrial robot with a fire-wire camera attached to the end effector. The robot is attached to a PC,

with a linux-based operating system and a RTAI realtime execution extension. To control the robot, the OROCOS-software (Soetens 2006; Bruyninckx 2001) is used. The image processing is done using the OpenCV (OpenCV 2001) library.

In the experiment color images with a resolution of 640×480 pixels are processed. Four image points (on the same object) are selected for tracking before execution. These points should be stable on the object, this way motion of the points corresponds with motion of the object. For these points an *Object tracking* skill, as explained in the previous section is used. Between the object and a wall a *Keep safe distance* skill, as explained in Sect. 5.5.4, is used. The object is moved (by a human) towards the (virtual) wall which is located at $-1m$ of the base frame of the robot. The safety boundary is chosen $b = 0.05m$. This means that the constraint on the distance to the wall is only activated if the robot end effector x-position in the base frame passes $-0.95m$, which is clearly noticeable in Fig. 4.2.

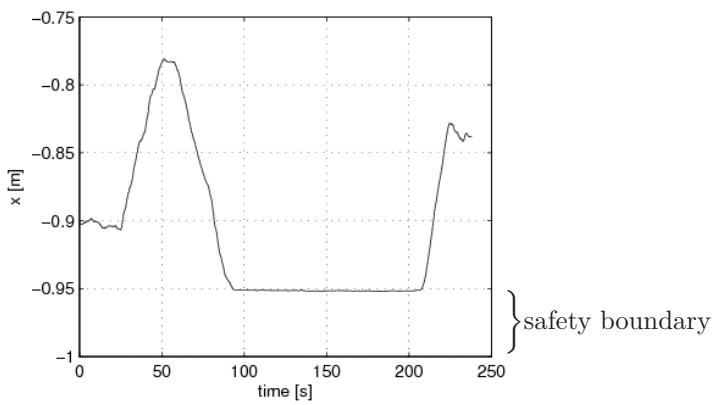

Figure 4.2: X-position of the robot end effector. It is clear that the constraint on the distance from the wall only becomes active after 90 seconds and successfully keeps the robot end effector at a safe distance of $0.05m$ from the wall which is located at $-1m$.

Figure 4.2 shows the distance x between the robot end effector and the wall. As long as the distance of the robot's end effector to the wall does not enter the safety boundary b, the robot is only tracking the selected points. If the robot enters the security zone, which is the case between $90s$ and $200s$, the resulting configured robot motion keeps the error for the tracked points as small as possible without the robot's end effector coming closer to the wall due to the constraint weight configuration.

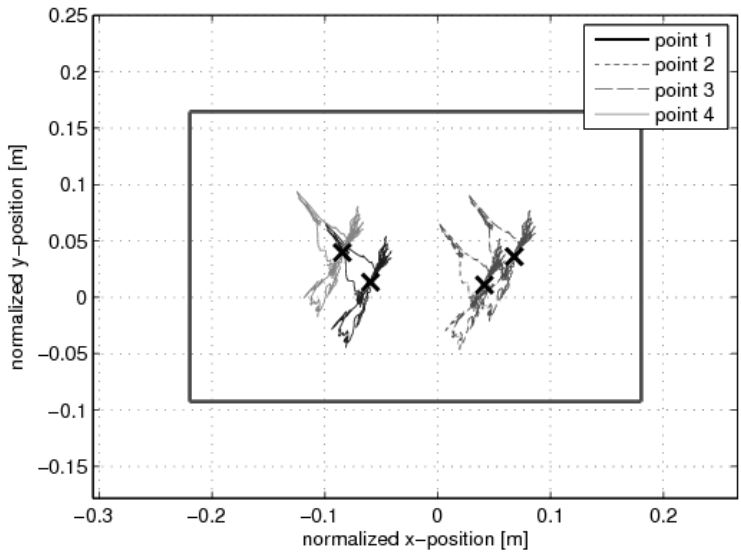

Figure 4.3: Normalized position of the tracked points in the normalized camera-image, the crosses are the desired positions, the rectangle describes the image edge boundary for the activation/deactivation of the tracking constraints.

Figure 4.3 shows the measured positions of the tracked points in the image. Because the feedback constant $\boldsymbol{k}_p$ used to keep the tracked points at their desired position is small, the points do not stay exactly on the desired values $\boldsymbol{y}_{desired}$, defined by the black crosses in Fig. 4.3. Even when the constraint on the distance of the wall is activated the error on the points does not grow significantly. Figure 4.4 shows an experiment with the *Tracking multiple objects simultaneously* skill with two constrained image points on two different objects. The tracking constraints are only activated when the points enter the safety boundary which is indicated with the orange lines. In this experiment the other task-related constraints are not influenced if the tracking constraints are deactivated.

4.3 Conclusion

This chapter shows how the iTaSC methodology from De Schutter et al. 2007 can be extended to incorporate constraints that are defined in any feature space, The presented methodology merges the iTaSC methodology with the Interaction model from the Task Function approach from Samson et al. 1991, which for a given sensor introduces an Interaction matrix $\boldsymbol{H}$, from which a

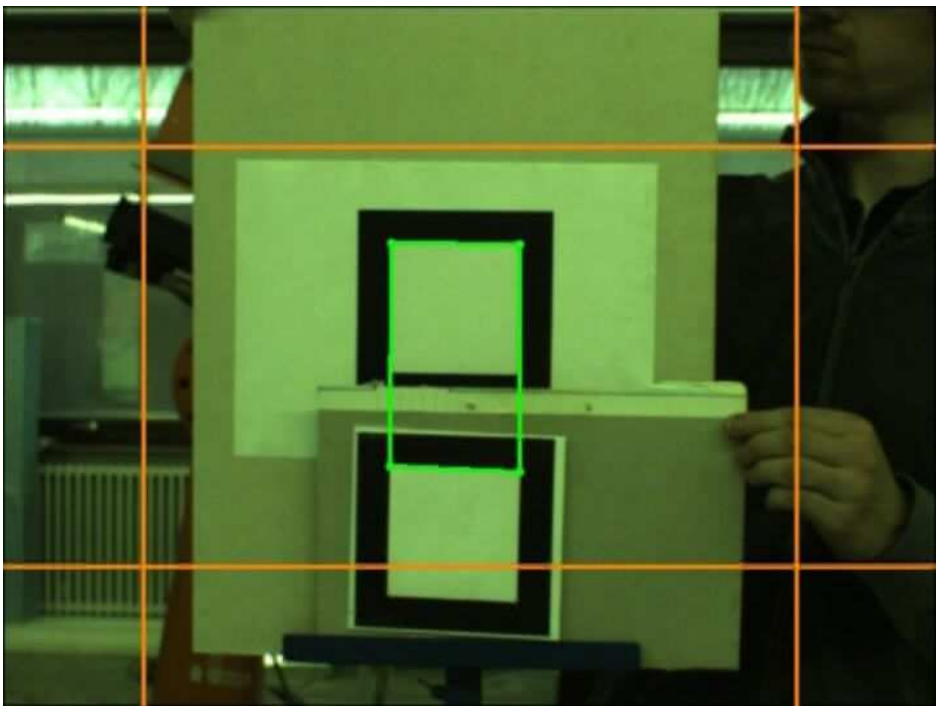

Figure 4.4: The camera image with the four tracked points, in green the actual points and in orange the edge boundary which activates the tracking constraints.

virtual sensor linkage can be built. This virtual linkage can be treated in the same way as the virtual kinematic loops from the regular iTaSC methodology. It however represents the inverse relation between the feature coordinates and the pose between the objects. Therefore it does not need to include 6 independent auxiliary coordinates. The original iTaSC specification methodology can be seen as a special case of the presented extension where the inverse of the feature jacobian $\boldsymbol{J_f}$ is equal to the Interaction matrix $\boldsymbol{H}$ of the presented extended iTaSC formulation. The resulting extended iTaSC methodology allows specifying and controlling constraints in different spaces, image space, cartesian space, joint space or any other task or sensor space, without adding complexity in the control of the robot.

The experimental results show how the constraints in the image space can easily be combined with other task-related constraints.

Chapter 5

Skills

It is possible to fly without motors, but not without knowledge and skill.

Wilbur Wright (1867 - 1912)

The aims of this chapter are to motivate why the concept of a *(control) skill* deserves a prominent place in the context of robot control systems, and to explain how to add the first level of skill functionality to current industrial-grade robot programming languages based on the constraint-based motion specification methodology presented by De Schutter et al. 2007. This chapter formalizes the interface to build, configure and coordinate an iTaSC motion specification. This chapter is motivated by a pragmatic, bottom-up approach to extend the functionality of, in the first place, industrial robot controllers with skill functionalities. Skill functionalities contain the knowledge to coordinate the execution of the instantaneous motions in a task, and to configure each of them appropriately before launching the task.

5.1 Hierarchical control structure

5.1.1 Different control levels

Figure 5.1 sketches an architecture of the next generation industrial robot control that (i) requires only limited extensions to current industrial robot controllers, and (ii) opens up new opportunities for commercial business models. It is a simple instantiation of the well-known (but extremely often re-invented!) three-layer architecture, (Saridis and Stephanou 1977), which is based on how the execution in a biological system is structured.

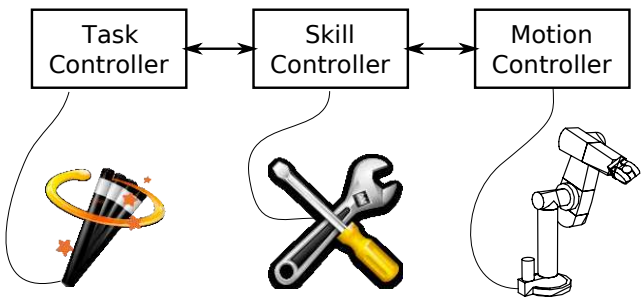

Figure 5.1: Skill-based control system, with decreasing intelligence and increasing control bandwidth from left to right.

This chapter uses *Task*, *Skill* and *Motion*, to address the three different levels and focuses on the skill level, as the link between the *sub-symbolic* world of continuous-time sensors, controllers and estimators, and the *symbolic* reasoning of task planning and decision making.

Since no industrial robot system exists yet that enables full configuration and coordination of the few instantaneous motion controllers they offer, an extra layer based on (De Schutter et al. 2007) is added to encapsulate the existing industrial robot systems. This extra *iTaSC motion controller* offers the functionality needed to use the presented control architecture in practice with existing industrial robot systems. Chapter 7 proposes a software architecture for the next generation of robot controllers that includes the iTaSC methodology to specify tasks. Figure 5.2 sketches this extension in a hierarchical robot control architecture.

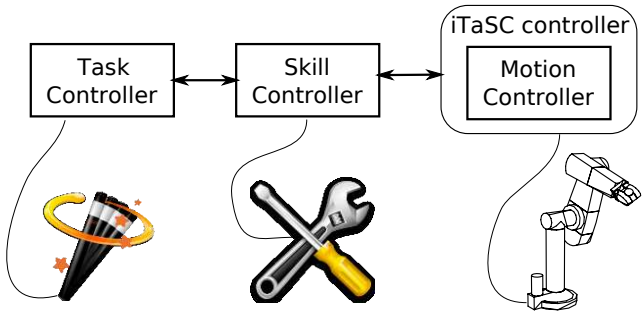

Figure 5.2: Skill-based control system, with decreasing intelligence and increasing control bandwidth from left to right, adapted to include an extra layer to enable full configuration and coordination of existing industrial robot systems.

iTaSC motion controller

The *iTaSC motion controller* extends current industrial controllers with the more universal *instantaneous motion specification and control* functionality explained in Sect. 5.2. This controller's main focus is to compute (in *hard realtime*) the next control input (in a continuous control setting) for its actuators, based on the current configuration of the motion, control and estimation specification, i.e. the desired robot-environment-task interaction. It has a complete formalized model of the robot, including sensors, tools, environment and uncertainties.

This controller has a soft realtime communication channel with the *skill controller*, from which it receives its next instantaneous motion control specification, and to which it communicates the status of the current motion execution.

The knowledge and the mathematics used at this level are typically of *continuous-time* nature: kinematic and dynamic models of the robot, and models of the processes that take place in its interaction with the environment; state-space control and estimation algorithms; 3D perception (localization and tracking); etc.

The skill controller

The *skill controller* is connected to the iTaSC robot controller. It has a formalized mathematical representation of the model used in the iTaSC motion

controller to *interpret* the received numerical data from the iTaSC motion controller and to *interpret* symbolically the (actual and intended) motions in the context of the ongoing skill. The skill controller will fire configured *events* when it detects that:

- some of the instantaneous motion specifications have been satisfied,
- the specified constraints have been violated in such a way that the current configured motion cannot finish successfully anymore, or
- a certain estimator specification converged or is unable to converge.

On the basis of all this information, the *skill controller* decides when, and which newly configured instantaneous motion to send to the *iTaSC motion controller*.

The knowledge and the mathematics used at this level are typically of *discrete time and model* nature: which controllers and observers to activate and to monitor; what configuration to give to them; updating its mathematical model of objects and humans in the robot vicinity; etc.

The task controller

The *task controller* is not a real "controller" anymore (since it is not involved in any direct feedback loop with the real robot and environment), but rather a logical planner. It contains a repository of robot skills (both the versions for the skill controller as for the robot controller), with the extra *meta-level knowledge* about which skills to use for what kind of tasks in what kind of environments, and with what kind of robots, tools and sensors. It is also the place to store and improve models about *human-machine interactions* (both physical and cognitive), *human intent models*, and *human motion patterns*. It has a two-way communication with the *skill controller*:

- to provide the *skill controller* with (formal specifications of) the knowledge needed in a skill, and
- to receive the logs of the execution of all the *skill controllers* it has been coordinating, such that it can learn how to improve the current skill specification and coordination.

The knowledge and the mathematics used at this level are typically of *logical, symbolic* nature, trading-off semantic clarity (for the transformation to skill and robot control specifications) versus natural language (for interaction with the human knowledge builders and machine operators).

5.1.2 Interaction aspects

The proposed hierarchy is however not a *black and white* separation. In different robot domains the different levels seem to overlap. For instance a gain scheduling controller lies in the middle of the skill controller and the iTaSC robot controller. Also the skill controller level can be subdivided into multiple layers, which is usually the case in behavior-based robotics (Brooks 1986).

One task controller is typically connected to multiple skill controllers. All skill controllers interact with one iTaSC robot controller, which encapsulates all robotic systems that are used in the application.

The following complementary *knowledge aspects* must be dealt with, in the interactions between all above-mentioned controller components:

- *Representation*: a formal description is needed for all relevant aspects of instantaneous motions and their coordination. It must be sufficiently abstract to be represented in a mathematical form that computers can process, at both numerical ("sub-symbolic") and symbolic levels. Chapter 8 explains how this can be done for the iTaSC robot controller level.
- *Configuration*: the above-mentioned representation has a number of parameters that have to be given concrete values in each individual motion specification. The parameters can be filled in using off-line human programming, on-line estimation, queries from the task controller, or a dialogue with the human operator.
- *Control and estimation*: the above-mentioned representation and configuration is transformed in executable code to control the robot motion, and to estimate unspecified or uncertain parameter values.
- *Monitoring*: the running controllers and estimators make the robot move in a particular way. The skill controller checks and interprets the correspondence between the actual motion and the specified motion, and reports all deviations that are significant in the context of the executed skill. On the basis of that information, the skill performs its coordination activity, i.e., it decides to continue with the ongoing motion, to reconfigure some of its parameters, or to switch to another available motion specification.

5.2 Skill controller view on the iTaSC motion specification

This section makes concrete suggestions for the *representation* and *specification* of the *instantaneous motion*, focusing on the interaction with the skill control level. The functional aspects of instantaneous motions have already been presented many times in the literature before, with De Schutter et al. 2007 providing a unified modeling and specification framework. The modeling and specification approach of that latter publication is summarized in Sect.1.2.1. The following steps provide a skill controller view on the steps from Sect.1.2.1.

5.2.1 Modeling

Step 1: Construction of the robot scene graph

At the construction of the "scene graph" the kinematic chain(s) of the robot(s) involved in the skill, as well as the *location* of sensors, tools, motors and relevant objects from the environment, are all included in this runtime geometric "skeleton" to which *object* and *feature* frames can be attached, as well as other skill-relevant data structures, such as the *shape* of objects. The concept of a "scene graph" is well-established in computer animation and gaming (Strauss and Carey 1992; MacIntyre and Feiner 1998), and covers exactly what is needed in robotics. The task programmer adds *objects* and *feature frames* on the objects (including the robot kinematic chains) in the scene graph that can interact. Each feature frame also gets a *semantic tag*, to describe its *type*; for example, a controlled feature frame, an estimated feature frame, etc.

5.2.2 Configuration

The first three steps only include the configuration of motion specification.

This step specifies the manipulators process models $\boldsymbol{P}_{robot}$ and $\boldsymbol{Q}_{robot}$, the manipulator control input $\boldsymbol{u}$, the manipulator kinematics, the uncertainty coordinate process models $\boldsymbol{P}_{uncert}$ and the measurement equations $\boldsymbol{h}$, which are needed in Eqs. (3.5) and (3.13).

Step 2: Constraint configuration

During this step the *constraints* between different feature frames are specified, and by doing so a virtual kinematic loop according to Sections 3.2 and 3.3

is created. Each virtual kinematic loop represents an interaction between two feature frames. The interaction *dynamics* in each loop are described by *constraint relationships* between the feature frames of the loop. These constraints can be as simple as a *desired* distance between the origins of both frames, or as complex as the *real* physical dynamics of a welding or deburring process involving multiple tools and sensors. The specified constraints can be *equality* as well as *inequality* constraints (Decré et al. 2009), and the specification can be *partial*, i.e., not all six degrees of freedom between the two feature frames in a loop have to be constrained. The constraints can be *hard* in case of natural constraints, or *soft* in the case of artificial constraints, e.g., via constraint forces derived from a potential field. Each interaction constraint relationship includes a number of parameters. For example, a controlled constraint can have a linear velocity control law, with accompanying feedback parameters.

The concept of feature frames (and constraints between them) is not limited to cartesian space only, but generalizes to joint space, and even to the "sensor spaces" of the available sensors (e.g., cameras or laser scanners), see Chapter 4. The mathematical formulation of the interaction constraint relationships then involves the "*Jacobian transformations*" between the various spaces. Similarly, the presented approach has no problems with involving different (joint, cartesian, sensor) space controllers: position, velocity, acceleration, torque, current, pressure,...

This step defines the virtual manipulator models that create the virtual kinematic loops $\boldsymbol{l}$ (3.5); the output equations $\boldsymbol{f}$ (3.9), which define the output in terms of the controllable coordinates, $\boldsymbol{q}$ and $\boldsymbol{\chi_f}$; and the constraint equations with their desired values (3.11).

Step 3: Optimization configuration

In this step the overall *objective functions* for the constraint and joint space *redundancies* are specified. In a typical task, the available robot systems can have more or fewer degrees of freedom available than required by an instantaneous motion. For instance a 7DOF robot arm has more degrees of freedom available to control a certain distance between its end effector and an object, in contrast with a 4DOF robot arm that has not enough degrees of freedom to accomplish a full 6D pose constraint for its end effector. Hence, one has to indicate with which functions the robot controller should solve this *redundancy*; for example, minimization of the instantaneous kinetic energy of the robot; minimal energy consumption; staying closest to desired trajectories, safety, optimal view of sensors on the scene, etc. or *discrepancy* due to conflicting constraints; for example by weighting or prioritizing constraints.

This step defines the metric $\boldsymbol{M_q}$ (3.19) for each robot manipulator, $\boldsymbol{M_q}$ is the same for all active skills that control the same robot manipulator. Each skill can specify a local metric $\boldsymbol{M_c}$ (3.21) and priorities (3.21), for each of the constraints used in that skill. The task controller adds a global metric $\boldsymbol{M_c}$ and global priorities to indicate the importance of an entire skill with respect to other active skills.

5.2.3 Computation

The last two steps are related to the computation of the robot input from the configured iTaSC motion specification. They are considered to be continuous and are the computation steps of the iTaSC robot controller, which should be able to execute them in realtime.

Step 4: Robot motion computation

In this step the *constrained optimization problem* that results from Steps 1 to 3 for all active constraints is solved. The constraints of all active skills are merged together, because the overall robot motion is defined by all active skills. The solution is the desired joint values in position, velocity, torque,..., depending on the chosen interface for the robot. A good design decouples the choice of a *solver* for the optimization problem from the specification of the problem, and allows the task controller to configure the selection of the most appropriate one.

This step calculates $\boldsymbol{u}$ from the problem specified in the previous steps (3.15) or (3.27).

Step 5: Model update

Update the "scene graph" using the prediction-correction procedure and iterate Step 4.

5.3 iTaSC configuration interface

Once Step 1 is completed and the robot scene graph is constructed, the skill controller will configure the iTaSC specification of the iTaSC robot controller. In order to do so, a formalized interface for Step 2, the motion specification configuration, and Step 3, the motion optimization configuration has to be defined.

5.3.1 Constraint configuration

As already explained in Sect. 3.3.1, a constraint is defined using Eqs. (3.9) and (3.11), respectively, the selection of the controllable variable, and the constraint on this variable. Because of the modeling of the virtual manipulators, every controllable variable has a symbolic interpretation, for instance distance, normal orientation, height, etc.

Constraint c_i:

$$c_i : \langle \texttt{name}; \boldsymbol{f}; \boldsymbol{K}; \boldsymbol{\gamma}_i \rangle \tag{5.1}$$

with

- $c_i.\texttt{name}$, a name tag to give the constraint a human-readable symbolic interpretation,
- $c_i.\boldsymbol{f}$, the output equation (3.9) $\boldsymbol{y}_i = \boldsymbol{f}(\boldsymbol{q}, \boldsymbol{\chi_f})$ selecting which coordinates to constrain,
- $c_i.\boldsymbol{K}$, the constraint control equation, which depends on the overall control resolution scheme (velocity, acceleration or torque) to calculate the desired value of the constraint output, and
- $c_i.\boldsymbol{\gamma}_i$ the constraint control equation's parameters, which depend on the specific type of constraint control equation.

Examples of constraint configurations can be found in Sect. 5.5.

Constraint interface:

- add a new constraint c_i (and a virtual manipulator if needed) using (5.1),
- remove a constraint c_i, and
- change an existing constraint c_i using $c_i.\boldsymbol{\gamma}_i$

5.3.2 Estimation configuration

Section 3.3.2 explains how to estimate the modeled uncertainty coordinates. An uncertainty estimation problem is defined using the linearized process model of the modeled uncertainties $\boldsymbol{P}_{uncert}$ (3.13) and the measurement equation $\boldsymbol{h}$ (3.14). The estimation is configured in two different ways because the sensor measurements do not have to be directly coupled with the uncertainty coordinates.

Sensor interface:

- add or change an existing measurement equation $\boldsymbol{h}$ (and a virtual manipulator if needed).

Uncertainty interface:

- add or change a process model $\boldsymbol{P}_{uncert}$ for the modeled uncertainty coordinates.

5.3.3 Optimization configuration

Only in the under-constrained and over-constrained motion specification cases the optimization configuration is important.

- In the under-constrained case the optimization configuration will define the metric $\boldsymbol{M_q}$ in Eq. (3.19) to specify which norm has to be minimized.
- In the over-constrained case the optimization configuration will define whether a weighting approach, prioritizing approach or a combination of the two will be used. Depending on the approach it will define in which priority-level, p_i a constraint c_i will end up, and/or what the metric $\boldsymbol{M_q}$ in Eq. (3.21) or what the value for w_i of the constraint c_i is, and by doing so define its relative importance among the other constraints in the motion specification.

Optimization interface:

- change for each robotic manipulator, the metric $\boldsymbol{M_q}$,
- change for each constraint c_i, its local priority p_i and/or local weight w_i, and
- change for each skill, its global priority P_j and/or global weight W_j.

5.3.4 Solver configuration

The solver that will calculate the robot manipulator control input from the constraint input can have a set of solver specific parameters σ. For example if a damped least squares solution is used, the damping factor λ, used during the inversion of the task specification matrix $\boldsymbol{A}$ is an element of σ.

Solver interface:

- choose the solver, and
- change the solver specific parameters σ.

5.3.5 Enabling motion coordination: Feedback from the configured motion specification

To make coordination of the currently configured iTaSC specification possible, the iTaSC specification has to send information back. The following continuous data flow from the iTaSC specification to the skill-level should be provided to make a minimal form of coordination possible:

- the current values of the output $\boldsymbol{y}$ of each constraint c_i, and
- the current estimated value and other stochastic data such as the current covariance for each estimated uncertainty coordinate.

5.4 Skill representation and specification

This section makes concrete suggestions for the *representation* and *specification* of skills. The approach of Sect. 5.2 to specify instantaneous motions is quite general. The major difference with the traditional motion specification approach is that specifications are not given via explicit setpoints (being 6D geometric poses) but by constraints. (Of course, an explicit setpoint is just a special case of a constraint, so the presented approach is a superset of the traditional one.) Constraint-based specifications allow for far easier *composition* of partial specifications, redundancy resolution, weighting or prioritizing of motions, and numerical solution via a constrained optimization algorithm. It is then left to the creativity of the human task programmer to use the available flexibility well and to come up with the "optimal" skill model, that is, an intelligent, robot-, environment- and task-dependent choice of:

- which feature frames fit to the skill at hand, and how are the virtual manipulators modeled to make the constraint specification based on the auxiliary coordinates as simple as possible;
- which auxiliary coordinates in the scene graph are known, which ones are to be estimated, and which ones are to be constrained; and
- the desired constraint relationships, the estimator algorithms, and the task optimization functions.

5.4.1 Limitations

This generic approach has limitations too, of course, such as:

- *Predictability*: the ease of a multi-feature frame and constraint-based approach is its *composability*, but if a lot of conflicting constraints are used in the same priority level, the behaviour of the composed system can be hard to predict. The iTaSC motion specification is however still a lot more explicit than the use of potential fields to drive the motion to the desired goal as it is used a lot in behavior-based robotics.
- *Monitoring*: during execution, the skill controller must monitor every aspect of the controlled motion, and how it influences the desired (and also undesired!) robot-task-environment interactions. The skill needs knowledge to identify the dangerous and erroneous conditions, and to react appropriately.
- *Optimization solvers*: only some combinations of goal functions, constraints, and transformation between control spaces (joint, cartesian, sensor) can be solved sufficiently fast for (soft and hard) realtime use.

The common theme of these limitations is *knowledge*: a good skill *knows* how its compositing components can and should work together, and finds an "optimal" execution via appropriate *configuration* (of the individual instantaneous motions' parameters) and *coordination* (of their executions). This is why in research dedicated to learning, the learning of the configuration and coordination through human demonstration, is a very hot topic nowadays (Billard and Siegwart 2004).

5.4.2 Reusability

A skill is responsible for a well-defined, logical part of the overall task, which in our approach is a set of constraints that together form a *functional motion* (Brooks 1986). This set should contain the minimal amount of constraints to describe the desired motion, in order to enable maximum *reusability* of the specified skill. Every skill has a single nominal goal in terms of constraints and defines a symbolic interface for the high-level task. Since the constraints can be independent of the robotic system itself and can only depend on the *capability* of the robotic system, a skill using the same type of constraints can be used in different robotic systems.

5.4.3 Skill representation and execution: the Finite State Machine

A *Finite State Machine* (FSM) (Gill 1962; Ginsburg 1962) can coordinate the *behavior* of one single activity, such as the one of the iTaSC motion controller mentioned before. This section proposes to use a FSM for the representation and execution logic of a skill. This choice is not without any reason. In almost all previous work in different robotic domains, coordination is implemented using FSMs or related methodologies (Brooks 1986; McCarragher and Asada 1992; Finkemeyer et al. 2005; Dillmann et al. 1995).

A FSM is built up out of the following primitives:

- states,
- actions which are executed within a state,
- transitions between different states, and
- events triggering those transitions.

Every state of a skill's FSM is described by one specific configuration of the continuous control laws and observers specified using the previously defined interface of the iTaSC robot controllers.

The skill's state

A state $\boldsymbol{S}_j$ is defined by

- a specific set of constraints $\boldsymbol{C}_s$, and their set of parameters $\boldsymbol{\gamma}_i$
- the local weights for the constraints w_i and the local priorities p_i,
- a choice for the process models for the uncertainties and measurement equation for the estimation configuration.

The skill's state action

The action in a state $\boldsymbol{S}_j$ can only make continuous changes within its discrete configuration. The possibly available continuous parameters in a state are

- the constraint equation parameters $\boldsymbol{\gamma_i}$, for instance to define a continuous changing trajectory for a desired output value;

- the constraints local weights w_i, for instance to phase in or phase out the action of a constraint; and
- any continuous parameters in the process model or measurement equations of the estimation problem, for instance to include the results of a learning algorithm in an online fashion.

The skill's events

An event in a skill can be triggered in different ways:

- by a condition on the value of a continuously changing variable in a configured state, for instance the value of an estimated uncertainty, the value of a constraint output; and
- by a discrete signal coming from the above task controller, which triggers a desired change in the motion specification.

The former is an event triggered by the running state itself, the latter is triggered by the "external" task controller.

The skill's transition from one state to another

Only a state transition allows a discrete change in the motion specification and it can only be activated by a specified event.

If a smooth transition is desired between the constraint configuration of two different states an intermediate state can automatically be created to phase in and phase out the different constraints. This intermediate *transition state* includes the constraint configuration of the old and the new state and defines a transition parameter α. This parameter α is used to adapt the weights of the constraints as such that the old state's constraint configuration is phased out, while the new state's constraint configuration is phased in:

$$\begin{aligned} \boldsymbol{w}_{\text{old transition}} &= (1-\alpha)\boldsymbol{w}_{\text{old}}, \\ \boldsymbol{w}_{\text{new transition}} &= \alpha\boldsymbol{w}_{\text{new}}, \end{aligned} \tag{5.2}$$

with α monotonically changing from 0 to 1, $\boldsymbol{w}_{\text{old}}$ the collection of all constraint weights of the old state, and $\boldsymbol{w}_{\text{new}}$ the collection of all constraint weights of the new state. The rate of change is defined in the transition state.

From a software point of view, FSMs form the bridge between, on the one hand, the "sub-symbolic" continuous time world of the motion specification between

robot and environment, and, on the other hand, the "symbolic" discrete knowledge of execution coordination, configuration, synchronization, etc. The names of the states, events and transitions form the symbolic information, which is interpreted at the task controller level.

Each skill will have a *nominal* sequence of instantaneous motions, but a possibly large set of *alternative* sequences. However, the FSM model must be very dynamically *reconfigurable* online, since, when having switched to a non-nominal branch of the FSM, that new branch could receive the "nominal" status itself. This behavior is the learning that happens at the task controller level.

5.4.4 Formal representation

Finite state machines can be formally represented using state charts (Harel 1987). More recently FSM can also be represented by the Unified Modeling Language (UML) (OMG 2009), which is a standardized general-purpose modeling language in the field of software engineering. The standard is managed, and was created by, the Object Management Group. UML includes a set of graphical notation techniques to create visual models of software-intensive systems. UML has very rich semantics and notations for describing state machines. UML state machines overcome the limitations of traditional finite state machines while retaining their main benefits. UML state machines introduce new concepts like hierarchically nested states, which can be used to create multiple layers of skills. UML state descriptions can be represented using the UML diagrams.

5.5 Example application: Human-aware force-controlled manipulation task using two robots

The following sections describe the different skills which are active in the example application introduced in Chapter 3. In each section Step 2 and Step 3 of Sect. 5.2 for the specific skill will be described using the formalised interface from Sect. 5.3 while the skill coordination is described using the proposed methodology from Sect. 5.4.

5.5.1 Tracking of an object's unknown contour

This skill configures and coordinates the force-controlled motion of a probe in contact with the unknown contour of an object.

Constraint configuration

The constraint c_1 is defined as follows:

- `name`: unknown contour tracking,
- $\boldsymbol{f}$:

$$\boldsymbol{y}_1 = \boldsymbol{\chi_f}_1, \tag{5.3}$$

- $\boldsymbol{K}$: at the velocity level:

$$\dot{\boldsymbol{y}}_1 = \dot{\boldsymbol{y}}_{d1} + \boldsymbol{K}_p(\boldsymbol{y}_{d1} - \boldsymbol{y}_1), \text{ and} \tag{5.4}$$

- $\boldsymbol{\gamma}_1$: the desired feedforward velocity $\dot{\boldsymbol{y}}_{d1}$, the desired value $\boldsymbol{y}_{d1}$ and the position feedback constant $\boldsymbol{K}_p$.

Estimation configuration

The force sensor values are used to estimate the values of $\boldsymbol{\chi_u}_1$. The measurement equation $\boldsymbol{h}_1$ is defined as:

$$\boldsymbol{h}_1 : \begin{bmatrix} z_1 \\ z_2 \end{bmatrix} = \begin{bmatrix} K_{stiff} & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} y \\ \theta \end{bmatrix}, \tag{5.5}$$

with $z_1 = f_n$ the measured normal contact force in the plane of the contour, K_{stiff} the modelled normal contact stiffness, and z_2 the orientation of the tangent obtained by the force measurement: $z_2 = \texttt{atan2}(f_n, f_t)$, with f_t the measured tangent contact force in the plane of the contour.

The process model $\boldsymbol{P}_{uncert1}$ for $\boldsymbol{\chi_u}_1$ is defined as:

$$\frac{d}{dt} \begin{pmatrix} y_u \\ \theta_u \\ \dot{\theta}_u \end{pmatrix} = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} y_u \\ \theta_u \\ \dot{\theta}_u \end{pmatrix}, \tag{5.6}$$

in discretized form this becomes:

$$\begin{pmatrix} y_u^k \\ \theta_u^k \\ \dot{\theta}_u^k \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & \Delta t \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} y_u^{k-1} \\ \theta_u^{k-1} \\ \dot{\theta}_u^{k-1} \end{pmatrix}, \tag{5.7}$$

with a^i the value of variable a at timestep i and Δt the discrete time step . The second order model for the tangent orientation uncertainty helps the skill to take the effect of the curvature of the contour into account.

Coordination

The FSM for this skill contains the following states which all contain the constraint c_1: for all states $\boldsymbol{S}_i$: $\boldsymbol{C}_s = c_1$ and $\boldsymbol{w}_1 = \boldsymbol{1}$.

- *No contact*: in this state no contact between the probe and the object's contour exists. A constant feedback law is used to keep the position relation between the probe and the modeled contact point fixed, the probe is aligned with the contour while being kept at a safe distance:

$$\boldsymbol{\gamma}_1 : \dot{\boldsymbol{y}}_d = \boldsymbol{0}\,,\ \boldsymbol{y}_d = [0, 0, y_{safe}, 0, 0, 0]^T \text{ and } \boldsymbol{K}_p = \text{diag}(\boldsymbol{k}_p), \tag{5.8}$$

  with y_{safe} a safe normal distance between the probe and the contour and $\boldsymbol{k}_p$ six position feedback constants. The estimation configuration is inactive since there are no contact forces yet.

- *Approach*: the action in this state reduces the distance between the probe and the object's contour using a constant velocity along the predicted contour normal:

$$\boldsymbol{\gamma}_1 : \dot{\boldsymbol{y}}_d = [0, 0, v_{approach}, 0, 0, 0]^T\,,\ \boldsymbol{y}_d = \boldsymbol{0} \text{ and } \boldsymbol{K}_p = \text{diag}(\boldsymbol{k}_p), \tag{5.9}$$

  with $\boldsymbol{k}_p(3) = 0.0$. The estimation configuration is inactive since there are no contact forces yet.

- *Track*: in this state the contact is being actively controlled:

$$\boldsymbol{\gamma}_1 : \dot{\boldsymbol{y}}_d = [0, v_{track}, 0, 0, 0, 0]^T\,,\ \boldsymbol{y}_d = \boldsymbol{0} \text{ and } \boldsymbol{K}_p = \text{diag}(\boldsymbol{k}_p), \tag{5.10}$$

  with v_{track} the desired tracking velocity and $\boldsymbol{k}_p(2) = 0$. The estimation configuration is activated with an initial estimate for $\boldsymbol{\chi_u}_1 = \boldsymbol{0}$ at the first contact. v_{track} is determined proportionally with the current confidence in the estimation of $\dot{\boldsymbol{\chi}}_{\boldsymbol{u}1}$, which is indicated by the covariances of the estimated parameters. This means that if the tangent and curvature are well estimated the tracking velocity is increased, if the estimation is not yet converged the tracking is slowed down.

- *Clear*: in this state the contact is deliberatly broken using a constant feedback law and the probe and object are brought in a safe state:

$$\boldsymbol{\gamma}_1 : \dot{\boldsymbol{y}}_d = \boldsymbol{0}\ ,\ \boldsymbol{y}_d = [0, 0, y_{safe}, 0, 0, 0]^T \text{ and } \boldsymbol{K}_p = \text{diag}(\boldsymbol{k}_p), \tag{5.11}$$

  with y_{safe} a safe normal distance between the probe and the contour. The estimation configuration is deactivated.

The following events and related transitions are used in the FSM:

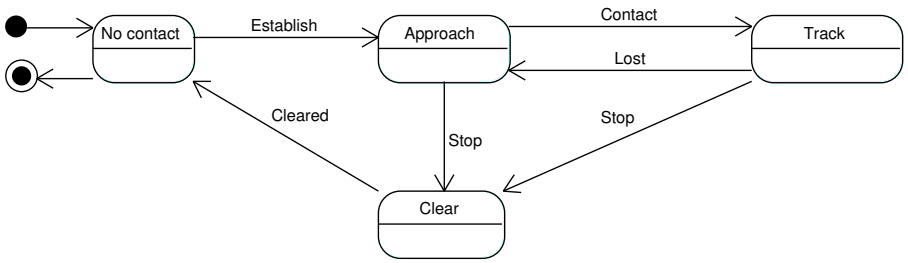

Figure 5.3: A UML state machine representation for the unknown object's contour following skill.

- *Establish*: this event is fired from the "higher" level task controller. The transition from the *No contact* state to the *Approach* state is activated.
- *Contact*: this event is fired by the *Approach* state when a contact is established, this is indicated by the force measurement. When the event is fired the skill stores an initial prediction for $\boldsymbol{T}_{front}^{contact}$ and activates the transition to the *Track* state.
- *Lost*: this event is fired by the *Track* state, and it is triggered by the force measurement. The skill stores the $\boldsymbol{T}_{contact}^{probe}$ at the moment of contact loss, sets the arc length s back to zero and activates the transition to the *Approach* state.
- *Stop*: this event is fired from the "higher" level task controller and activates the transition to the *Clear* state.
- *Cleared*: this event is fired by the *Clear* state when the safety condition is met ($y = y_{safe}$) and activates the transition to the *No contact* state.

Figure 5.3 shows a graphical UML state machine representation of the unknown object's contour tracking problem.

5.5.2 Estimating the orientation and height of a plane

This skill configures and coordinates the estimation procedure for the orientation and height of the front plane of the object using a laser distance measurement.

Estimation configuration

The measurement equation $\boldsymbol{h}_2$ is defined as:

$$\boldsymbol{h}_2 : z_1 = z, \tag{5.12}$$

with z_1 the laser distance measurement.

The process model $\boldsymbol{P}_{uncert2}$ for the uncertainty coordinates $\boldsymbol{\chi_{u}}_{front}$ in discretized form is defined as:

$$\begin{pmatrix} h_u^k \\ \phi_u^k \\ \psi_u^k \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} h_u^{k-1} \\ \phi_u^{k-1} \\ \psi_u^{k-1} \end{pmatrix}, \tag{5.13}$$

with h_u the unknown normal distance and ϕ_u and ψ_u the unknown orientation of the normal of the front plane with respect to the $\{object\}$ frame.

Coordination

The FSM of the skill contains the following states:

- *Active*: in this state the laser distance measurement is assumed to be reliable, this is the case when the laser distance sensor is actually pointing on the object's frontal plane. The estimation is activated.
- *Inactive*: in this state the laser distance measurement is assumed to be unreliable, because the laser distance sensor is not pointing on the object's frontal plane anymore. Therefore the estimation is deactivated.

The following events and transitions can occur:

- *Leave*: this happens when the laser distance sensor leaves the object's frontal plane and can be detected by an abrupt increase of the distance measurement.
- *Enter*: this happens when the laser distance sensor enters the object's frontal plane and can be detected by an abrupt decrease of the distance measurement.

It is the job of the "higher" level coordination to inform the skill in which of both states it has to start. The skill itself cannot know this when the application is started.

Figure 5.4 shows the UML state machine representation for this skill.

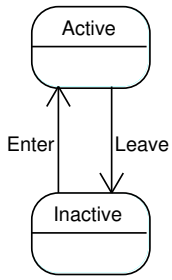

Figure 5.4: A UML state machine representation for the plane estimating skill.

5.5.3 Pointing a camera to a person.

This skill configures and coordinates the motion of the camera to point it towards a person, taking into account possible discrete switches in the desired person to point to.

Constraint configuration

Only three auxiliary coordinates are constrained, x, y and θ, the constraint c_3 is defined as follows:

- `name`: pointing,
- $\boldsymbol{f}$:

$$\boldsymbol{y}_3 = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix} \boldsymbol{\chi}_{\boldsymbol{f}3}, \tag{5.14}$$

- $\boldsymbol{K}$: at the velocity level:

$$\dot{\boldsymbol{y}}_3 = \dot{\boldsymbol{y}}_{d3} + \boldsymbol{K}_p(\boldsymbol{y}_{d3} - \boldsymbol{y}_3), \text{ and} \tag{5.15}$$

- $\boldsymbol{\gamma}_3$: the desired feedforward velocity $\dot{\boldsymbol{y}}_{d3}$, the desired value $\boldsymbol{y}_{d3}$ and the position feedback constant $\boldsymbol{K}_p$.

Coordination

The FSM of this skill contains the following states:

- *Inactive*: in this state no action is defined. $\boldsymbol{C}_s = \varnothing$.
- *Track*: $\boldsymbol{C}_s = c_3$ and $\boldsymbol{w}_3 = \boldsymbol{1}$. The state's action tries to keep the person straight up and in the center of the camera image:

$$\boldsymbol{\gamma}_3 : \dot{\boldsymbol{y}}_d = \boldsymbol{0}, \boldsymbol{y}_d = \boldsymbol{0} \text{ and } \boldsymbol{K}_p = \text{diag}(\boldsymbol{k}_p), \tag{5.16}$$

  with $\boldsymbol{k}_p$ three position feedback constants.
- *Switch*: $\boldsymbol{C}_s = c_3$ and $\boldsymbol{w}_3 = \boldsymbol{1}$ and the camera has to switch to another person. The action defines a trajectory in constraint space between both persons:

$$\boldsymbol{\gamma}_3 : \dot{\boldsymbol{y}}_d = \dot{\boldsymbol{y}}_{d\ \text{trajectory}}, \boldsymbol{y}_d = \boldsymbol{y}_{d\ \text{trajectory}} \text{ and } \boldsymbol{K}_p = \text{diag}(\boldsymbol{k}_p^{\text{switch}}), \tag{5.17}$$

  with $\dot{\boldsymbol{y}}_{d\ \text{trajectory}}$ and $\boldsymbol{y}_{d\ \text{trajectory}}$ determined by a trapezoidal velocity profile and the values of $\boldsymbol{y}_3$ for the new person to point to. The feedback constants in $\boldsymbol{k}_p^{\text{switch}}$ are lower than the ones in the *Track* state.

The following events and transitions are defined.

- *Start*: this event is fired by the "higher" level task controller, and activates the transition to the *Switch* state.
- *Switch*: this event is fired by the detection of an abrupt change of $\boldsymbol{y}_3$ which causes $\boldsymbol{y}_3 > \boldsymbol{y}_3^{track}$, this is due to a change in the person to point the camera to. $\boldsymbol{y}_3^{track}$ indicates the stability boundary in which the configured controller used in the *Track* state is considered safe. The event activates the transition to the *Switch* state.
- *Track*: this event is fired by the *Switch* state when the switching trajectory is finished and $\boldsymbol{y}_3 < \boldsymbol{y}_3^{track}$. It activates the transition to the *Track* state.
- *Stop*: this event is fired by the "higher" level task controller and activates the transition to the *Inactive* state.

Figure 5.5 shows the UML state machine of this skill.

5.5.4 Keep a safe distance from a moving person.

This skill configures and coordinates the constraint on the distance between the closest moving person and the force controlled manipulation task.

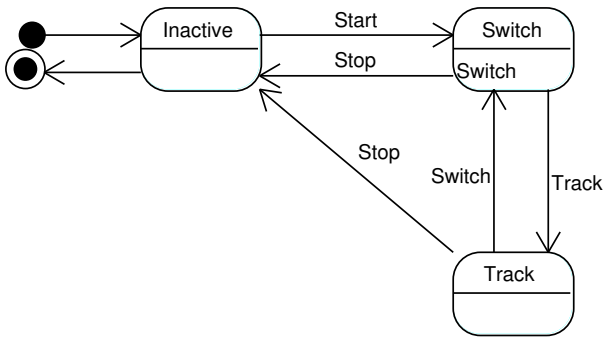

Figure 5.5: The UML state machine off the camera pointing skill.

Constraint configuration

Only one auxiliary coordinate x, which represents the distance between the $\{contact\}$ frame and $\{person\}$ frame projected in the ground plane of the world, is constrained. c_4 is defined as follows:

- `name`: keep distance,
- $\boldsymbol{f}$:

$$y_4 = [0, 1, 0, 0, 0, 0]\boldsymbol{\chi}_{\boldsymbol{f}4}, \tag{5.18}$$

- $\boldsymbol{K}$: at the velocity level:

$$\dot{y}_4 = k_p(y_{d4} - y_4), \text{ and} \tag{5.19}$$

- $\boldsymbol{\gamma}_4$: the desired value y_{d4} and the position feedback constant k_p.

Coordination

The FSM of this skill contains the following states:

- *Active*: in this state the distance constraint is active $\boldsymbol{C}_s = c_4$. The action defines the weight value for the constraint as $w_4 = \left(\frac{1}{y_4^2} - \frac{1}{b^2}\right)^2$ and $y_{d4} = y_{goal}$, with b the safety boundary which activates this state and y_{goal} a desired value for the distance with $y_{goal} > b$. The resulting weight goes to infinity if the distance approaches zero.

- *Inactive*: in this state the distance constraint is deactivated: $\boldsymbol{C}_s = \varnothing$.

The following events and transitions are defined:

- *Enter*: this event is triggered when the closest person crosses a safety boundary b, $y_4 < b$ and enters the workspace of the robots. The event activates the transition from the *Inactive* to the *Active* state.
- *Leave*: this event is triggered when the closest person crosses the safety boundary b, $y_4 > b$ and leaves the workspace of the robots and activates the transition from the *Active* to the *Inactive* state.

The UML representation is similar to the one of the plane estimating skill in Figure 5.4.

5.5.5 Multi-target tracking and localization.

The skills from Sects. 5.5.3 and 5.5.4 both need the estimated position of the closest person. This estimation is based on the 2D range laser scanner. This skill will give an estimate for the remaining uncertainty in $\boldsymbol{T}_w^{person}$. The estimation itself is not included in the iTaSC motion specification because it is far more complicated due to the fact that there are multiple targets. An external estimation is used based on (De Laet et al. 2010). The external estimation provides an estimate for $\boldsymbol{\chi}_{\boldsymbol{u}}$ and $\dot{\boldsymbol{\chi}}_{\boldsymbol{u}}$ modeled in the virtual manipulator of this skill. It will also trigger an event when another person becomes the "closest" person.

More details of this example application can be found in (Smits et al. 2008). A video of this application can be found on
`http://people.mech.kuleuven.be/~orocos`.

5.5.6 Keep start configuration

This skill will try to keep the robot manipulators in their position at first contact. No spatial relations are introduced.

Constraint configuration

All the robot joints are included in this constraint. c_6 is defined as follows:

- `name`: keep configuration,

- $\boldsymbol{f}$:

$$\boldsymbol{y}_6 = [\boldsymbol{q}_1^T, \boldsymbol{q}_2^T]^T, \tag{5.20}$$

- $\boldsymbol{K}$: at the velocity level:

$$\dot{\boldsymbol{y}}_6 = \boldsymbol{k}_p(\boldsymbol{y}_{d6} - \boldsymbol{y}_6), \text{ and} \tag{5.21}$$

- $\boldsymbol{\gamma}_6$: the desired value $\boldsymbol{y}_{d6}$ and the position feedback constants $\boldsymbol{k}_p$.

Coordination

The FSM of this skill contains the following states:

- *Active*: in this state the constraint is active $\boldsymbol{C}_s = c_6$, $\boldsymbol{w}_6 = \boldsymbol{1}$. The desired values are set as $\boldsymbol{y}_{d6} = \boldsymbol{y}_{dcontact}$.
- *Inactive*: in this state the distance constraint is deactivated: $\boldsymbol{C}_s = \varnothing$.

The following events and transitions are defined:

- *Activate*: this event is triggered when by the first contact between the probe and the contour. The event activates the transition from the *Inactive* to the *Active* state.
- *Deactivate*: this event is triggered by the "higher" level task controller and activates the transition from the *Active* to the *Inactive* state.

The UML representation is similar to the one of the plane estimating skill in Figure 5.4.

5.6 Task controller

The task controller is responsible for scheduling the different skills by sending to each skill the appropriate event at the appropriate time:

- Activate the *Approach* state of the unknown contour tracking skill by sending the *Establish* event.
- Activate the *Clear* state of the unknown contour tracking skill by sending the *Stop* event.

- Activate the *Switch* state of the camera pointing skill by sending the *Start* event.
- Activate the *Inactive* state of the camera pointing skill by sending the *Stop* event.
- Activate the *Inactive* state of the keep start configuration skill by sending the *Deactivate* event.

The task controller also configures the optimization configuration. In this example the prioritization strategy is chosen: the keep start configuration skill is given the lowest priority, all the other skills are put into the highest priority.

5.7 Conclusion

This chapter proposed a modular control architecture to enable the execution of multi-sensor, multi-robot, complex robot applications. For the next generation of robotic systems this new way of implementing robot applications will be mandatory to enable the necessary abstractions for the different levels of developers and users. The chapter presented a multi-layered control structure with a constraint-based instantaneous motion control and specification at the basis, a state-machine skill level to put coordination on top of the constraint control and the uncertainty estimation and a task level at the top, where the necessary monitoring and decision making has to be implemented. The power of the framework was illustrated with a human aware force-controlled manipulation task using two robots.

Chapter 6

Linear-time solver for acceleration-based constraints

Acceleration is finite, I think according to some laws of physics.

Terry Riley (1935 –)

6.1 Introduction

De Schutter et al. 2007 explain how the instantaneous motion specification, iTaSC can be solved for velocity-based robotic systems. The extension of this methodology for acceleration-based control can be found in (De Laet and De Schutter 2007) and is explained in detail in Section 3.4.2. These methodologies however require the weighted inversion of a non-square matrix of which the size depends on the total number of constraints and the total number of joints in the robotic system. Inverting a non-square matrix

is a computationally expensive "order n^3" calculation, especially for highly redundant systems like humanoids. By using the *Joint-Space Inertia Matrix* as a weighting matrix in joint space (Bruyninckx and Khatib 2000) during the inversion of the "task jacobian" a "physically consistent" motion for a redundant robotic system is obtained. De Sapio et al. 2006 also present a task-level control methodology for constrained multi-body systems, the authors discuss the symmetry between the constrained dynamics and the operational space control. Their solution however requires the inversion of jacobian matrices covering the entire kinematic structure, resulting in a methodology that is very time consuming and hard to use in a hard realtime setting. The demand for motion solvers that can be used in the hard realtime control loop of robotic systems drives the search for an "order n" algorithm.

Dynamic algorithms A *hybrid dynamics* algorithm solves the combined forward/inverse dynamics problem (Featherstone 2008), which is defined as follows: given some joint motions *and* forces, some end-effector motions *and* forces, and 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* of humanoid robots, since such tasks typically require specifications of motion and force on the end effectors, while the robot also wants to keep its posture close to its most "comfortable" configuration by adding internal joint motions.

The literature on robot dynamics is extremely rich, from the general dynamics work by Newton 1871, Euler 1776, Lagrange 1867, Gauss 1829, and Appell 1900, over modern *multi-body dynamics* (Gantmacher 1975; Roberson and Schwertassek 1988; Wittenburg 1977), to robot-specific dynamics, first as "order n^3" algorithms (Hooker and Margulies 1965; Lilov and Wittenburg 1977; Roberson and Wittenburg 1966; Luh et al. 1980), and later as "order n" algorithms (Featherstone 1983; Vereshchagin 1974), where n is the number of degrees of freedom in the kinematic chain of the robot. The contributions of this chapter are: (i) to extend the groundbreaking work by roboticists of the Bauman Institute in Moscow in the 70s and 80s, about linear-time dynamics algorithms for redundant *serial* robots with (partial) acceleration equality constraints (Popov et al. 1978; Vereshchagin 1974; Vereshchagin 1989) to *tree-structured* kinematic chains such that it becomes useful for humanoid robots and mobile manipulators; (ii) to show how this work can be used together with the constraint-based task specification framework, iTaSC from De Schutter et al. 2007 and (iii) to compare this work to the better-known priority-based inverse dynamics with (full) acceleration constraints on the end effectors, e.g., (Khatib et al. 2004; De Sapio et al. 2006; Khatib et al. 2008; Nakamura 1991).

Tree-structured robots are often *redundant* for their tasks, and need motion specification for *multiple* end effectors. This chapter only discusses *inertia-*

weighting as redundancy resolution criterion (Khatib 1987), and accepts a specification that consists of *acceleration constraints* on the end effectors. These constraints can be *partial*, i.e., allowing one or more degrees of freedom at an end effector to be left unspecified. (This chapter, as well as all cited papers, assume that the constraints on the various end effectors are *independent*, that is, no (virtual or real) *kinematic loops* exist.)

Overview of the chapter. Section 6.2 summarizes the well-known outward and inward recursions of linear-time dynamics algorithms. Section 6.4 includes acceleration equality constraints on the end effector and extends Vereshchagin 1989's hybrid dynamics algorithm from serial to tree-structured chains, Section 6.6 shows how the proposed algorithm can be used together with the constraint-based task specification framework from (De Schutter et al. 2007) and Section 6.7 makes a comparison with the better-known priority-based approach with full acceleration constraints.

6.2 Segment-to-segment recursions

Figure 6.1 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 *root* segment) frame $\{d_i\}$ of segment i and the *proximal* frame $\{p_{i+1}\}$ of segment $i+1$; (ii) these frames $\{d_i\}$ and $\{p_{i+1}\}$ coincide when joint $i+1$ is in its *zero position*. Without much loss of generality, each joint is considered to have one degree of freedom, a reference frame with its origin in the (ideal) joint axis, and with the unit vector of the Z axis along the motion degree of freedom that the joint allows. The "base" frame gets the index "0", and, for a chain with n joints, the *end effector* frame has index "$n+1$." The vector $\boldsymbol{Z}_i$ represents the unit twist vector of the ith joint. The position vector $\boldsymbol{r}^{i,j}$ connects the origin of segment frame $\{i\}$ to the origin of segment frame $\{j\}$. q, $\dot{q}$ and $\ddot{q}$ are respectively, the position, velocity and acceleration value of joint.

This section contains no new material, but summarizes the well-known *numerical solvers* for kinematic chains of generic types, for the sake of defining notations and terminology. Those solvers have *linear-time* computational complexity, and use an *outward* segment-to-segment recursion to propagate position, velocity and acceleration from the root to the leafs (Section 6.2.1) and an *inward* segment-to-segment recursion for force and inertia (Section 6.2.2). All physical properties are expressed with respect to the *reference point* at the origin of the proximal frame $\{p_i\}$ of the segment over which the recursion runs.

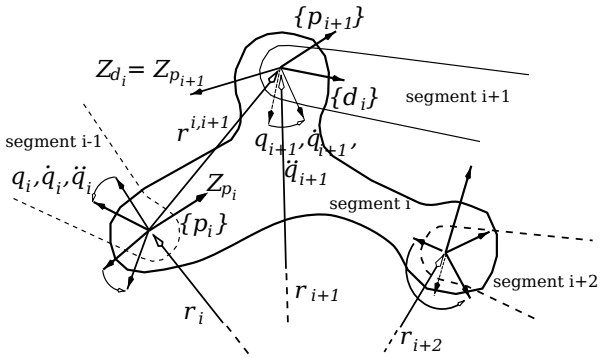

Figure 6.1: Reference frames and notation for segments in a tree-structured kinematic chain. "p_i" is the "proximal" frame of segment i, and "d_i" is the segment's distal frame. The frames' $\boldsymbol{Z}_i$ and $\boldsymbol{Z}_{i+1}$ axes lie along the (single degree of freedom) axes of the joints connected to the segment.

6.2.1 Outward position, velocity and acceleration recursions

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

1. a *change in reference point* from the proximal frame $\{p_i\}$ of segment i to this segment's distal frame $\{d_i\}$. This does not change any physical properties but involves only a *coordinate* transformation of the *vector representation* of the physical entity being transformed (pose, velocity, acceleration), and the *matrix representation* of the inertia operator.
2. the incorporation of the *physical contribution* (position, velocity, acceleration) at joint $i+1$.

The physical contribution at joint i

The position and orientation of (the proximal frame on) segment $i+1$ with respect to (the proximal frame on) segment i is a simple function of the measured joint angle q_{i+1}:

$$\boldsymbol{T}_{p_i}^{p_{i+1}} = \boldsymbol{T}_{p_i}^{d_i} \boldsymbol{T}_{d_i}^{p_{i+1}}(q_{i+1}), \tag{6.1}$$

with $\boldsymbol{T}_{p_i}^{d_i}$ the homogeneous transformation matrix between the ith distal and proximal frames, which is constant for rigid body segments. The *branching* that occurs in a tree-structured chain at segments from which two or more

sub-chains emerge, does not complicate the motion recursion at all, since no interaction is assumed between two branches.

Without loss of generality, this chapter from now on only presents the operations on the *physical vectors*. The *coordinate representations* and *screw transformations* needed to allow the addition of velocity and force vectors are omitted in order to avoid notation overload.

The *velocity recursion* finds the linear and angular velocity $\dot{\boldsymbol{X}}_{i+1} = (\boldsymbol{\omega}_{i+1}^T, \boldsymbol{v}_{i+1}^T)^T$ of segment $i+1$ (with the origin of the proximal frame $\{p_{i+1}\}$ as velocity reference point), given the linear and angular velocity $\dot{\boldsymbol{X}}_i = (\boldsymbol{\omega}_i^T, \boldsymbol{v}_i^T)^T$ of segment i given the joint angle q_{i+1} and the joint angle velocity $\dot{q}_{i+1}$ between both segments:

$$\dot{\boldsymbol{X}}_{i+1} = \dot{\boldsymbol{X}}_i + \boldsymbol{Z}_{i+1}\dot{q}_{i+1} \tag{6.2}$$

with $\boldsymbol{Z}_{i+1}$ the unit twist of the joint.

The *acceleration recursion* calculates the linear and angular acceleration $\ddot{\boldsymbol{X}}_{i+1} = (\dot{\boldsymbol{\omega}}_{i+1}^T, \dot{\boldsymbol{v}}_{i+1}^T)^T$ given the linear and angular acceleration $\ddot{\boldsymbol{X}}_i = (\dot{\boldsymbol{\omega}}_i^T, \dot{\boldsymbol{v}}_i^T)^T$, the joint angle speed $\dot{q}_{i+1}$, and the joint acceleration $\ddot{q}_{i+1}$. The result is similar to the velocity recursion, except for the gyroscopic *bias acceleration* $\ddot{\boldsymbol{X}}_i^b = \dot{\boldsymbol{X}}_i \times (\boldsymbol{Z}_{i+1}\dot{q}_{i+1})$ of the moving frame $\{p_{i+1}\}$ due to the velocity $\dot{\boldsymbol{X}}_i$ of segment i:

$$\ddot{\boldsymbol{X}}_{i+1} = \ddot{\boldsymbol{X}}_i + \boldsymbol{Z}_{i+1}\ddot{q}_{i+1} + \ddot{\boldsymbol{X}}_i^b \tag{6.3}$$

This uses the property that $\boldsymbol{V}_i \times \boldsymbol{V}_j$ (spatial cross product) is the time derivative of a spatial motion vector $\boldsymbol{V}_j$ that is fixed to a body that moves with a velocity $\boldsymbol{V}_i$.

6.2.2 Inward force and inertia recursions

The *kinematic* acceleration recursions in the paragraphs above deal with the calculation of the acceleration of segment $i+1$ if the position, 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 6D vector pair $\boldsymbol{F}^T = (\boldsymbol{m}^T, \boldsymbol{f}^T)$, applied to the segments.

The relationship between the acceleration $\ddot{\boldsymbol{X}}_i$ and the force $\boldsymbol{F}_i$ of segment i for an unconstrained segment i is given by the segment's inertia matrix $\boldsymbol{M}_i$. However, if segment $i+1$ is connected to segment i by means of an ideal, *non-actuated* joint, the force $\boldsymbol{F}_i$ is not completely available to accelerate segment i

because it also has to accelerate segment $i+1$, through the connecting joint constraint.

The segment acceleration $\ddot{\boldsymbol{X}}_{i+1}$ transmitted through the joint from proximal to distal segment, and caused by the segment acceleration $\ddot{\boldsymbol{X}}_i$, is given by Eq. (6.4)

$$\ddot{\boldsymbol{X}}_{i+1} = \left(\mathbf{1} - \boldsymbol{Z}_{i+1}\left(\boldsymbol{Z}_{i+1}^T\boldsymbol{M}_{i+1}\boldsymbol{Z}_{i+1}\right)^{-1}\boldsymbol{Z}_{i+1}^T\boldsymbol{M}_{i+1}\right)\ddot{\boldsymbol{X}}_i, \tag{6.4}$$

$$= \boldsymbol{P}_{i+1}^T\ddot{\boldsymbol{X}}_i. \tag{6.5}$$

with

$$\boldsymbol{P}_{i+1} = \mathbf{1} - \boldsymbol{M}_{i+1}\boldsymbol{Z}_{i+1}\left(\boldsymbol{Z}_{i+1}^T\boldsymbol{M}_{i+1}\boldsymbol{Z}_{i+1}\right)^{-1}\boldsymbol{Z}_{i+1}^T. \tag{6.6}$$

See (Featherstone 2008, Chapter 7) for more details. The force $\boldsymbol{F}_i$ needed to accelerate segment i by an amount $\ddot{\boldsymbol{X}}_i$ is the sum of (i) the inertial force to accelerate segment i, and (ii) the force transmitted through the joint to accelerate the next segment $i+1$, and can be written as:

$$\boldsymbol{F}_i = \boldsymbol{M}_i^a\ddot{\boldsymbol{X}}_i, \tag{6.7}$$

$$\text{with} \quad \boldsymbol{M}_i^a = \boldsymbol{M}_i + \boldsymbol{P}_{i+1}\boldsymbol{M}_{i+1}. \tag{6.8}$$

$\boldsymbol{M}_i^a$ is the so-called *articulated body inertia* (Featherstone 2008, Chapter 7). The part $\boldsymbol{F}_i$ of the force $\boldsymbol{F}_{i+1}$ working on segment $i+1$ and transmitted in inward direction to segment i is:

$$\boldsymbol{F}_i = \boldsymbol{F}_{i+1} - \boldsymbol{M}_{i+1}\boldsymbol{Z}_{i+1}(\boldsymbol{Z}_{i+1}^T\boldsymbol{M}_{i+1}\boldsymbol{Z}_{i+1})^{-1}\boldsymbol{Z}_{i+1}^T\boldsymbol{F}_{i+1}, \tag{6.9}$$

$$= \boldsymbol{P}_{i+1}\boldsymbol{F}_{i+1}. \tag{6.10}$$

So, in matrix form, the total inward recursion $\boldsymbol{F}_i \leftarrow \boldsymbol{F}_{i+1}$ becomes:

$$\boldsymbol{F}_i = \boldsymbol{P}_{i+1}\boldsymbol{F}_{i+1} - \boldsymbol{M}_{i+1}^a\boldsymbol{Z}_{i+1}(\boldsymbol{Z}_{i+1}^T\boldsymbol{M}_{i+1}^a\boldsymbol{Z}_{i+1})^{-1}\tau_{i+1} + \boldsymbol{F}_i^b + \boldsymbol{F}_i^e, \tag{6.11}$$

with $\boldsymbol{F}_i^b = \boldsymbol{M}_i^a\ddot{\boldsymbol{X}}_i^b$ the velocity-dependent *bias force* generated by the angular velocity and the mass properties of segment i, and $\boldsymbol{F}_i^e$ the *external force*, i.e., the resultant of all forces applied to segment i, and τ_i the applied joint torque at joint i.

6.2.3 Tree-structured chains

The outward recursion requires no changes with respect to serial chains, since every segment and joint is only on one serial path from the root of the tree.

As for the inward recursion when segment i is a branching node in a tree, Eq. (6.7) must be changed to contain the *sum* over all inertias $\boldsymbol{M}_{i+1}^a$ of the distal segments connected to segment i, and Eq. (6.11) contains the *sum* over all $\boldsymbol{F}_{i+1}$ of the segments connected to segment i, plus the joint torque contributions of all joints at the distal frames $i+1$. See (Featherstone 2008) for more details.

6.3 Dynamics algorithms

The relationship between the force τ_i delivered at a joint and the acceleration $\ddot{q}_i$ of that joint is given by (Featherstone 2008, Chapter 7):

$$\tau_i = (\boldsymbol{Z}_i^T \boldsymbol{M}_i^a \boldsymbol{Z}_i)\ddot{q}_i + \boldsymbol{Z}_i^T(\boldsymbol{F}_i + \boldsymbol{M}_i^a \ddot{\boldsymbol{X}}_{i-1}). \tag{6.12}$$

The factor in front of the joint acceleration $\ddot{q}_i$ is the component on the joint axis, $\boldsymbol{Z}_i$ of the articulated body inertia $\boldsymbol{M}_i^a$ of all segments more distal than i. The joint also feels (the component on the joint axis of) the forces $\boldsymbol{F}_i$ (6.11) from the distal segments and the inertia force $\boldsymbol{M}_i^a \ddot{\boldsymbol{X}}_{i-1}$ of its own segment's motion, caused by the more proximal joints. For the forward dynamics, the joint torques τ_i are given, and the acceleration at joint i follows straightforwardly from Eq. (6.12):

$$\ddot{q}_i = (\boldsymbol{Z}_i^T \boldsymbol{M}_i^a \boldsymbol{Z}_i)^{-1} \left\{ \tau_i - \boldsymbol{Z}_i^T \left(\boldsymbol{F}_i + \boldsymbol{M}_i^a \ddot{\boldsymbol{X}}_i \right) \right\}. \tag{6.13}$$

For the inverse dynamics, the joint accelerations $\ddot{q}_i$ are calculated from the desired acceleration of the end effectors, Section 6.4. Gravity is taken into account by initializing the recursion with the gravitational acceleration: $\ddot{\boldsymbol{X}}_0 = \boldsymbol{g}$. Once $\ddot{q}_i$ is known, the total acceleration of the ith segment can be found:

$$\ddot{\boldsymbol{X}}_i = \ddot{\boldsymbol{X}}_{i-1} + \ddot{q}_i \boldsymbol{Z}_i + \ddot{\boldsymbol{X}}_i^b, \tag{6.14}$$

with $\ddot{\boldsymbol{X}}_i^b$ the *bias acceleration* due to the non-vanishing angular velocities of the proximal segment. Now, the joint acceleration calculation of the next joint $\ddot{q}_{i+1}$ can be calculated, etc.

6.4 Acceleration constraints—Gauss principle of least constraints approach

This section explains how to solve for $\ddot{q}$ when *acceleration (equality) constraints* are added to some segments on a kinematic chain using the Gauss principle of least constraints. Figure 6.2 shows an overview of the input and output of the

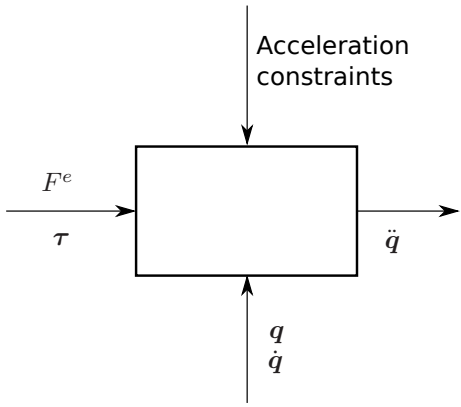

Figure 6.2: Overview of the input and output of the presented hybrid dynamic algorithm.

algorithm. The methodology introduced by this section is based on the work of Vereshchagin 1974 and Vereshchagin 1975. 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 problem can be solved in a recursive way (Popov et al. 1978; Vereshchagin 1989).

6.4.1 End effector constraint in serial chain

This section does not add new knowledge and is entirely based on the work of Vereshchagin 1989. First, assume only the *end effector* segment, "N", in a *serial* chain is not allowed to accelerate arbitrarily, but must satisfy the following (linear) constraints:

$$\boldsymbol{A}_N^T \ddot{\boldsymbol{X}}_N = \boldsymbol{b}_N. \tag{6.15}$$

If there are m constraints, $\boldsymbol{A}_N$ is a $6 \times m$ matrix, and each of its columns can be interpreted as a "unit" *constraint force* acting on the segment. The right-hand side m-dimensional vector $\boldsymbol{b}_N$ represents the unit "*acceleration energy*", $\boldsymbol{\mathcal{Z}}$ generated in the constraints.[1] *Virtual* constraints can represent *desired accelerations* (partially) imposed on the segment by the human programmer; for example:

[1] Classical mechanics knows this product of a force with an acceleration already since the work by Gauss 1829, Hertz 1894, Gibbs 1879, and Appell 1900. Since "*Zwang*" is the German term (Hertz 1984; Saint Germain 1900) used in the early publications in the domain of constrained dynamics that appeared in the dominant German scientific literature of the early 20th century, the symbol $\boldsymbol{\mathcal{Z}}$ is commonly used in the international literature to denote acceleration energy.

- to only allow motion in the Z-direction of a reference *point* on the segment, the constraint matrices can be as follows:

$$\boldsymbol{A}_N = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 1 & 0 \\ 0 & 1 \\ 0 & 0 \end{bmatrix}, \quad \boldsymbol{b}_N = \begin{bmatrix} 0 \\ 0 \end{bmatrix}. \tag{6.16}$$

  The columns of $\boldsymbol{A}_N$ are "unit" constraint forces in the horizontal X and Y directions, that must keep the acceleration in those directions to zero.

- the specification above defines only the *constraints*, and not the *driver* for the motion. There are basically two ways of doing the latter: (i) by applying an *external force* (possibly derived from a potential field, such as gravity, or an artificial, task-specific equivalent), or (ii) by adding an *acceleration setpoint constraint*, as in this example:

$$\boldsymbol{A}_N = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \quad \boldsymbol{b}_N = \begin{bmatrix} 0 \\ 0 \\ x \end{bmatrix}. \tag{6.17}$$

  This adds a desired linear acceleration of x m/s^2 in the vertical direction to the specification above. Note that the physical units of $\boldsymbol{b}_N$ are *acceleration energy* Nm/s^2, and not acceleration. But since $\boldsymbol{b}_N$ represents a *unit* force, the numerical value of $\boldsymbol{b}_N$ can be interpreted as an acceleration with units m/s^2 or rad/s^2.

- to allow movement of the segment along Z without allowing rotations, the constraint matrices represent five constraints:

$$\boldsymbol{A}_N = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}, \quad \boldsymbol{b}_N = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}. \tag{6.18}$$

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

- to give the segment a desired acceleration $\ddot{\boldsymbol{X}}_N$, the constraint matrices can be as follows:

$$\boldsymbol{A}_N = \boldsymbol{1}_{6\times 6}, \quad \boldsymbol{b}_N = \ddot{\boldsymbol{X}}_N. \tag{6.19}$$

The m actual constraint forces are not known in advance, but will be found by the recursive dynamics algorithm below. More precisely, the *working directions* of all constraint forces are known (these are the "unit" matrices $\boldsymbol{A}_N$ from the examples above), but not their $m \times 1$ vector of *magnitudes* $\boldsymbol{\nu}$. So, the real constraint forces (at the end effector) will be $\boldsymbol{A}_N\boldsymbol{\nu}$. The introduction of segment constraints leads to the following extensions to the recursive formulas of Section 6.3:

- The ("unit") constraint forces in matrix $\boldsymbol{A}_N$ are propagated by the inward force recursion of Eq. (6.10):

$$\boldsymbol{A}_i = \boldsymbol{P}_{i+1}\boldsymbol{A}_{i+1}. \tag{6.20}$$

- During the inward recursion, we keep track of how much of the "constraint acceleration energy" $\boldsymbol{b}_N$ is already generated by the *(external and inertial) forces*, and by the *applied joint torques* (and, hence, do not need to be generated by the artificial constraint forces anymore):

$$\boldsymbol{\beta}_i = \boldsymbol{\beta}_{i+1} + \boldsymbol{A}_{i+1}^T \left\{ \ddot{\boldsymbol{X}}_{i+1}^b + \boldsymbol{Z}_{i+1} D_{i+1}^{-1} \left(\tau_{i+1} - \boldsymbol{Z}_{i+1}^T (\boldsymbol{F}_{i+1} + \boldsymbol{M}_{i+1}^a \ddot{\boldsymbol{X}}_{i+1}^b) \right) \right\}, \tag{6.21}$$

  with $D_{i+1} = \boldsymbol{Z}_{i+1}^T \boldsymbol{M}_{i+1}^a \boldsymbol{Z}_{i+1}$, $\boldsymbol{\beta}_i$ the constraint acceleration energy already generated at segment i and $\boldsymbol{\beta}_N = 0$.

  The terms in the curly braces represent all accelerations that result from inertial forces behind the joint, and from the force, τ_{i+1} applied to the joint.

- 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 $\boldsymbol{\mathcal{Z}}_i$:

$$\boldsymbol{\mathcal{Z}}_i = \boldsymbol{\mathcal{Z}}_{i+1} - \boldsymbol{A}_{i+1}^T \boldsymbol{Z}_{i+1} D_{i+1}^{-1} \boldsymbol{Z}_{i+1}^T \boldsymbol{A}_{i+1}, \quad \boldsymbol{\mathcal{Z}}_N = 0. \tag{6.22}$$

  The jth row of $\boldsymbol{\mathcal{Z}}_i$ 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* behind the current joint; the multiplication with $\boldsymbol{Z}_{i+1} D_{i+1}^{-1} \boldsymbol{Z}_{i+1}^T$ results in the *acceleration* generated at this joint by those constraint forces; and the multiplication with $\boldsymbol{A}_{i+1}^T$ yields the acceleration energy contributions at this joint.

- When the recursion arrives at the base ($i = 0$), one can solve for the still unknown constraint force magnitudes $\boldsymbol{\nu}$, using the final constraint acceleration energy balance:

$$\boldsymbol{\mathcal{Z}}_0 \boldsymbol{\nu} = \boldsymbol{b}_N - \boldsymbol{A}_0^T \ddot{\boldsymbol{X}}_0 - \boldsymbol{\beta}_0. \tag{6.23}$$

  $\boldsymbol{\nu}$ is then solved from Eq. (6.23):

$$\boldsymbol{\nu} = \boldsymbol{\mathcal{Z}}_0^{-1} \left(\boldsymbol{b}_N - \boldsymbol{A}_0^T \ddot{\boldsymbol{X}}_0 - \boldsymbol{\beta}_0 \right). \tag{6.24}$$

  For a robot with a rigidly fixed base, the acceleration $\ddot{\boldsymbol{X}}_0$ consists of gravity only. The numerical complexity of the solution of this set of linear equations is $\mathcal{O}(m^3)$, with m the number of constraints, because of the required inversion of $\boldsymbol{\mathcal{Z}}_0$.

  The matrix $\boldsymbol{\mathcal{Z}}_0$ is symmetric, as is apparent from Eq. (6.22). But, since each recursion in Eq. (6.22) 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 $\boldsymbol{\mathcal{Z}}_0$, and hence to generate the m constraint forces. Even then, $\boldsymbol{\mathcal{Z}}_0$ can be *singular* (that is, not of full rank, and hence not invertible), which is the mathematical indication that the kinematic chain is physically unable to generate the desired constraint Eq. (6.15). A *(weighted) pseudo-inverse* solution can then provide a set of joint forces that *approximates* the desired acceleration constraints; the weighting takes place in the space of the *constraint magnitudes* $\boldsymbol{\nu}$, and can be used to indicate that some constraints are more important than others. A priority-based inversion can also be used, this methodology uses the null space of high-priority constraints to make sure lower-priority constraints do not influence the high-priority constraints (Ben-Israel and Greville 1980).

- The outward joint acceleration remains similar to the unconstrained case of Eq. (6.13), except that the extra joint torque $\boldsymbol{A}_i \boldsymbol{\nu}$, from the constraint resolution is added to generate the final desired constraint forces:

$$\ddot{q}_i = (\boldsymbol{Z}_i^T \boldsymbol{M}_i^a \boldsymbol{Z}_i)^{-1} \left\{ \tau_i - \boldsymbol{Z}_i^T \left(\boldsymbol{F}_i + \boldsymbol{M}_i^a (\ddot{\boldsymbol{X}}_i^b + \ddot{\boldsymbol{X}}_{i-1}) + \boldsymbol{A}_i \boldsymbol{\nu} \right) \right\}. \tag{6.25}$$

Algorithm 6.1 shows pseudo-code for the above described methodology.

6.4.2 Null space

The *null space acceleration* of the kinematic chain corresponds to the joint accelerations that generate a zero Cartesian acceleration at the end effector,

Algorithm 6.1: Recursive hybrid constraint solver for a serial chain

// initial outward recursion
for $i = 1$ **to** NS **do**
// JntToCart is a function that directly calculates the required
// Cartesian space output from joint space input

$$[\boldsymbol{X}_J, \boldsymbol{Z}_i, \boldsymbol{M}_i] = \texttt{JntToCart}\left(\texttt{segments}(i), q_i\right)$$

$$\dot{\boldsymbol{X}}_i = \dot{\boldsymbol{X}}_{i-1} + \boldsymbol{Z}_i \dot{q}_i$$

$$\ddot{\boldsymbol{X}}_i^b = \dot{\boldsymbol{X}}_i \times \boldsymbol{Z}_i \dot{q}_i$$

$$\boldsymbol{F}_i^b = \dot{\boldsymbol{X}}_i \times (\boldsymbol{M}_i \dot{\boldsymbol{X}}_i)$$

end
// inward recursion
for $i = NS$ **to** 0 **do**

$$u_{i+1} = \tau_{i+1} - \boldsymbol{Z}_{i+1}^T \left(\boldsymbol{F}_{i+1} + \boldsymbol{M}_{i+1}^a \ddot{\boldsymbol{X}}_{i+1}^b\right)$$

$$\boldsymbol{M}_i^a = \boldsymbol{M}_i + \boldsymbol{M}_{i+1}^a - \boldsymbol{M}_{i+1}^a \boldsymbol{Z}_{i+1} \boldsymbol{D}_{i+1}^{-1} \boldsymbol{Z}_{i+1}^T \boldsymbol{M}_{i+1}^{a\,T}; \boldsymbol{M}_{N+1}^a = \boldsymbol{0}$$

$$\boldsymbol{F}_i = \boldsymbol{F}_i^b + \boldsymbol{F}_i^e + \boldsymbol{F}_{i+1} + \boldsymbol{M}_{i+1}^a \ddot{\boldsymbol{X}}_{i+1}^b + \boldsymbol{M}_{i+1}^a \boldsymbol{Z}_{i+1} \boldsymbol{D}_{i+1}^{-1} u_{i+1}; \boldsymbol{F}_{N+1} = \boldsymbol{0}$$

if $i < NS$ **then**

$$\boldsymbol{A}_i = \boldsymbol{A}_{i+1} - \boldsymbol{M}_{i+1}^a \boldsymbol{Z}_{i+1} \boldsymbol{D}_{i+1}^{-1} \boldsymbol{Z}_{i+1}^T \boldsymbol{A}_{i+1}$$

$$\boldsymbol{\mathcal{Z}}_i = \boldsymbol{\mathcal{Z}}_{i+1} - \boldsymbol{A}_{i+1}^T \boldsymbol{Z}_{i+1} \boldsymbol{D}_{i+1}^{-1} \boldsymbol{Z}_{i+1}^T \boldsymbol{A}_{i+1}; \boldsymbol{\mathcal{Z}}_N = \boldsymbol{0}$$

$$\boldsymbol{\beta}_i = \boldsymbol{\beta}_{i+1} + \boldsymbol{A}_{i+1}^T \left(\ddot{\boldsymbol{X}}_{i+1}^b + \boldsymbol{Z}_{i+1} \boldsymbol{D}_{i+1}^{-1} u_{i+1}\right); \boldsymbol{\beta}_N = \boldsymbol{0}$$

end
end
// solving for constraint force magnitudes

$$\boldsymbol{\nu} = \boldsymbol{\mathcal{Z}}_0^{-1} \left(\boldsymbol{b}_N - \boldsymbol{A}_0 \ddot{\boldsymbol{X}}_0 - \boldsymbol{\beta}_0\right)$$

// final outward recursion
for $i = 1$ **to** NS **do**

$$\ddot{q}_i = \boldsymbol{D}_i^{-1} \left(u_i - \boldsymbol{Z}_i^T \left(\boldsymbol{M}_i^a \ddot{\boldsymbol{X}}_{i-1} + \boldsymbol{A}_i \boldsymbol{\nu}\right)\right)$$

$$\ddot{\boldsymbol{X}}_i = \ddot{\boldsymbol{X}}_{i-1} + \boldsymbol{Z}_i \ddot{q}_i + \ddot{\boldsymbol{X}}_i^b$$

end

$\ddot{\boldsymbol{X}}_N = 0$. This is a special case of the constrained end effector in Eq. (6.15):

$$\boldsymbol{A}_N = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}, \quad \boldsymbol{b}_N = 0. \tag{6.26}$$

Many inverse dynamics applications want to superimpose a null space motion to the outcome of the constrained ID algorithm, for example, in order to add *posture control* for a humanoid robot, i.e., joint forces or accelerations to generate motions of the legs, torso and arms that do not change the desired hand motion constraints, but that serve to keep the whole robot close to a certain posture, for balancing or obstacle avoidance. Such posture control can, in principle, use an *arbitrary* subset of all possible joints, and the algorithm will find the joint forces that can still realise the desired segment constraints; of course, incompatible specifications of posture control and segment constraints can occur.

The presented recursions deal with the above-mentioned null space motions as follows:

- if the null space motion is specified as desired *accelerations* $\ddot{q}_i$ of some of the joints, it can be incorporated in the initial outward acceleration recursion. This will take them into account in a natural way, resulting in the computation of the corresponding joint forces in the later inward recursion step.
- if the null space motion is specified as desired *forces* τ_i at some of the joints, these are added as given forces to the recursion in Eq. (6.21).

Of course, such extra null space joint forces will violate the minimal acceleration energy property of the presented algorithm, but that violation is then *desired* by the robot task, and not a "failure" of the algorithm.

The presented algorithm does not provide any help in choosing what joint accelerations or forces are appropriate for a desired posture control.

6.4.3 Multiple constraints in a tree

The recursions of the previous Sections 6.4.1 and 6.4.2 apply to *serial* kinematic chains, with a constraint on only the last segment. Extending this to multiple constraints on different segments in a tree-structured kinematic chain does

not change the recursions in Eq. (6.20) and Eq. (6.21) of, respectively, the constraint force matrix $\boldsymbol{A}$ and the accumulated acceleration energy β of each *individual* constraint, but involves some extensions for the acceleration energy *coupling matrix* $\boldsymbol{\mathcal{Z}}$ and for the computation of the *constraint force magnitudes* $\boldsymbol{\nu}$ which is a new contribution with respect to the work of Vereshchagin 1989. The following paragraphs explain the extensions for the case of two fused constraints, but the procedure generalizes to any number of constraints:

- When a constraint applies on a segment that is not a leaf segment, the inward recursions for Eqs. (6.20)-(6.22) can start from that segment, because more distal segments do not contribute to the constraint satisfaction or violation at that segment.

- When the recursion of a k-dimensional constraint c reaches a segment where it must be joined with the recursion of an l-dimensional constraint d, both acceleration constraint coupling matrices $\boldsymbol{\mathcal{Z}}^c$ and $\boldsymbol{\mathcal{Z}}^d$, Eq. (6.22), are fused into a $(k+l) \times (k+l)$-dimensional constraint matrix $\boldsymbol{\mathcal{Z}}^{cd}$:

$$\boldsymbol{\mathcal{Z}}^{cd} = \begin{pmatrix} \boldsymbol{\mathcal{Z}}^c & 0 \\ 0 & \boldsymbol{\mathcal{Z}}^d \end{pmatrix}. \tag{6.27}$$

  This "fusion" is done before going over the first common joint, hence the recursion of $\boldsymbol{\mathcal{Z}}^{cd}$ continues from there on, as in Eq. (6.22), but now with the $6 \times (k+l)$ constraint force matrix $\boldsymbol{A}^{cd}$:

$$\boldsymbol{A}^{cd} = \begin{pmatrix} \boldsymbol{A}^c & \boldsymbol{A}^d \end{pmatrix}. \tag{6.28}$$

  The physical interpretation of $\boldsymbol{\mathcal{Z}}^{cd}$ 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 Eq.(6.23) must be solved:

$$\boldsymbol{\mathcal{Z}}_0^{cd} \begin{pmatrix} \boldsymbol{\nu}^c \\ \boldsymbol{\nu}^d \end{pmatrix} = \begin{pmatrix} \boldsymbol{b}_N^c - \boldsymbol{\beta}_0^c \\ \boldsymbol{b}_N^d - \boldsymbol{\beta}_0^d \end{pmatrix} - (\boldsymbol{A}_0^{cd})^T \ddot{\boldsymbol{X}}_0. \tag{6.29}$$

  The constraints can be satisfied if $\boldsymbol{\mathcal{Z}}_0^{cd}$ is of full rank, which can then be inverted. The numerical complexity of this problem is $\mathcal{O}(m^3)$, with m the *total* number of constraints ($m = k + l$ in the case above).

6.5 Examples of different use-cases

This section presents simulation experiments on a planar serial two-link mechanism. They illustrate the different use-cases of the algorithm, without making the examples overly complex and hard to understand the influences of different parameters. The two-link mechanism contains two segments, each of them is attached to their parent by an actuated joint along the Z axis, both segments have a length of 0.4m and have point-masses of 0.3kg attached at their distal frame. When no constraints are applied the end effector (EE) of the mechanism is free to move in the $X-Y$-plane. The mechanism is shown in Figure 6.3.

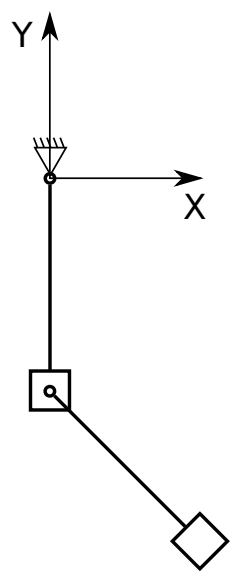

Figure 6.3: The two-link mechanism used in the presented experiments in its initial position.

- In the first experiment the EE is fully constrained:

$$\boldsymbol{A}_N = \begin{pmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 1 & 0 \\ 0 & 1 \\ 0 & 0 \end{pmatrix}, \quad \boldsymbol{b}_N = (\ddot{x}_{des}, \ddot{y}_{des}), \tag{6.30}$$

with $\ddot{x}_{des} = \mathrm{PD}(x_{start})$, i.e. maintain the initial X-position and $\ddot{y}_{des} = \mathrm{PID}(0.5)$ control the Y-position to a new value of 0.5m. A PD controller

is used for the X-constraint instead of $\ddot{x}_{des} = 0$ to compensate for the integration errors and stability. In the initial position of the mechanism $q_1 = 0$ and $q_2 = \pi/4$.

- In the second experiment the EE is only constrained in the X-direction:

$$\boldsymbol{A}_N = (0, 0, 0, 1, 0, 0)^T, \quad \boldsymbol{b}_N = (\ddot{x}_{des}), \tag{6.31}$$

with $\ddot{x}_{des} = \mathrm{PD}\,(x_{start})$, i.e. maintain the initial X-position. Again a PD controller is used to compensate for the integration errors and stability. A null-space motion on the first joint is implemented as a virtual spring:

$$\tau_1 = K(\pi - q_1), \tag{6.32}$$

i.e. joint 1 is forced to go to π.

- In the third experiment the EE is again only constrained in the X-direction and maintained at the initial position but now an external force is applied on the second segment:

$$\boldsymbol{f}_{ext} = (0, 0, 0, -1N, 1N, 0)\,. \tag{6.33}$$

- The fourth experiment is the same as the first experiment but the constraint on the X-direction is removed.

Figure 6.4 shows the calculated torques to satisfy the constraints. Notice that in the experiments without the external force the torques all go back to zero, but in the case of the external force a remaining torque is needed to satisfy the constraint. Figure 6.5 shows the resulting joint positions in the experiments and Figure 6.6 the resulting end effector motion. It is clearly shown in Figure 6.6 that only in the last experiment where no constraint is active on the X-direction the EE is moving in the X-direction.

6.6 Using the acceleration-based recursive constraint solver together with iTaSC

This section explains how the presented algorithm can be used to solve an iTaSC motion specification at the acceleration level with the Gauss' principle of least constraint as the redundancy optimization metric. The presented algorithm is able to solve this problem more efficient then the algorithm presented in Section 1.2.1.

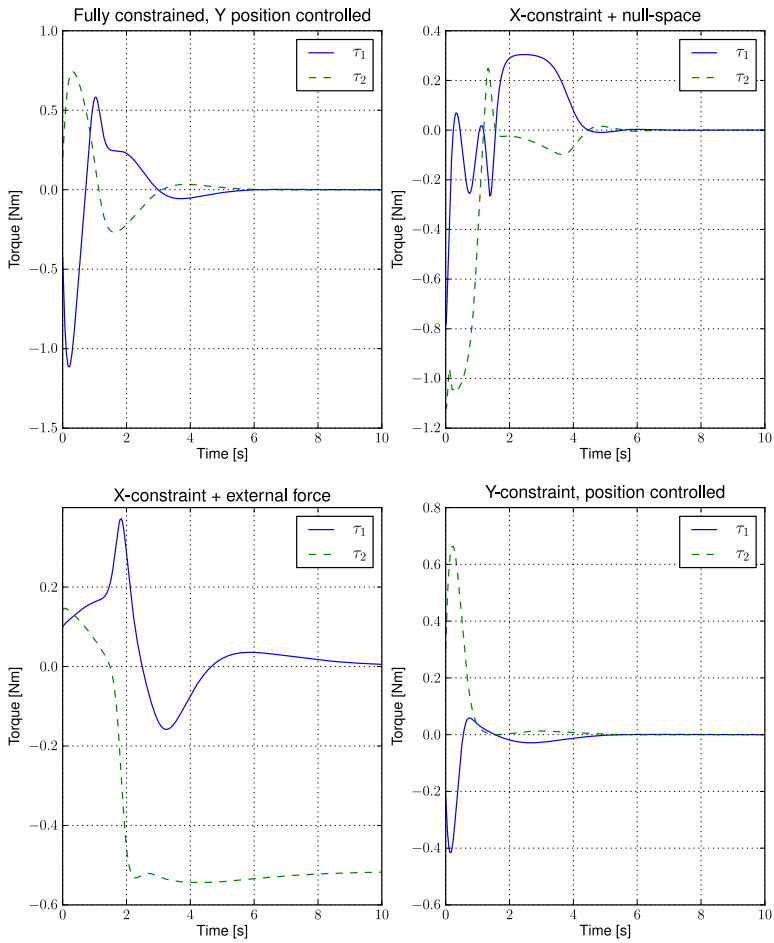

Figure 6.4: Torques to be applied to satisfy the constraints in the four different simulation experiments: (i) both X and Y are constrained, Y is controlled to a new desired position, (ii) only X is constrained and a virtual spring is added on the first joint, (iii) only X is constrained and an external force is applied on the second segment, (iv) only Y is constrained and controlled to a new desired position. Notice that in the simulation experiments without the external force the torques all go back to zero, but in the case of the external force a remaining torque is needed to satisfy the constraint.

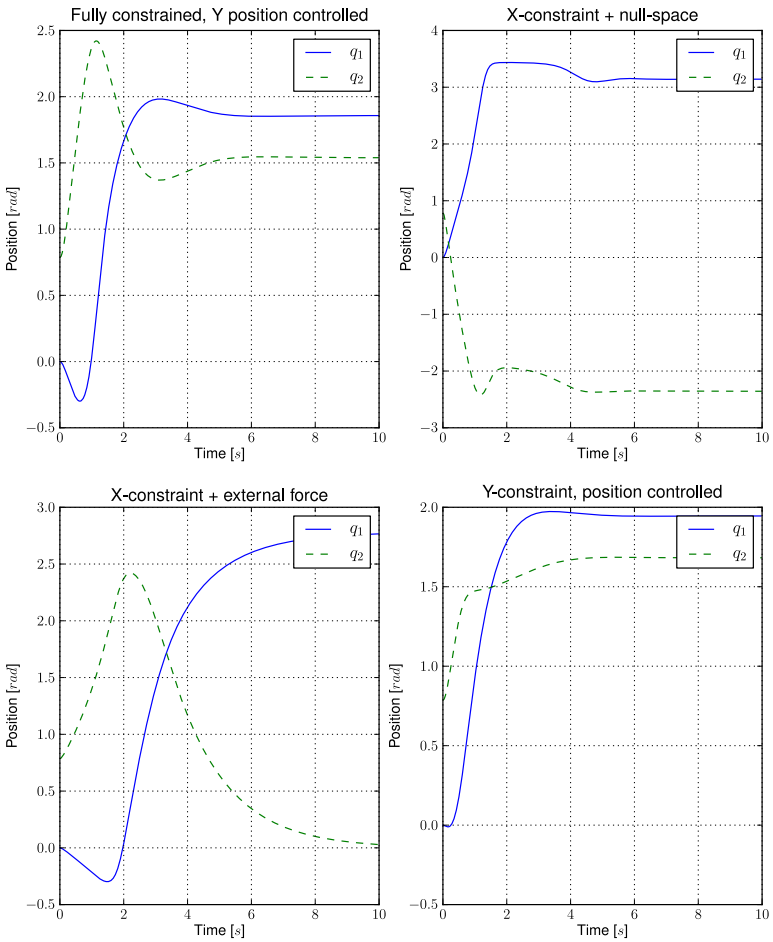

Figure 6.5: Resulting joint positions in the four different experiments: (i) both X and Y are constrained, Y is controlled to a new desired position, (ii) only X is constrained and a virtual spring is added on the first joint, notice how q_1 goes to π (iii) only X is constrained and an external force is applied on the second segment, (iv) only Y is constrained and controlled to a new desired position. Notice that in spite of the similar resulting EE motion of the first three experiments the joint motion is different.

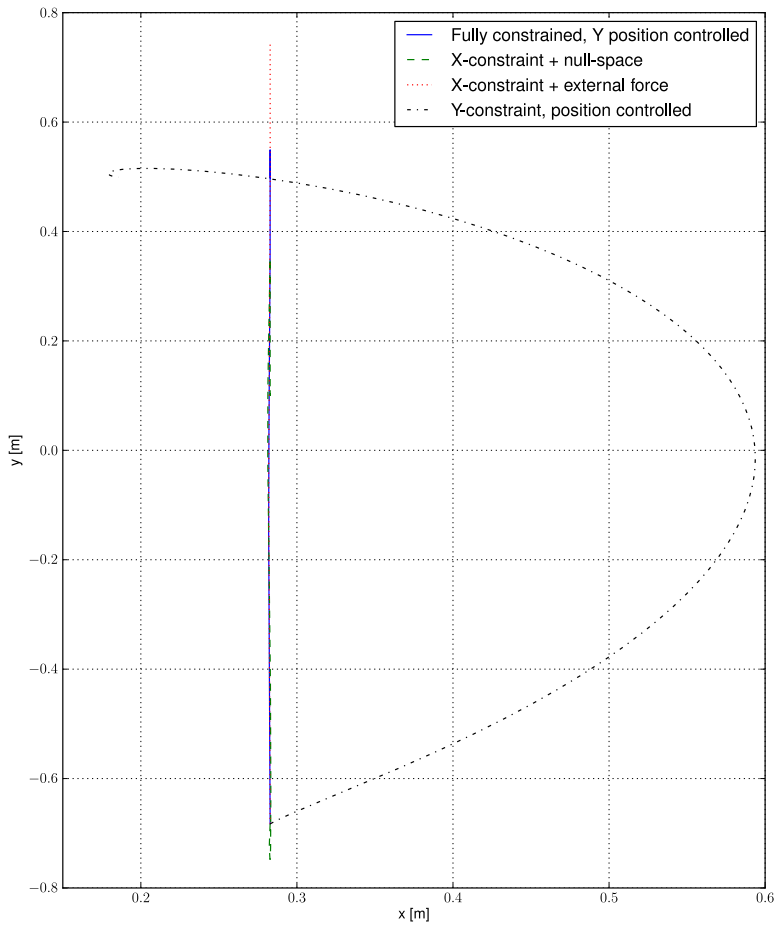

Figure 6.6: The resulting motion of the end effector of the two link mechanism in the four different simulation experiments: (i) both X and Y are constrained, Y is controlled to a new desired position, (ii) only X is constrained and a virtual spring is added on the first joint, (iii) only X is constrained and an external force is applied on the second segment, (iv) only Y is constrained and controlled to a new desired position. It is clearly shown that only in the last experiment where no constraint is active on the X-direction the EE is moving in the X-direction.

Equation (3.8) presents the loop equation of the virtual kinematic at the acceleration level, created by an iTaSC specification. The desired acceleration of the object frame N attached to the robot, can be calculated as

$$\ddot{\boldsymbol{X}}_N = \boldsymbol{J_q}\ddot{q}_d + \dot{\boldsymbol{J}}_{\boldsymbol{q}}\dot{q}. \tag{6.34}$$

Combining Eq. (6.34) with Eq. (3.8) results in:

$$\ddot{\boldsymbol{X}}_N = -\dot{\boldsymbol{J}}_{\boldsymbol{f}}\boldsymbol{\chi_f} - \boldsymbol{J_f}\ddot{\boldsymbol{\chi}}_{\boldsymbol{f}} - \dot{\boldsymbol{J}}_{\boldsymbol{u}}\boldsymbol{\chi_u} - \boldsymbol{J_u}\ddot{\boldsymbol{\chi}}_{\boldsymbol{u}}. \tag{6.35}$$

Combining Eqs. (3.28)(without constraints on the joints) and (6.34) results in:

$$\begin{aligned} \boldsymbol{C_f}\boldsymbol{J_f}^{-1}\ddot{\boldsymbol{X}}_N &= \ddot{\boldsymbol{y}} + \boldsymbol{C_f}\boldsymbol{J_f}^{-1}\left(\dot{\boldsymbol{J}}_{\boldsymbol{f}}\dot{\boldsymbol{\chi}}_{\boldsymbol{f}} + \dot{\boldsymbol{J}}_{\boldsymbol{u}}\boldsymbol{\chi_u} + \boldsymbol{J_u}\ddot{\boldsymbol{\chi}}_{\boldsymbol{u}}\right) \\ \boldsymbol{A}_N\ddot{\boldsymbol{X}}_N &= \boldsymbol{b}_N, \end{aligned} \tag{6.36}$$

which is clearly the same as Eq. 6.15, with $\boldsymbol{A}_N = \boldsymbol{C_f}\boldsymbol{J_f}^{-1}$ and $\boldsymbol{b}_N = \ddot{\boldsymbol{y}} + \boldsymbol{C_f}\boldsymbol{J_f}^{-1}\left(\dot{\boldsymbol{J}}_{\boldsymbol{f}}\dot{\boldsymbol{\chi}}_{\boldsymbol{f}} + \dot{\boldsymbol{J}}_{\boldsymbol{u}}\boldsymbol{\chi_u} + \boldsymbol{J_u}\ddot{\boldsymbol{\chi}}_{\boldsymbol{u}}\right)$

6.7 Comparison with priority-based approach with full acceleration constraints

The differences between this chapter's approach, based on Vereshchagin 1989's weighted, partially specified acceleration constraints, and the *priority-based approach* with complete 6D constraints as in (Khatib et al. 2008) are:

- this chapter's algorithm is fully linear-time (for number of joints), recursive, and does not compute nor invert jacobian matrices of the full tree-structured kinematic chain.
- this chapter deals with both *forward* and *inverse dynamics*; the priority-based approach solves only the inverse dynamics.
- the acceleration constraints in this chapter can be *partial*, while the priority-based approach assumes full six-dimensional accelerations.
- this chapter can both *weigh* or *prioritize* different constraints if they cannot all be satisfied at the same time; the priority-based approach always first solves the *highest priority* constraint, and then uses the *acceleration null space* of that constraint to solve the second highest priority, etc.

6.8 Conclusion

(Vereshchagin 1989) contains a fully recursive, linear-time, inverse and forward dynamics algorithm with partial acceleration constraints, whose counterpart has, to the best of the author's knowledge, never been developed. The contribution of this chapter is to extend Vereshchagin's work to the case of tree-structured robots, with multiple task "end effector" frames, so that it becomes applicable to modern humanoid robots and mobile manipulators. A comparison with the priority-based approach reveals that both methods are complementary.

A `C++` implementation of the algorithm in this chapter is under development as part of the kinematics an dynamics library (*KDL*) in the *OROCOS* project, (Bruyninckx et al. 2003). *KDL* is discussed in Chapter 9 and is available under an open source license.

Chapter 7

Software architecture for robot skills

Programming today is a race between software engineers striving to build bigger and better idiot-proof programs, and the Universe trying to produce bigger and better idiots. So far, the Universe is winning.

Rick Cook

Chapter 5 already highlighted the proposal for the architecture of next generation robot controllers that allows the implementation of robot skills. This chapter extends this vision on how the different parts, and in particular the iTaSC robot controller, of this structure can be implemented in a software architecture. This chapter explains how an iTaSC motion control system can be built from different reusable components and how it is deployed into a running system. The skill coordination and configuration from Chapter 5 can easily be integrated with the presented architecture. An example illustrates the resulting

component architecture for the human-aware force controlled manipulation task with two cooperating robot manipulators.

7.1 Introduction

The presented architecture enables the mandatory decoupling between the actual robotic system hardware and the control system that controls its motion. By using a component-based architecture, which highly decouples the different information sources in the iTaSC motion specification, the developed specific components become highly reusable in different application settings, and can be used in a distributed system. Brugali and Scandurra 2009 and Brugali and Shakhimardanov 2010 explain the need for component-based software on top of object-oriented software and explain how components should be designed to enhance the reusability of components in different component-based systems.

The requirements for both the *skill controller* and the *iTaSC motion controller* will be identified and a proposal for a component-based software architecture is made. Since none of the industrial robot manufacturers supports the use of iTaSC-based motion specifications, an OROCOS-based software library is developed that tries to fulfill these requirements.

7.2 Component-based software architecture for iTaSC

The presented component-based software architecture only covers those components that are relevant for the constraint-based motion control and estimation of the robotic system. This means that

- the presented architecture is *not a complete* robotic system control architecture, and
- the presented architecture only covers that part of the entire architecture that deals with the constraint-based motion control and estimation, iTaSC.

7.2.1 The component model

Before explaining how the components of the iTaSC software architecture look like, the used component model is explained. The proposed component model

is based upon the 4 C's: Communication, Computation, Configuration and Coordination (Radestock and Eisenbach 1996; Radestock and Eisenbach 2003). The interface of a component includes specific software structures for each C.

- *Communication* realizes the exchange of data between components. The components don't care where the information comes from or goes to, a single component only cares about the semantic information of the data.
- *Computation* implements the behavior of each individual component. It computes data to send out, using the data it gets from other components depending on its configuration.
- *Configuration* allows to adapt the available parameters of the component's computation.
- *Coordination* allows the component to fire events to let other components know what is happening inside the component and allows the component to react to events fired by other components and change its configuration or computation according to the semantic meaning of the event. The component does not care which other components react to the events it fires or which other components send out the event it is reacting to.

In the current section, Section 7.3, Section 7.4, and Section 7.5 no specific implementation for the component model is chosen. Section 7.6 discusses the advantages of using the Orocos RTT subproject for the developed reference implementation of the presented architecture.

The iTaSC component infrastructure contains different *types* of components. Each type has a specific interface of its 4 C's to make the entire infrastructure work together. Extra implementation-specific interfaces can be added to a specialized implementation of a component type.

The following component types are identified in an iTaSC component system

- Scene
- Virtual Manipulator
- Robot
- Object
- Interaction Model
- Constraint
- Solver

7.2.2 Scene

The *Scene* is the core component of the iTaSC component infrastructure. The *Robot*, *Object*, *Interaction Model* and *Virtual Manipulator* components are not aware of each others existence. The only iTaSC related component they directly communicate with is the *Scene* component.

Configuration

The *Scene*'s configuration interface includes:

- for each *Constraint* component, its weights $\boldsymbol{w}_i$ and priority p_i,
- for each *Robot* component, the pose of the base of its internal kinematic model with respect to another object frame provided by another *Robot/Object* component or the "world" frame of the *Scene* and the redundancy optimization metric $\boldsymbol{M_q}$ in case of a *Robot* component,
- for each *Object* component, the pose of the base of its internal kinematic model with respect to another object frame provided by another *Robot/Object* component or the "world" frame of the *Scene*,
- for each *Virtual Manipulator* component, the object frames to which the internal kinematic model of the spatial relation is connected,
- for each *Interaction Model* component, the object frames to which the internal virtual linkage is connected,
- the selection of the *Solver* component used to compute the desired robot input, and
- the level at which the motion is resolved, this can be velocity or acceleration.

Communication

The *Scene* defines a generic communication interface for the data flow between all instantiated *Virtual Manipulators*, *Interaction Models*, *Robots*, *Objects*, *Constraints* and the instantiated *Solver*. The *Scene* receives from each of:

- the *Robot* components: its poses of the end effector object frames $\boldsymbol{T}_e^b$ with respect to its base, and in a velocity resolved scheme its robot jacobians $\boldsymbol{J_q}$,

- the *Object* components: its pose $\boldsymbol{T}_{o1}^{o2}$ of the object frame $\{o2\}$ with respect to the object frame it is attached to $\{o1\}$ and if available the uncertainty feedforward $\boldsymbol{J_u}\dot{\boldsymbol{\chi}}_{\boldsymbol{u}}$ in a velocity resolved scheme,
- the *Virtual manipulator* components: its manipulator pose $\boldsymbol{T}_{o1}^{o2}$ and in a velocity resolved scheme its jacobian $\boldsymbol{J_f}$ and if available the uncertainty feedforward $\boldsymbol{J_u}\dot{\boldsymbol{\chi}}_{\boldsymbol{u}}$,
- the *Interaction Model* components: its interaction matrix $\boldsymbol{H}$,
- the *Constraint* components: its output equation, which in a velocity resolved scheme is represented by $\boldsymbol{C_f}$ or $\boldsymbol{C_q}$, and the desired values of its outputs, which in a velocity resolved scheme are $\dot{\boldsymbol{y}}_d$.

The *Scene* sends to each of:

- the *Robot* components: the computed desired robot input. In case of a velocity resolved scheme this is $\dot{\boldsymbol{q}}$,
- the *Object* components: the resulting external pose for the object $\boldsymbol{T}_{o1}^{o2}$ due to the robot motion and the estimated pose in the attached virtual manipulator, and in a velocity resolved scheme the rate of change of this pose ${}_{o1}\mathbf{t}_{o1}^{o2}$,
- the *Virtual manipulator* components: the resulting pose between the object frames it is connected to $\boldsymbol{T}_{o1}^{o2}$, due to the robot motion and the external estimated uncertainty of the object frames it is attached to, and in a velocity resolved scheme also the rate of change of this pose ${}_{o1}\mathbf{t}_{o1}^{o2}$,
- the *Interaction Model* component: the resulting pose between the object frames it is connected to, $\boldsymbol{T}_{o1}^{o2}$, and in a velocity resolved scheme the rate of change of this pose ${}_{o1}\mathbf{t}_{o1}^{o2}$.

Computation

It constructs the overall control problem (3.11):

$$\boldsymbol{A}\dot{\boldsymbol{q}} = \dot{\boldsymbol{y}}_d^{\circ} \tag{7.1}$$

and uses the algorithm provided by the *Solver* component to compute the desired robot manipulator inputs.

Figure 7.1 shows an overview of the iTaSC component architecture and the communication data-flow with respect to the Scene component in case of a velocity resolved control scheme.

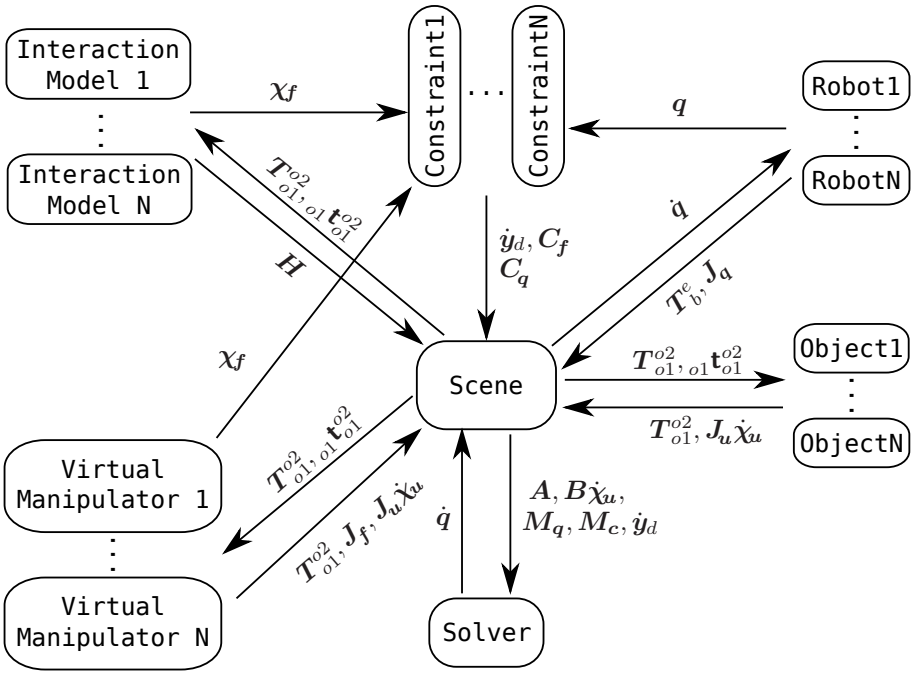

Figure 7.1: Overview of the iTaSC component structure, it shows the communication data-flow between all the components in case of a velocity resolved control scheme.

7.2.3 Virtual Manipulator

The *Virtual Manipulator* component is a *template* component. The template defines the generic interface for the communication with the *Scene* component and for the communication with the *Constraint* component.

Configuration

A specific implementation of the template contains a specific spatial relation model with at least 6 independent auxiliary coordinates χ_f and potentially some uncertainty coordinates χ_u. The model represents a specific parameterized relation between two object frames. The parameters are chosen in such a way to make the definition of a constraint between two object frames as simple as possible, and to give the constraint a semantic meaning. Depending on the model the configuration interface includes component specific parameters.

The estimation configuration consists of the process model of the uncertainties and the measurement equation relating the measurements with the auxiliary coordinates and the uncertainty coordinates.

Communication

The virtual manipulator is unaware of the object frames it is connected to, it receives the relative pose $\boldsymbol{T}_{o1}^{o2}$ between the object frames, and in case of a velocity resolved scheme the rate of change ${}_{o1}\mathbf{t}_{o1}^{o2}$ between the object frames, from the *Scene* component. If the *Virtual Manipulator* contains a measurement equation a communication channel is created to receive the sensor measurements. The component sends the scene the internally updated value of the relative pose $\boldsymbol{T}_{o1}^{o2}$ and in case of a velocity resolved scheme its jacobian $\boldsymbol{J_f}$ and if available the the uncertainty feed-forward term $\boldsymbol{J_u}\dot{\boldsymbol{\chi}}_{\boldsymbol{u}}$.

Computation

Using the information from the *Scene* component together with the process model of the uncertainty coordinates, the external measurements and the measurement equations, the component is able to update its internal model and calculate the current values for $\boldsymbol{\chi_f}$, $\boldsymbol{\chi_u}$ and possibly $\dot{\boldsymbol{\chi}}_{\boldsymbol{u}}$. Using these updated values the component updates the pose $\boldsymbol{T}_{o1}^{o2}$ between the object frames and in case of a velocity resolved scheme its jacobian $\boldsymbol{J_f}$ and possibly the uncertainty feedforward term $\boldsymbol{J_u}\dot{\boldsymbol{\chi}}_{\boldsymbol{u}}$.

7.2.4 Robot

The *Robot* component is a template component. The template defines a generic communication interface with the *Scene* component and the *Constraint* component.

Configuration

A specific implementation of the template contains a specific kinematic model of a robotic manipulator. It defines the available object frames, $\boldsymbol{T}_{bi}^{e}$ to which virtual manipulators or sensors can be connected.

Communication

The *Robot* component is the iTaSC architecture's interface to the actual or simulated robot manipulators and receives the desired manipulator output, $\dot{\boldsymbol{q}}$ in case of a velocity resolved scheme from the *Scene* and sends it to an external component from which it receives the measured joint position $\boldsymbol{q}$. The *Robot* sends the *Scene* the updated values for the object frames $\boldsymbol{T}_{bi}^{e}$ and in case of a velocity resolved scheme the robot jacobian $\boldsymbol{J}_{\boldsymbol{q}}$.

Computation

Using the measured joint positions the *Robot* component updates its kinematic model and calculates the poses of the object frames, $\boldsymbol{T}_{bi}^{e}$ and in case of a velocity resolved scheme the robot jacobian $\boldsymbol{J}_{\boldsymbol{q}}$.

7.2.5 Object

The *Object* component is a template component that defines the generic communication interface with the *Scene* component.

Configuration

A specific implementation contains the specific kinematic model for the uncertainty of the objects position. It defines its object frame $\{o2\}$ to which other objects, virtual manipulators or sensors can be connected with respect to the object frame $\{o1\}$ it is attached to. The component can also contain the process model for its modeled uncertainties, and possibly a measurement equation.

Communication

The object does not know to which other object frame in the scene it is connected, it receives the external calculated relative pose, $\boldsymbol{T}_{o1}^{o2}$ and in case of a velocity resolved scheme the rate of change between the object frames, ${}_{o1}\mathbf{t}_{o1}^{o2}$ from the *Scene* component. If the *Object* contains a measurement equation a communication channel is created to receive the sensor measurements. The *Object* sends the internally updated pose $\boldsymbol{T}_{o1}^{o2}$ back to the *Scene*.

Computation

Using the externally calculated pose $\boldsymbol{T}_{o1}^{o2}$, the process model, the measurement equation and the measurements the *Object* component updates its uncertainties and the pose $\boldsymbol{T}_{o1}^{o2}$ and if available, in case of a velocity resolved scheme the uncertainty feedforward $\boldsymbol{J_u}\dot{\chi}_{\boldsymbol{u}}$.

7.2.6 Interaction Model

The *Interaction Model* is a template component. The template defines the generic interface for the communication with the *Scene*.

Configuration

A specific implementation of the template contains a specific interaction model between, or a measurement equation that depends on the pose between the object frames it is attached to, see Section 4.1.

Communication

The object does not know to which other object frame in the scene it is connected, it receives the externally calculated relative pose, $\boldsymbol{T}_{o1}^{o2}$ and in case of a velocity resolved scheme the rate of change between the object frames, ${}_{o1}\mathbf{t}_{o1}^{o2}$ from the *Scene* component. If the *Interaction Model* contains a measurement equation a communication channel is created to receive the sensor measurements. In case of a velocity resolved scheme it sends the *Scene* the interaction matrix $\boldsymbol{H}$.

Computation

Using the external calculated pose $\boldsymbol{T}_{o1}^{o2}$ and the measurement, the interaction model $\boldsymbol{H}$ is updated.

7.2.7 Constraint

The constraint is a template component. The template defines the generic interface for the communication with the *Scene*.

Configuration

A specific implementation of the template contains an output equation that defines the output $\boldsymbol{y}$ in terms of the controllable coordinates $\boldsymbol{\chi_f}$ and $\boldsymbol{q}$ of the available *Virtual Manipulator*, *Robot* and *Interaction Model* components and a control equation to compute the desired value of the output $\boldsymbol{y}_d$ in terms of its control parameters $\boldsymbol{\gamma}$, see Section 5.2.

Communication

A communication channel is created for each variable the output depends on. The computed desired output and the output equation, in the case of a velocity resolved scheme this is $\boldsymbol{C_f}$ and $\boldsymbol{C_q}$, is send to the *Scene*.

Computation

The desired output is computed using the configured control equation from the current values of the coordinates the output depends on.

7.2.8 Solver

The Solver component is not an active component. It can be considered as a *Container* component that offers an algorithm to the *Scene* to solve the desired joint output from the configured iTaSC specification. It can be configured using solver-specific parameters.

7.3 Deployment of the iTaSC Scene

The deployment of the iTaSC scene is one of the most important steps. Once this is done the run-time takes over and the *Coordination* is in charge of the execution. The deployment of the Scene is not the same as the deployment of the individual components. The deployment of the individual components is the responsibility of the component framework that is used for the implementation. The deployment of the iTaSC scene takes care of creating the necessary communication connections between the different components and is the responsibility of the configuration step of the *Scene* component.

7.3.1 Registration of Robots and Objects

Every Robot or Object component that is involved in the iTaSC motion specification has to register itself at the *Scene* component. At the registration the *Scene* will store the following information for each *Robot*/*Object* component:

- the total number of actuated joints in case of a *Robot* component,
- the pose of the base of the internal kinematic model with respect to another already registered object frame or the "world" reference frame of the scene,
- all the object frames that are provided by the internal kinematic model of the registering component and that can be involved in a virtual kinematic loop, and
- all structures needed for the run-time data communication with the respective component.

7.3.2 Registration of Virtual Manipulators and Interaction Models

Every *Virtual Manipulator* and *Interaction Model* needs to be registered at the *Scene*. During the registration the following information is stored for each Virtual manipulator component:

- the two (already registered) object frames to which the virtual manipulator or interaction model is attached, nad
- all structures needed for the run-time data communication with the respective component.

7.3.3 Registration of the Constraints

Every *Constraint* component that defines an output has to be registered at the Scene. During the registration the following information is stored for each Constraint component:

- all structures needed for the run-time data communication with the component.

7.3.4 Selection of the Solver

Different kinds of solvers exist to solve the control problem: a regular weighted pseudo inverse algorithm, a damped least squares solution or other optimization algorithms. The *Scene* is configured to use the algorithm of one of the available *Solver* components. Every solver should be able to handle the following information:

- the control problem to solve (7.1),
- the robot manipulator optimization metric $\boldsymbol{M_q}$, and
- the constraint weights $\boldsymbol{w}$ and their priorities $\boldsymbol{p}$.

7.4 Motion Coordination

The previous sections explain what the iTaSC software architecture looks like, how it is deployed and how the data communication is set up. Once this is done the configured components are coordinated using the *Skills* from Chapter 5. As explained in Chapter 5 the skills are implemented using finite state machines.

The single skill coordinates a minimal set of constraint components. This means the skill only coordinates those *Constraint* components that define the output that is used for the functional behavior that the skill describes.

The required interface for the coordination of the motion specification is explained in Section 5.3. For each constraint the skill motion is coordinated using the following parameters:

- The priority and the weight of the constraint, which define the motion behavior in case of an over-constrained specification. The *Scene* component has the appropriate structures to coordinate these generic parameters in the skill's finite state machine.
- The estimated current value and desired value of the constraint output and the constraint's control parameters. The *Constraint* component has the appropriate structures to coordinate these constraint-specific parameters in the skill's finite state machine.
- The estimated values for the uncertainty coordinates and parameters indicating the convergence of the estimation process. The *Virtual manipulator* and *Object* components have the appropriate structures to provide this information to the skill's state machine.

Every component can define component-specific events which can be used in the finite state machine logic. These events have a semantic meaning which is related to the component that defines it.

7.5 Example: Human-aware force-controlled manipulation task using two robots

Figure 7.2 shows the components and their connections for the example presented in Chapter 3.

The blue boxed components are the *Robot/Object* components:

- The *Robot 1* component contains the kinematic model of the first robot to which the camera and the 2.5D contour object are attached. It provides only one object frame, $\{e1\}$ which is the robot's end effector.
- The *Robot 2* component contains the kinematic model of the second robot to which the probe, the force sensor and the laser distance sensor are attached. This component also provides only one object frame, $\{e2\}$ which is this robot's end effector.
- The *Person* component contains the uncertainty model for the closest person. The estimation of $\chi_{\boldsymbol{u}}$ and $\dot{\chi}_{\boldsymbol{u}}$ is not done internally but provided by an external component which implements the multi target tracking and localization algorithm as described in (De Laet et al. 2010). It provides the object frame, $\{person\}$.
- The *Contour* component contains the uncertainty model of the height and orientation of the front plane of the 2.5D contour object. It provides the object frames $\{front\}$ and $\{contact\}$.
- The *Camera*, *Probe* and *Laser* component do not contain any model, they just provides the object frames $\{camera\}$, $\{probe\}$ and $\{laser\}$.

The green boxed components are the virtual manipulator components:

- The *Contact - probe control* component contains the kinematic model needed to model the contact between the $\{contact\}$ and the $\{probe\}$ frame.
- The *Contact - probe estimation* component contains the kinematic model needed for the estimation of the modelled contact point between the $\{contact\}$ and the $\{probe\}$ frame. The difference with the previous

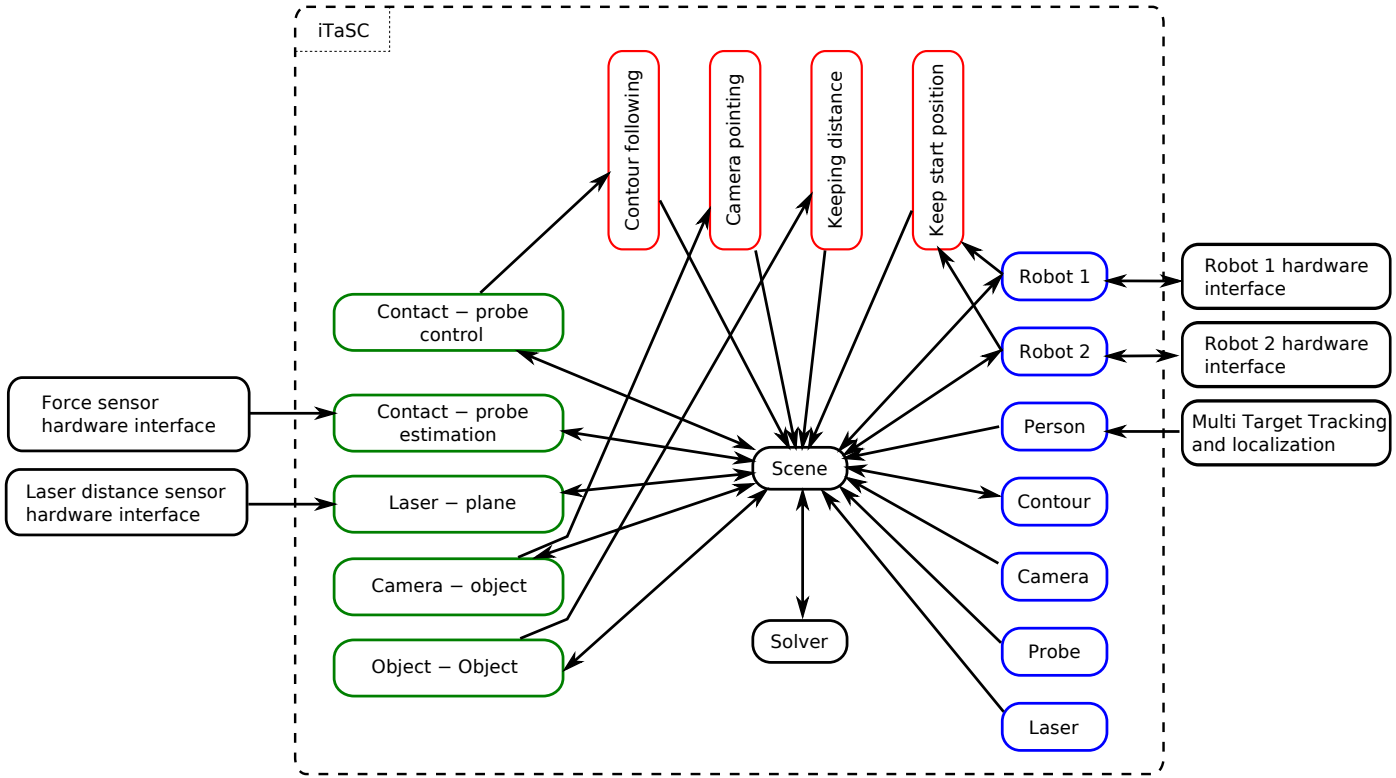

Figure 7.2: The different components and their data flow connections for the "human aware force controlled manipulation using two robots" example. Only the components inside the dashed iTaSC block are iTaSC related. The blue boxed components are *Robot/Object* components, the green boxed components are *Virtual Manipulator* components and the red boxed components are the *Constraint* components. The external components take care of the hardware interface.

component is that the modelled coordinates map directly to the force measurements of the *Force sensor* component and are not optimized for the control specification.

- The *Laser - plane* component contains the kinematic model needed for the estimation of the height and orientation of the modelled front plane of the 2.5D contour object using a laser distance sensor. One of the auxiliary coordinates directly maps to the laser distance measurement. This component gives an estimate of the pose of the $\{front\}$ frame with respect to the $\{laser\}$ frame.
- The *Camera - object* component contains the kinematic model to express the projection of an object frame, which in this case is the $\{person\}$ frame projected in the $\{camera\}$ frame.
- The *Object - object* component contains the kinematic model to express the distance between two object frames, in this case between the $\{person\}$ frame and the $\{contour\}$ frame. One of the auxiliary coordinates in the model directly maps onto the distance.

The red boxed components are the *Constraint* components:

- The *Contour following* component defines the desired output for the auxiliary coordinates of the *Contact-Probe control* component, to track the contour.
- The *Camera pointing* component defines the desired output to keep the tracked person's head in the middle of the camera image using the auxiliary coordinates of the *Camera-object* component.
- The *Keeping distance* constraint defines an output on the auxiliary coordinate of the *Object-Object* component to keep a certain minimal distance between the $\{person\}$ frame and the $\{contour\}$ frame.
- The *Keep start position* constraint defines a desired output for all the robot manipulator joints. The desired position value is set to a specific position that represents the start position of both robots.

More details about the definition of these constraints can be found in Section 5.5.

How the specific skill FSM are implemented is discussed in Chapter 5. The contour following, camera pointing and keeping distance skills are all handled in the first priority level. The keep start position constraints are put in the second priority level. This results in a spring behavior for all the joints without interfering with the constraints of the other skills.

7.6 Orocos RTT-based implementation

To create a practical implementation of the presented component-based iTaSC software architecture the underlying software architecture used for this implementation should provide the following:

- a component primitive which implements the proposed component model from Section 7.2.1,
- hard-realtime execution of the computation of each component,
- a realtime-capable communication infrastructure for the dataflow between all the components,
- distribution of the components to different processes or machines, and
- FSM scripts to provide coordination for the skills.

A reference implementation of the entire generic iTaSC component-based architecture and all the example specific components is developped using the Real-Time Toolkit (RTT) of the OROCOS project (Soetens 2006; Soetens and Bruyninckx 2005; Bruyninckx 2001).

How the Orocos RTT meets the demands of the component-based iTaSC software architecture is clear from the *What is RTT?* page of the OROCOS website:

> *The RealTime Toolkit (RTT) library allows application designers to build highly configurable and interactive component-based realtime control applications.*
>
> *You might use it to:*
>
> - *Control devices ranging from sensors to complete robots*
> - *Tune your algorithms at run-time*
> - *Write your controller as a hierarchical state machine*
> - *Configure components and the application from XML files*
> - *Interact with your devices directly from a GUI or command prompt*
> - *Extend it with your own data types*
> - *Extend your legacy control applications with all the above*
> - *Run it on standard operating systems as well as dedicated realtime systems*

The realtime toolkit allows components to run on (realtime) operating systems and offers realtime scripting capabilities, the component communication and distribution API and XML configuration. Figure 7.3 shows the position of RTT in the overall application stack.

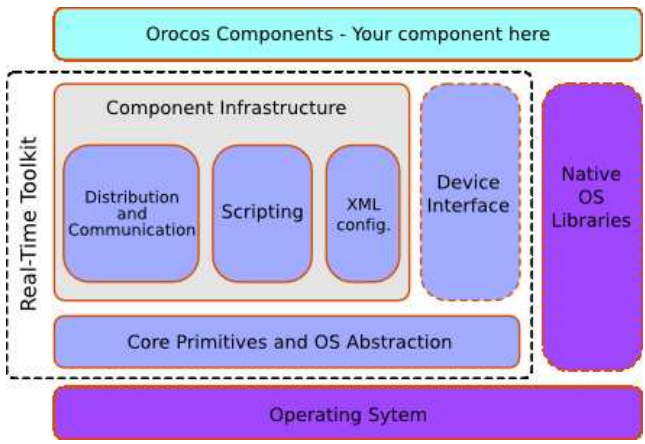

Figure 7.3: The RealTime Toolkit Application Stack: an Orocos component is built upon the RealTime Toolkit (RTT) library.

Each component is built using the "TaskContext" primitive: an active object which offers thread-safe and efficient ports for (lock-free) data exchange. It can react to events, process commands, or execute Finite State Machines in hard realtime. It can be configured on-line through a property interface (set/get values) and XML files. Figure 7.4 shows the generic interface of the RTT "TaskContext".

7.7 Conclusion

This chapter explains what a component-based architecture for iTaSC-based motion specification, control, estimation and coordination (Skills) looks like, to provide the necesarry decoupling and modularity. It presents a generic modular component architecture that allows to build small reusable components to flexibly create an automatic configurable run-time system. A specific instantiation of the architecture is illustrated using a human-aware force controlled manipulation task using two robots. A reference implementation of the presented architecture is developed using the RealTime Toolkit of the OROCOS project.

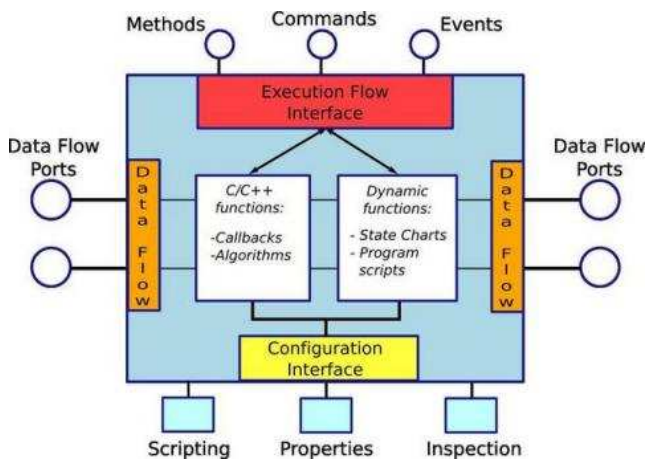

Figure 7.4: Orocos TaskContext: Data flows through ports and is manipulated by algorithms in the component.

Chapter 8

Formal representation of the Robotics Scene Graph

A standard international language should not only be simple, regular, and logical, but also rich and creative.

Edward Sapir (1884 – 1939)

This chapter identifies and documents the modeling requirements for the kinematic structures in robotic platforms. These requirements are compared with the Scene Graph features offered by the COLLADA standards from the computer animation domain. Since the COLLADA standard meets the requirements very well, this chapter proposes a Robotic Scene Graph standard using the COLLADA file format. This chapter explains what extensions are needed to use COLLADA for the Robotics Scene Graph Standard and how constraint-based motion can be specified using the COLLADA standard.

8.1 Introduction

Future robot control systems will have to deal with the coordinated control of multiple robots, often in time-varying and geographically spread configurations, in which newly arriving robots have to be integrated seamlessly and effectively into the control system. Also a lot of research effort is going into *learning* both motion specification and motion coordination. These learned motion and coordination specifications have to be interchangeable between various robotic systems. Therefore it is not only necessary to have a universal standard representation to describe all the relevant features of a robotic system but also a standardized representation for the specification of its desired motion and the coordination of this desired motion.

In Chapter 5, the Finite State Machine is presented as the unifying representation for the coordination of motion in all robotic domains. The UML standard (OMG 2009) of the OMG standardization group is one of the most adopted formal representation standards for Finite State Machines. XABSL (Lötzsch et al. 2004) is an example of how hierarchical state machines can be defined in a text file format using XML.

For the representation of robot systems various representations already exist, but none of them is already adopted as a standard. The reuse of the existing representations outside the scope of the founding project of the representation is non-existing. Hence, the need arises to create a universal standard formal representation to describe all relevant features of robotic systems and their motion specification.

However, "all relevant features" of a robot can mean many things, from the kinematic, dynamic and actuation properties, over the available sensor and tools, to the set of motions and tasks that a particular robot controller can execute.

This chapter makes the case for a formal representation of the "Robotics Scene Graph", which covers:

- the representation of kinematic and dynamic (lumped parameter) models of the *robots*,
- the possibility to attach to these robots sensors, motors, tools, shape, and task-relevant reference points or frames, and
- the possibility to locate the robots with respect to objects in the world (which often are other kinematic chains) that are relevant to the task.

The chapter also takes into account that the presented formal representation can be used as part of a (future) larger standards eco-system, such as

AutomationML (AutomationML 2008) to describe more features of robotics systems, e.g., more detailed descriptions of the dynamic properties of the motors, sensors, or joints, or the description and control of complex sensor-based motion tasks and/or processes.

8.2 The Robotics Scene Graph Standard

This chapter makes a concrete standardization proposal, called *Robotics Scene Graph Standard*. Wide acceptance of such a standard will improve the reusability of robot platform representations and motion specification, and, hence, also the *planning*, *sensing* and *control* software that works on top of such representations. The *Robotics Scene Graph Standard* covers the kinematic chain aspects of robot platforms, including the location of sensors, motors and tools on the chain, as well as the location of objects in the scene that are relevant to the robot's task. The *functional properties* of such "peripherals" are not part of the Robotics Scene Graph, but the suggested standard describes the *geometric* properties of their attachments to the carrying kinematic structure, and provides *semantic tags* to link to external representations of these properties. The suggested standard is appropriate for *persistent storage*, i.e., to represent information about the robotics platform that should be read from, or written to, a file. The standard is less appropriate to communicate the *run-time state* (actual positions, velocities, accelerations, forces, etc.) of a robot platform to and from controllers in realtime.

The following, mostly non-technical, trade-offs are the drivers behind the suggested standard proposal:

- *Appropriate decoupling*: this chapter motivates the place of the standard in the larger context of robotics hardware and software-modeling, interfacing and programming. The standard fills the gap between the (software) access to the individual robot hardware components (sensors, motors, tools) on the one hand, and software libraries for robot motion control, estimation, and planning on the other hand. It is large enough to be useful as a stand-alone standard, and yet small enough to contain only modeling primitives that would not profit from further subdivision into smaller standards.
- *Community support*: the suggested standard fills a gap in almost all existing robotics software projects, and, hence, gives them a minimal-effort opportunity to add the functionality to exchange robotic platform models.

- *Tool support*: the standard should be compatible to a large extent with existing computer animation and computer-aided engineering software (e.g., Blender (Roosendaal 2002), Catia (Dassault Systèmes 1984), or 3D Studio Max (Autodesk Inc. 1996)) which already support numerous existing standards such as COLLADA, STEP, STL, VRML, etc. The tool support facilitates the creation of computer-aided specification and simulation of motion programs for robotics platforms.

The core of every medium-scale robotic platform is its *kinematic structure*, that is, the interconnection structure of rigid body segments and joints. The robot mechanically allows *to attach* sensors and tools, and *to move* them according to the robot's task, by means of the actuators attached to the robot's joints. (Micro-robots are an example of robots that do not have kinematic structures, but rely on other principles for their motion generation and control.) The scientific and technological background behind robot control is well understood since several decades already (Paul 1981; Popov, Vereshchagin, and Zenkevich 1978; Vukobratović 1973), and covers a wide variety of kinematic families, ranging from the serial chains of industrial robots whose kinematic designs matured in the 70s, over the mobile robots of the 80s, to the parallel, walking, mobile manipulation and humanoid robots that became commonplace in the 90s. Notwithstanding the maturity of the field, no universally accepted standard format exists to represent kinematic structures, let alone complete robotic platforms. Hence, generic robot control systems can not yet be made to work with all possible robot hardware that they are, in principle, able to control.

8.2.1 Current existing robot description formats

Different description formats already exist which support part of the necessary features needed in the Robotic Scene Graph.

Unified Robot Description Format (URDF)

The Unified Robot Description Format is defined by ROS (Willow Garage 2009). It is an attempt to create a general robot description. It assumes that the robot model consists of rigid body links connected by joints. Every rigid body link has three properties: (i) its inertial properties, (ii) its visual properties which describe how it should be visualized and (iii) its collision properties that describe the geometric model that should be used for collision detection. Every joint element contains information about its parent and child element and its dynamic properties and motion limits. It already has some extensions to incorporate the description of sensors, but this is only used

in simulation together with Gazebo (Koenig and Howard 2004), a robotics simulation package. It does not support the ability to specify motions or locations of the robot with respect to other objects in the "world".

Gazebo's world description format

Gazebo (Koenig and Howard 2004) is a 3D multiple robot simulator including dynamics. It does not only simulate the dynamics of the robotic system, but it can also simulate different kinds of sensors. It is designed to work together with Player (Gerkey et al. 2007), and recently is also available as part of ROS (Willow Garage 2009). It defines its own description format to create its simulation scene, including robots, sensors and environment. Gazebo's world description is composed of different *models*, which can be robots, sensors, manipulatable objects or static environment features. The models are only known internally in Gazebo. This description is thus less extensible to other systems and less appropriate to describe a complete robotic system in the XML description.

OpenRave XML description format

OpenRave, the Open Robotics and Animation Virtual Environment (Diankov and Kuffner 2008) is targeting for real-world autonomous robot applications, and includes a seamless integration of 3-D simulation, visualization, planning, scripting and control. It has developed its own XML format to store all robot and scene descriptions. It supports the description of the *environment* including *objects* represented by rigid bodies that can be connected by joints. *Robots* extend the rigid body structure by including a manipulator object, these objects include previously defined joints and can contain a pointer to a specific solver to solve the manipulators kinematics. *Sensors* can be attached to a rigid body, its attributes depend on the type of sensor.

The description format of OpenRave is the most extensive of the above presented specific robotics formats. But is does not yet provide a way to specify the desired motion, or any motion related information. And since OpenRave's desricription format is only used within the OpenRave software it can hardly be considered as a standard.

8.2.2 Robotics Scene Graph requirements

This section explains which properties of robotic platforms need, and need not, be represented in a *Robotics Scene Graph* standard. That choice is motivated

by separating the properties into four well-defined levels of information, the exact scope of which is defined by the desire to *maximally decouple* the formal specifications needed at each level. The next sections discuss the requirements to represent the kinematic structure and the geo-spatial positioning which are needed in the standard.

Four levels of access to the Robotics Scene Graph information

The following four levels of robot control software all require access to the scene graph of the robotic platform that they are controlling, for different reasons and with different, complementary, information involved:

1. *Interaction with the hardware*
   A robot controller must be able to read the raw data from its sensors, and to send commands to the driver electronics of its motors. At this level, there is no need to know the precise geometric way in which these hardware components are *connected* to the robotic platform. The Scene Graph must *only* provide the *list of unique identifiers* of all pieces of hardware to access. These identifiers are needed to facilitate compatibility checks between the hardware drivers in the robot control software and the hardware that is actually present on the robotics platform.

2. *Geometric, kinematic and dynamic properties*
   This is the level of a formal representation of:

   - the mechanical properties of the *kinematic structure* of the platform(s);
   - the *location* of *sensors*, *motors* and *tools* on those kinematic structures;
   - the *location* and *geometry* of objects that are relevant to the robot's task (including other kinematic structures); and
   - the *geo-spatial location* of the robotic platform(s) (i.e., where is it placed on the earth coordinate system).

   The above mentioned kinematic structure is a light-weight, geometric "skeleton" data structure, with "hooks" attached at multiple locations, linking to other data structures that represent:

   - the geometric *shape*;
   - the *dynamic parameters* of all objects of the robotic platform;
   - the *dynamic parameters* of all the joints and their connection to the motors.

- the *interconnection* between the objects and the joints; and
- any other *frames* or *tags* attached to objects that are relevant to the robot's task.

The location and geometry of the relevant objects in the environment are given with respect to the *root* of the Scene Graph; this can (but does not have to) be the root segment of a kinematic structure in the scene. The root data structure can be linked (at runtime) to geo-spatial coordinates on the earth globe where the robotics platform is located.

All the information in this level is typically permanently fixed for each *scene* (i.e., a set of cooperating kinematic structures and environment objects), and does not change for each individual motion or task performed by the individual robotic platforms in the scene.

3. *Instantaneous constraint-based motion specification*
This representation level *adds* specific information to the Scene Graph representation of the previous level, for each *motion* that the platform has to perform. This extra information allows:

   - *to specify* what motion is expected of the platform,
   - *to compute* which robot joint motions and/or forces are required to generate the desired instantaneous motion and/or force for all sensors, tools or frames, and
   - *to update* some parameters in the Scene Graph (e.g., joint angles, or locations of objects) that change through the executed motion.

   This level and the previous one are decoupled, in a strictly hierarchical way, since the kind of extra motion information can be *attached* to the Scene Graph of a kinematic structure (or kinematic structures) in a transparent way, that is, without changing anything to the representation used at the previous level. A large body of literature exists for this level; a unified, "best practice" specification and control formalism has been presented in (De Schutter et al. 2007), and has been used in all other chapters in this thesis.

4. *Motion coordination*
This level of robot programming is concerned with making decisions about which instantaneous motion specification to schedule next, and with what configuration of their parameters. The extra information with respect to the previous level is the specification of:

   - the events from sensors, controllers, or from the state of the kinematic structure (such as reaching a joint limit or any other motion constraint) that give rise to a (potential) switch in configuration of instantaneous motion model, and

- the *coordination* algorithm that makes the decisions based on these events.

Again, this level and the previous one are strictly hierarchically decoupled, linked to each other by only the motion event and configuration semantics. How this level can be specified and represented is explained in Chapter 5.

Best practice in representations

The subdivision into four levels, as given above, provides a *loosely decoupled* hierarchy in the overall formal representation of robotics platforms, and it is the result of the following "best practice" trade-offs:

- the scene graph "skeleton" is in itself a very simple data structure, that provides all services to couple the information of all four levels in a structured, efficient and unambiguous way.
- the overlap of the information shared by two neighbouring levels is much lower than the amount of information in each particular level.
- each level presents a formal *interface representation* to the lower level, in which various *implementations* of that formal interface can exist.
- the scene graph contains all information about the components needed in the execution of robot tasks, but it does not represent any task itself. The task specification level "just" has *to configure* the scene graph level with the appropriate *constraints* and to provide a *computational solver* to transform the constrained-based task specification into instantaneous "commands" to the scene graph components.

The *Robotics Scene Graph* standard of this chapter corresponds to the complete second level, and, together with the first level (which is an example of the proven concept of a *Hardware Abstraction Layer*), it is completely understood by the current robotics industry and research groups, and, hence, ready to be standardized. Standardization of the third and fourth levels is much less straightforward, because the "best practice" trade-offs at these levels have not yet been identified, let alone formalized, with sufficient maturity to make a standardization effort successful. Section 8.4.2 however presents an effort for the formal representation of motion specification based on the constraint-based motion specification of (De Schutter et al. 2007).

8.2.3 The Robot Scene Graph's structure

This section explains the primitive elements that, together, form a formal representation of a Robot Scene Graph. In its minimal form, the Robot Scene Graph consists of a geometric skeleton data structure, to which representations of the other primitives of the robotics platform are attached. Its main goal is to identify the necessary components of the Robot Scene Graph. This chapter follows this minimalistic approach, and, hence, suggests the standard to contain the following primitives:

- `Universal Identifier`: this is a *string* that gives each robotics platform a world-wide unique name. This chapter suggests the well-known "inverse class hierarchy" notation used in Java: `lab_id.robots.robot_platform_id`, where `lab_id` is a unique name of the lab, factory,... in which the robot with local identifier `robot_platform_id` is located. The advantage of this `Universal Identifier` suggestion is that it is easy to transform into a *Universal Resource Identifier* of the W3C standard, and hence to use it to allow computers to access the data on the World Wide Web.
- `Scene Graph`: the "container" data structure in which possibly multiple *kinematic structures* can be placed (together with other *objects* (rigid bodies) that are relevant in the robotic application), and in which "relationships" of various kinds can be specified between (`Frames` on) the kinematic structures and the objects.
- `Root`: the root of a `Scene Graph` is a reference frame, that can be placed somewhere in the world, and with respect to which all the geometric information of all other primitives is expressed. Hence, moving the root moves the complete scene, with all its kinematics structures and objects. The `Root` specification in the standard contains no data, at least not in the file representing a particular robotics platform. It is only in the geo-spatial representation (Section 8.2.4) of the world that `Root` frames of different robotics platforms get a real value.
- `Frame`: this is a representation of the position and orientation of a right-handed reference frame, with respect to the reference frame of an object, or the `Root`. `Frames` are the geometric "hooks" to attach all other primitives. The `Frame`s *attributes* represent the *homogeneous transformation* of the specified location with respect to the reference frame of the object it is attached to.
- `Segment`: this represents a single rigid body (with information in some *attributes* about (i) its inertial parameters, and (ii) the `Frame`s attached to it), and to which other primitives can be hooked up. Each `Segment`

has a "home" `Frame`, which represents the "zero" position and orientation of this `Segment` with respect to the previous `Segment` it is attached to in a kinematic structure. Or rather, with respect to the `Frame` attached to the `Joint` which connects this segment to the previous `Segment`. A segment can have several semantic `Tag`s; to link with external primitives like its `Shape`.

- `Joint`: this represents a motion constraint between two `Segment`s. For robotics (and computer animation) purposes, the following set is rather exhaustive: revolute, prismatic, ball, and Cardan joints. A `Joint` connects a `Frame` on one `Segment` to a `Frame` on another `Segment`. A `Joint` can have several *attributes*, to represent its *limits* in position, velocity, acceleration, and/or force. It can also have several semantic `Tag`s; for example, to link to external descriptions of its flexibility, or damping.

- `Shape`: this is a "plug-in hook" that points to a formal description of the geometrical shape of a `Segment`. The `Shape` is relevant for motion planning, collision detection, sensor simulation, visualization, etc. The envisaged standard provides some *built-in* shape attributes, for only the most common geometrical shapes (cylinder, cube, sphere, etc.).

- `Tag`: this is a *semantic attribute* that gives a unique name (and, hence, *meaning*) to each of the previous elements. This allows, for example, to link the representation in this *Robotics Scene Graph* standard to other representations; or to place (the standard representation of) a robot arm on a mobile platform when the arm has already its own *Robotics Scene Graph* representation file, and the `Tag` refers to the `Unique Identifier` of that arm.

The primitives above are all *static* data structures: they contain data that do not change over time. However, the kinematic structure itself will undoubtedly move during its lifetime as the core of a robotics platform. Such a more complete motion representation of a kinematic chain belongs to the level of the *instantaneous motion specification and control*, and it needs provisions for the following "dynamic" primitive:

- `Motion`: this is the primitive that provides "plug-in" functionality to allow users to add motion to different parts of the Scene Graph structure, in a format that is defined outside of this *Robotics Scene Graph* standard. For example, `Motion` can add *constraints* in a `Joint` or between two or more `Frames`.

At the Robot Scene Graph representation level, `Motion` is a special type of static *semantic tag* attached to the appropriate locations in the scene, and

that *links* to concrete instantiations of "dynamics" or "computations" outside of the kinematics structure of the Scene Graph. This provision of (only) tags (with standardized meaning) provides the desired *loose coupling* between the different levels in the robot platform control: the *names* of the semantic tags can be added to the *Robotics Scene Graph* which is the subject of this section, but the *meaning* and the corresponding *functionality* should be standardized at the higher levels.

8.2.4 Geo-spatial positioning

One representation of a Robotics Scene Graph can be *instantiated* multiple times in the same world, each time with a different set of concrete *attributes* in the above-mentioned components. For example: a different location for the `Root`, or different limits for the `Joints`, etc.

8.3 COLLADA

This section explains how the most common existing formal scene graph representation standard from computer animation, COLLADA ((Barnes and Finch 2008), version 1.5) represents the primitives introduced in Section 8.2.3. The aim is not to give an exhaustive summary of the standard, but to identify its relevant primitives.

- COLLADA is a *royalty-free, open and community-driven standard*, designed for *persistent storage* of data (such as, to read or write to a file), and for automated transformations into other formats, such as: Application Programming interfaces (API), on-line message formats,...
- COLLADA is built on top of the *eXtensible Markup Language* (XML) standard for structured, human-readable encoding of information. Hence, it is supported by a rich ecosystem of software tools, tutorials and documentation.
- COLLADA has several primitives built in, that are directly useful for robotics, such as rotation and homogeneous transformations, rigid bodies, joints and kinematic models.
- COLLADA supports quite a lot of features that make sense in 3D computer animation, but much less in robotics. For example, material surface properties and textures, light and shader parameters.

  These properties *are* relevant in a robotics context, when one wants to simulate light-based sensors, such as cameras or laser range finders;

however, this use case is too specialised, and too difficult to standardize *exhaustively*, such that it is best not to include it in a standard proposal for robotics platforms.

- COLLADA's Scene Graph is a *Directed Acyclic Graph* (DAG), which means that it can be traversed in a serialized way. Such serializations are used to define the order in which elements in the Scene Graph are "visited".
- COLLADA separates the visualization, kinematics and dynamics of the scene graph structure.
- COLLADA provides mechanisms for *extensions* and *profiles*, which provide the opportunity to create domain-specific customization without loosing the advantage of a common core standard.
- The `Universal Identifier` of a Robotic Scene Graph is not *explicitly* part of COLLADA, since it is a standard built on top of the XML standard, and the concept of an *URI* (*Universal Resource Identifier*) is already well established there. However, the *format* and the *interpretation* for the `Universal Identifier` is to be defined by the application domain.

The major goal of COLLADA is to form a basis for *common data exchange* among 3D applications. It seems to have succeeded in this ambition, since all major 3D computer animation packages provide COLLADA import and export. Here are the COLLADA features that correspond to the list of requirements of Section 8.2.3:

- `Universal Identifier`: the `<asset>`: `<title>`, `<unit>` can be used to create a Universal Identifier for the Scene Graph, the `title` should contain the local identifier `robot_platform_id`, the `unit` should contain the unique `lab_id`.
- `Scene Graph`: is represented in COLLADA by the `<scene>` primitive. It can contains one or more instances of a `<physics_scene>`, one instance of a `<kinematics_scene>` and one instance of a `visual_scene`. The physical scene contains one or more `<physics_model>`s, that represent spatial configurations of rigid bodies with inertial parameters and the shape of the objects for collision checking; the kinematics scene contains one or more `<kinematic_model>`s that each represent the *geometry* of one kinematic structure, with (mass-less) segments and joints, and with `attachments` representing frames on a link to which other objects can be attached. The visual scene contains the information about how the objects should be visualised. Both the visual scene and the kinematic scene can reference to the visual scene to make the visual scene move.

- `Root`: there can be only one `<scene>` element in a COLLADA model. This is the root element of the scene graph. A `scene` contains the corresponding decoupled physics, visual and kinematic scene. The COLLADA scene graph data structure in XML is a *Directed Acyclic Graph* (DAG). Any closed loops are explicitly "cut" in their respective scene.
- `Frame`: the objects in the scene (including those in a `kinematic_model`) can be given motion-oriented `frames`. COLLADA provides already four more specific kinds of frames on kinematic chains that are directly relevant in robotics: `<frame_origin>`, `<frame_tip>`, `<frame_tcp>`, and `<frame_object>`, representing, respectively, the root frame of a kinematic chain, its end-point, a task-specific "tool center point" attached to the end-point, and an offset frame with respect to an object in the scene.
- `Segment`: represented by the `<link>` element in a `kinematic_model`, by a `rigid_body` in a `physics_model` and by a `node` in the `visual_scene`.
- `Joint`: represented by the `<joint>` element in a `kinematic_model`, by a `<node>` in the `visual_scene` and by a `rigid_constraint` in the `physics_model`.
- `Shape`: represented by the `<geometry>` element (with a large number of built-in shape types) in the `visual_scene` and by the `<shape>` element in the `physics_model`.
- `Tag`: represented by the `sid` of each element.
- `Motion`: motion of kinematic and physics models is supported via various elements: `<animation>` (to create a time-varying signal), `<channel>` (to couple an animation to a parameter in the scene), the `<articulated_system>` to make a kinematic scene "move" and the `<force_field>` to make a physical scene move.
- Geo-spatial position of the scene is represented by the `<geographic_location>` of the `<asset>`-element of the scene. The geographic location contains `<longitude>`, `<latitude>`, and `<altitude>` attributes.

COLLADA provides domain-specific extensibility via its `<extra>` element. The COLLADA standards body stimulates application domains to submit common suggestions for further versions of the standard under the form of `profile`s for the `<technique>` element.

Multiple parsers for COLLADA files are available, also in open source, e.g., OpenCOLLADA (NetAllied Systems GmbH 2009) and COLLADA-DOM (Khronos Group 2006).

8.4 Extensions and other standards

To add extensions and other standards to a well designed basic standard is done using the proven concept of *profiles* (UML). Profiles subdivide whole domains in parts with increasing complexity or specificity, and provide matching versions of the same standard.

8.4.1 Robotics extensions to COLLADA

The following extensions to COLLADA have to be made to make the standard useful as a formal representation format for the Robotics Scene Graph:

- Definition of following semantic tags: representations for sensors, and actuators.
- Easier connection between the physics information and the kinematics information. The physics `<rigid_constraint>` maps the kinematics `joint`, but both cannot directly be linked to each other, a `visual_scene` `<node>` has to be used to connect both.

8.4.2 Constraint-based motion specification using the COLLADA standard

This section explains how the existing COLLADA standard can be used to specify instantaneous constraint-based motion as explained in (De Schutter et al. 2007). The presented approach uses the combination of the `<extra>` and `<technique>` to add the necessary, but iTaSC-specific, data.

The proposal tries to reuse the COLLADA elements as much as possible.

Kinematics scene

The top-level structure of the motion specification is the `kinematic_scene`. The kinematics scene contains the instantiation of the relevant `kinematic_model` for the robots, virtual manipulators and object uncertainties.

For the constraint-based motion specification, iTaSC, all relevant information is stored in the `<extra>`-tag of the kinematics scene.

An iTaSC profile is defined for the `<technique>`-tag of the extra information of the kinematics scene:

```
<kinematics_scene>
 <extra>
  <technique profile="iTaSC" xmlns="http://url-to/iTaSC-
      Schema">
  <!-- iTaSC specific data />
  </technique>
 </extra>
</kinematics_scene>
```

As explained in Section 1.2.1 the constraint-based motion specification starts with creating virtual kinematic loops by creating virtual kinematic manipulators between the object frames of interest.

The description of the virtual kinematic manipulators is done using the standard COLLADA `kinematic_model` element.

Object frame providers The kinematic models of the robots and objects that provide object frames that will be controlled using iTaSC are instantiated in the `objects` element. For each `robot` or `object` element the respective kinematic model is instantiated and if necessary the values of available parameters of the model are set. The object frames of interest for the motion specification are defined using the iTaSC-specific `frame_object` element, every frame gets a unique `name` and is linked to a specific `link` in the kinematic model of the `robot` or `object`. All robots and objects that are involved in the iTaSC specific motion specification have to be instantiated in the `objects` element.

Virtual kinematic loops Virtual kinematic `loops` are created between the available object frames. Every `loop` element instantiates a `kinematic_model` of a virtual manipulator or a `interaction_model` of a sensor that closes a virtual loop between two object frames. Every loop has a `connect`ion between `root` link and `end` link of the respective model to the `target` object frames.

Constraints The constraints are defined by selecting the `output` from the available joint axes of the kinematic models of the robot and the joint axes of a virtual manipulators or parameters of an interaction model. The output equation is defined using a `formula` in terms of the available model parameters. The controller is defined using a `formula`. Both formulas can be described using MathML (W3C Math Working Group).

Uncertainties An `uncertainty` is defined by selecting the appropriate `axis` of the joint which models the uncertainty. The `proces_model` is defined using a `formula`, which, again, can be described using MathML.

Below is a small example on what a complete constraint-based motion specification could look like:

```
<kinematics_scene sid="example">
<technique profile="iTaSC" xmlns="http://url-to/iTaSC-Schema
    ">
 <objects>
  <robot sid="robot1">
   <instance_kinematic_model url="#ROBOT_1">
   <frame_object sid="base" link="robot/model/base" />
   <frame_object sid="endeffector" link="robot/model/
       endeffector" />
   <root target="example"/>
  </robot>
  <object sid="object1">
   <translate> 1.0 2.0 3.0</translate>
   <frame_object sid="" link="root" />
   <root target="example/>"
  </object>
 </objects>
 <loops>
  <loop sid="loop1">
   <instance_kinematic_model sid="virtual_1" url="#VIRTUAL_1
       "/>
   <connect>
    <root target="object1/root" link="virtual/rootlink" />
    <end target="robot1/endeffector" link="virtual/endlink"
        />
   </connect>
  </loop>
  <loop sid="loop2">
   <instance_interaction_model sid="camera_1" url="#CAMERA_1
       "/>
   <connect>
    <root target="robot1/endeffector" link="camera_1/root" /
        >
    <end target="object1/root" link="camera_1/end" />
   </connect>
  </loop>
 </loops>
 <constraints>
  <constraint sid="distance" >
   <new_param id="output1"/>
   <controller level=VELOCITY sid="controller1">
    <instance_formula>
     <set_param ref="target">
      <connect_param ref="distance.output1"/>
     </set_param>
```

```
    <set_param id="source">
     <connect_param ref="loop1.virtual1.axis3"/>
    </set_param>
   </instance_formula>
  </controller>
 </constraint>
</constraints>
<uncertainties>
 <uncertainty sid="thickness" >
  <output axis="virtual/joint_thickness"/>
  <instance_process_model>
   <!-- process model details omitted/>
  </instance_process_model>
 </uncertainty>
</uncertainties>
</technique>
```

8.4.3 Compatibility with other robotics standards

The formal representation for a Robotics Scene Graph standard presented in the previous sections (including the suggested extensions to COLLADA) does not, of course, cover all aspects that are relevant to current and future robot control systems, as discussed in Sec. 8.2.2.

To start with, it does not provide any information on how the robots in the scene can be interfaced via computer APIs. This requires, among other things, a standardization of robot *controller* representations. This interface must be limited to the properties that the standard can represent on joints and segment, hence: position, velocity, acceleration, force. Non-instantaneous motion control is the subject of other standards. This means that multiple standards have to be used, and hence must be compatible with each other, to model an entire robotic application.

AutomationML (AutomationML 2008) is being developed for data exchange in large-scale automation systems. AutomationML already includes COLLADA as a sub-standard for representing the kinematic models.

8.5 Conclusion

This chapter presents a motivated discussion about to what extent the COLLADA standard, from the domain of computer animation, is appropriate to serve as formal representation of a *Robotics Scene Graph* standard, for the

persistent storage of the kinematic structure properties of robotics platforms. Section 8.4.2 is a first attempt to include constraint-based motion specification in the COLLADA format.

All core features of the presented *Robotics Scene Graph* standard are available in the COLLADA standard: scene graphs, kinematic structures (including their dynamics), and geo-spatial locations. But the standard contains already more than what the robotics community needs, *and* the standard is well known in the robotics community; hence, there is the natural tendency to create a robotics-specific standard. This chapter motivates why choosing for adoption of an already existing, though not perfectly fitting, standard is the better choice. The advantages of adopting one of these standards (besides the already available relevant features) are that

- they have a large community (much larger than the robotics community!) to support tooling and documentation, and
- the integration into 3D visualisation and programming tools (such as Blender (Roosendaal 2002)) is made easy.

The disadvantage is that the computer animation community has put (and will keep on putting) a significant amount of items into the standards that have no direct use for robotics (e.g., interpolation on colors), while some highly needed items in robotics (e.g., force controlled motion) are not very relevant in computer animation.

The first stakeholders to discuss and adopt the suggested *Robotics Scene Graph* standard are the larger *open source* software projects: ROS (Willow Garage 2009), Player (Gerkey et al. 2003; Gerkey et al. 2007), Orocos (Soetens and Bruyninckx 2005; Bruyninckx 2001), and Orca (Brooks et al. 2008; Brooks et al. 2004). ROS has already defined its own *Universal Robot Description Format* (URDF), the scope and contents of which are overlapping very much with the suggested *Robotics Scene Graph* standard. The open source 3D computer animation program Blender (Roosendaal 2002) already supports the COLLADA standard (Barnes and Finch 2008). The same holds for Europe's major robot manufacturers, ABB and KUKA, that are founding members of the AutomationML initiative (AutomationML 2008) which contains the COLLADA standard for similar purposes as discussed in this chapter. All these elements together bode well for the potential successful adaption of the suggested robotics platform standard.

Chapter 9

Kinematics and dynamics software library for robotics

We can only see a short distance ahead, but we can see plenty there that needs to be done.

Alan Turing

This chapter introduces the different design issues which should be taken into account for the design of a kinematics and dynamics software library (*KDL*). First the different levels of use of *KDL* are identified and the functionality of each level is defined. *KDL* only implements the library functionality which only covers the mathematical and program language representation levels of the kinematics and dynamics structures and algorithms. The natural classes of *KDL* are introduced. Most of the desired functionality is implemented in the Orocos subproject *KDL* which is available as an open-source C++ library under the GNU LGPL license at `http://www.orocos.org/kdl`.

9.1 Introduction

The software framework presented in Chapter 7 as well as a lot of other robotic control software frameworks need sophisticated kinematic and dynamic algorithms which can be reused for different models of kinematic structures.

A lot of this functionality can be provided by a general physics library, such as Bullet (Coumans 2005) or ODE (Smith 2000). But these libraries do not take advantage of the joint connection between the concatenating bodies that usually exist in robotics, machine tools and bio-mechanics.

9.1.1 Related work

The robotics toolbox (Corke 1996) for Matlab provides many functions that are useful in robotics such as kinematics, dynamics, and trajectory generation. But it does not support branching structures. It contains a lot of kinematics and dynamics algorithms but it is not easy to get these into the hard realtime control loop of a robot control system.

Roboop (Gourdeau 1997) is a `C++` robotics object oriented programming toolbox which is released under the LGPL license. It contains a lot of kinematics and dynamics algorithms. Roboop however does not have realtime use of its algorithms in mind and only aims for simulation. As such a lot of the algorithms are unsuitable to use in a hard realtime control loop. It also does not support branching structures.

9.1.2 The OROCOS Kinematics and Dynamics Library, KDL

This chapter presents a software library that offers a generic implementation of various kinematic and dynamic algorithms. For each of these implementations special care is taken (i) to use a generic API, such that using different algorithm implementations for the same purpose only changes the configuration procedure but not the usage of the algorithm in the running state of a robot control system; and (ii) to make all algorithm implementations safe to use in a hard realtime control setting.

The presented library is developed as part of the OROCOS project under the name of the *Kinematics and Dynamics Library* or *KDL*. It is already widely used in various robotics research labs and is a part of ROS (Willow Garage 2009), the Robot Operating System which is becoming the most used (open source) software system for robotics research.

9.2 Three levels of use

There are three levels in using *KDL*:

- *libraries* providing the raw class functionalities,
- *application architectures* ("templates") providing active or passive objects with a specific *Quality of Service*, and
- *end-user applications*. Between every level there is a 1-to-N relation: the same library is used in N templates, and each single template can be the foundation of N different end-user applications. *KDL* implements the functionality needed at the library level.

9.2.1 Libraries

Libraries contain data structures and algorithms, offered as *classes* in the object-oriented programming approach. A "full" *KDL* will consist of the following family of complementary libraries:

- kinematics and dynamics *transformations on rigid-body structures*, basically the classes representing open lumped parameter rigid body structures, and executing their *instantaneous* forward and inverse kinematics and dynamics transformations;
- *motions* which are passive objects, offering representation, interpolation, configuration, etc. of *non-instantaneous* motion of rigid body structures;
- motion *tasks* which are active objects, with state machines, events, different threads, persistence, etc.

At the motion task level, the OROCOS context of KDL comes only into the picture in the form of paying attention to the *real-time requirements* of the transformations on rigid body structures, and of the motion interpolation. So, each library will consist of a *real-time* sub-library and a *non-real-time* sub-library: the former contains only non-blocking methods with deterministic execution time, which means memory allocation is avoided except at creation time; execution time should be predictable and deterministic; and a thread-safe implementation is provided. Examples of methods in the former are the *factory* methods to create a rigid body structure from more primitive structures: such a factory object will undoubtedly have to do memory allocation after it has been created; therefore creation can take unpredictably long; the factory should typically be a "singleton" object for which thread-safety is not guaranteed.

The core part of *KDL* are objects for *6D geometry* calculations: reference frames, 6D motion and force vectors; the mappings between the "motion" space and the "force" space: stiffness and compliance, damping and inertia and all operations to transform these objects to different reference frames and different reference points.

9.2.2 Application architectures

Application architectures (in the form of *templates*; not to be interpreted in the C++ STL meaning of this word!) offer a "proven" way of solving particular families of problems. These templates are typically of the above-mentioned *task* type extended with features such as name serving and a scripting language, but one could also imagine a *kinematics server* application that provides the service to calculate a set of kinematic transformations for their clients.

This is the level where the OROCOS RTT subproject delivers the supporting infrastructure: activities, execution flow, data flow, events, name serving, etc. Different application families require different architectures, but they can all use the same application template composed of the same RTT components.

Ideally the *application architectures* are independent of the explicit type of rigid body structure and exact environment setup.

9.2.3 End-user applications

End-user applications are filled-in *templates*, each having its own application-specific programming language, terminology, etc.

This is the level where the OROCOS RTT scripting comes in, to translate the architecture API (methods, commands, events, data-ports, properties) into easy to understand language constructs that are "natural" for one particular set of end users.

9.3 Four levels of representation

There are four levels of representation (Prassler et al. 2009) in *KDL*:

1. *Ontology representation,*
2. *Mathematical representation,*
3. *Program language representation,*

4. *Native hardware representation.*

The "mapping" between levels is 1-to-N: for each level L_i, there exist various possibilities at level L_{i+1}. The *ontological* and *native hardware* levels are often not formalized, and, at best, exist implicitly in the domain. Since the trend in modern software systems is towards larger-scale, multi-domain and multi-platform integration, the lack of these two levels introduces more and more semantic and implementation ambiguities. The following sections explain and motivate these four levels into more detail.

9.3.1 Ontology representation

In order to represent a certain domain, the *meaning* (semantics) of all concepts and operations in that domain should be made absolutely clear. The practical problem is that one particular library provides functionalities that are useful in more than one domain, and that these domains can have very different terminology for the same physical concepts. For example, *KDL* has its origins in *robotics*, but can also support the *computer animation* domain, where a "kinematic chain" becomes an "armature", a "via point" becomes a "key frame", etc. This level of representation is currently not dealt with in *KDL*, and subject to initial initiatives with major projects like ROS.

9.3.2 Mathematical representation

In most engineering systems, the physical concepts used in the system are represented in a mathematical form. Typically, multiple mathematical representations are possible for the same physical concept. For example, the position of a robot can be represented mathematically by a *homogeneous transformation matrix*, or by the combination of a *position vector* and a set of *Euler angles*. The practical problem is that even these "standardized" representations are less standardized than they appear: Which one in the set of twelve possible Euler angles is used? Is the homogeneous transformation matrix representing a right-handed or left-handed reference frame? Etc.

These mathematical representation ambiguities come on top of the above-mentioned ontological difficulties. And, in addition, there is the extra problem of defining what physical units are used to represent the various concepts. Meters, millimeters or inches? Radians or degrees? Seconds or hours? Etc.

9.3.3 Program language representation

Even after having selected (and semantically documented) any particular mathematical representation, the problems are not yet over. Now the way how the mathematical primitives are to be encoded in computer program languages has to be chosen:

- Is a position coordinate represented by a "float", a "double", a "short integer", or a "signed integer"?
- Is a matrix to be stored in "column major" or in "row major" format?
- Are arguments in function calls accessed "by value" or "by reference"?
- ...

A library that neglects these program language representation aspects has a high risk of causing *very subtle* incompatibilities between various implementations, especially on differing program languages, compilers, or middle-wares.

9.3.4 Native hardware representation

Library makers must also take into account the variations in hardware platforms to the extent that they can influence the practical value of a standard:

- Are the meanings and implementation of, for example, "float" and "double" the same on all possible architectures?
- Can "enumeration types" flow over when porting an implementation from a 32 bit CPU to a deeply embedded application using 16 or 8 bit CPUs?
- Is there an influence of the byte order ("big endian" or "little endian") of the hardware architecture?
- ...

These aspects are even more subtle than in the case of program language representation. They are certainly to be taken into account when the standardization process involves the definition of *binary* file or message formats for storing and exchanging information between different components in a large-scale software system.

9.4 The natural classes in KDL

This section describes which are the natural classes in *KDL*, that is, which real-world objects and functionality must be represented, and how are these objects and functionality separated via maximally decoupled interfaces. The *rigid body structures* (consisting of ideal, *lumped parameter* components, with only the basic "lower kinematic pair" joint types: revolute, prismatic, cylindrical, screw, planar and spherical) are the "real" (i.e., physical) objects to be represented in *KDL*. Any specific *rigid body structure object* is then the *aggregation* of these "lower level" classes described in this section.

9.4.1 The segment

The basic element of each kinematic system is the *segment*, a segment consists of a *joint* and a *rigid body* ("link").

- The joint has a *type* (rotational, translational, spherical, elbow, none, etc.) and has its own joint-space dynamic *properties*: stiffness, damping, inertia, transmission ratio.
- A body has a *base-frame* specifying the connection point of the joint and stores its dynamic *properties*.

9.4.2 Topological family

Three *topological families* of rigid body structures exist:

- serial chains,
- trees, and
- graphs.

Each of these three topological families has its own *generic* representation and *numerical* solver algorithms. Of course, with possibly different implementations for each family, but always using the same concepts that are not used in a simpler topological family.

When used in a *motion* task, each rigid body structure (of whatever topological family) has a *state* which encodes the history of the motion. In the broader context of *KDL*, this state will have to be shared with other software components such as controllers and sensors. So, the states are separate objects, that can be exchanged in a realtime data flow.

9.4.3 Kinematic family

Within each topological family, various specialized *kinematic families* exist. A kinematic family is a specific *parameterized geometry* within one of the generic topological families, for which a much more efficient solution exists than the generic numerical solution in that topological family. (Think about the specific geometry of industrial robot arms, and about parallel kinematic manipulators which are very special graphs.) Of course, the same efficient solution can have multiple variations in itself, or more than one efficient solution can exist. The selection and configuration of these variations is done via the "property" members of the specific class.

9.4.4 Generic numerical solver

Different generic numerical solvers exists for each topological family. These solvers use the general description of the kinematic system and offer transformations of positions, velocities, accelerations and forces between the Cartesian space and the joint space of the kinematic system. Each generic solver class has hence its own specific property members.

9.4.5 Virtual classes

The discussion in the previous sections leads to the following summary of basis classes that *KDL* must offer. There are separate classes for each of the *domains* that are relevant to *KDL*:

1. *Force* domain ("`Force`"): representation of force and torque.
2. *Position* domain ("`Pos`"): only the position and the force–position dynamic property (stiffness) is represented.
3. *Velocity* domain ("`Vel`"): both the position and the velocity, together with their related dynamic properties (stiffness, damping) are represented.
4. *Acceleration* domain ("`Acc`"): position, velocity and acceleration, together with their related dynamic properties (stiffness, damping, inertia) are represented.

Of course, the domain must map in the *joint space* and *Cartesian space* parts of the kinematic chains. The reference frame transformations of all these domain classes are encoded in the *Geometry* part library of *KDL*.

So, here is the complete list of virtual classes:

- **Joint**: each *specific* type of joint class inherits from this virtual class, and has the following properties: type of the joint (enum-type), transmission information, and the domain-specific dynamic properties stiffness, damping (i.e., friction) and inertia.
- **Body**: this class contains two frames (*base frame* offset and the *end frame* offset with respect to the *base frame frame*) and the domain-specific dynamic properties.
- **Segment**: the basic building block for a kinematic chain. It has the following members: (i) a *Joint* and (ii) a *Body*.
- **Chain**: this class collects the information about the interconnection of the segments which are used to build this topological family. It also offers the factory functionality to build a serial chain out of segments.
- **Tree**: this class extends the Chain to represent the topological family of branching kinematic structures.
- For each particular **topological family**:
  - **JointArray**
  - **KeyframeArray**

  These arrays can contain any kind of information: position, velocity, acceleration or force. The arrays for a given kinematic structure have no meaning for another kinematic structures (different zero points and offsets, different positive direction, different limits, different kind of joints, etc.).
- One class for each set of specific **transformation algorithms** between joint space and Cartesian space.

  The *semantics* for all the transformation algorithms (and hence their generic API templates, such as **JntToCart()**) are easy to describe in a way that is independent of the kinematic family, but the specific implementations *are* family dependent, as well as the *properties* that belong to each specific implementation. In all, each kinematic family can eventually have a very large number of different transformation algorithm classes.

  The main types of transformations:

  - **Inverse Kinematics**. This class takes a kinematic system as input argument and offers methods to calculate the inverse transformations output. Each alternative solver (for instance at the

velocity domain: DLS (Wampler 1986), SDLS (Buss and Kim 2005), SRI (Nakamura and Hanafusa 1986),...) has his own class with its own algorithm specific properties.

– **Forward Kinematics**. This class takes a kinematic system as input argument and offers methods to calculate the forward transformations output.

These classes also have a varying number of "derived" classes, with the same numerical content but another physical, task-dependent or algorithmic meaning: joint angle positions, velocities, forces, etc.; desired values, measured values, estimated values, etc.; constraints; joint stiffness, damping, inertia, etc.; etc. Adding this large (but bounded) set of derived classes has only one advantage: semantic checks become possible; this advantage grows in importance with growing size and scope of the applications built with *KDL*.

9.5 Conclusion

This chapter discusses the design issues for a kinematics and dynamics library for robotics. First of all three different levels of use are identified, the library, application template and end-user application level. The presented kinematics and dynamics library *KDL* only offers functionality for the lowest level: the library. Four different levels of representation are identified, the ontological, mathematical, programming language and hardware representation.

The implementation of *KDL* offers a specific programming language representation for a specific chosen mathematical representation of the kinematics and dynamics for robotic structures. More details about the specific representation choices can be found on `http://www.orocos.org/kdl/User_Manual`.

KDL contains a generic forward recursive position and velocity solver which can be used for both serial and tree-like kinematic structures. Two solvers exist for the inverse velocity kinematics: one based on the generalised inverse of the kinematic structure's jacobian (Ben-Israel and Greville 1980), and another based on the damped least squares solution of the inverse (Wampler 1986). Both inverse velocity solvers use a singular value decomposition (SVD) of the jacobian. *KDL* offers two algorithms for the SVD which can be used in a realtime setting, one based on HouseHolder rotations (Golub and Van Loan 1989) and one based on Givens rotations (Maciejewski and Klein 1989). The latter uses the previous decomposition of the jacobian as a starting point for the next decomposition. Since the jacobian does not change a lot in a discrete control loop this new decomposition is very fast. For the inverse dynamics *KDL* offers the recursive Newton-Euler algorithm as it can be found in (Featherstone

2008). *KDL* is available under the GNU LGPL open-source license as part of the Orocos project (Bruyninckx 2001).

Chapter 10

General conclusions

't Is gebeurd.

Eric van Looy

10.1 Main contributions

This thesis contributes to the dissemination of the iTaSC framework by providing a methodology and software support for practical implementations of constraint-based motion specification, estimation, control and coordination.

This thesis introduces *skills* as an application domain-invariant "best practice" to coordinate the execution of the motion specification. The *Skill* deserves a prominent place in the context of robot control systems. If the robot will be taken out of its industrial work-cell and be introduced in the dynamic domestic environment, coordination will be the key concept to cope with constantly changing environments and desired motions. This thesis explains how to add the first level of *Skill* functionality to current industrial-

grade robot programming languages based on iTaSC, the constraint-based motion specification methodology presented by De Schutter et al. 2007. To decouple the continuous level of motion specification, control and estimation from the discrete level of motion coordination, a formal interface to build, configure and coordinate an iTaSC motion specification is presented. The modularity and power of the *Skills* is illustrated by an involved sensor-based multi-robot application in which two robots have to cooperate in a force-controlled manipulation task while making sure they keep an eye on the closest person and keep a safe distance from this person. The example application shows how modular control and estimation *Skills* can be specified separately and built together creating a sophisticated robot task. To allow maximal reusability and composability of skills, the constraint-based approach from iTaSC has to be used as the underlying framework for the coordination of motion. Finite state machines enable the coordination of the motion specification in which the monitoring of the constraints is the key to trigger the transitions to different states with different motion configurations.

This thesis also extends the underlying iTaSC methodology to enable the definition of motion constraints not only on the geometric auxiliary coordinates of the virtual manipulators defined by the original iTaSC methodology of (De Schutter et al. 2007), but also directly on coordinates from any feature space. This thesis uses an *Interaction model* which relates the motion between two objects to the feature coordinates of interest to accomplish this. The *Interaction model* creates a virtual linkage which can be treated the same way as the virtual kinematic loops from the original iTaSC methodology. It however represents the inverse relation between the feature coordinates and the pose between the objects. Therefore it does not need to include six independent auxiliary coordinates. The original iTaSC specification methodology can be seen as a special case of the presented extension where at the velocity level the inverse of the feature jacobian, $\boldsymbol{J_f}^{-1}$ is equal to the Interaction matrix $\boldsymbol{H}$ of the presented extended iTaSC formulation. Because the *Interaction model* models the inverse relation, it does not need to be built up from six independent auxiliary coordinates to make sure it is invertible. The extended iTaSC methodology allows specifying and controlling constraints in different spaces, such as image space, Cartesian space, joint space or any other task or sensor space without adding complexity in the control of the robot. The extended methodology is illustrated with a visual servoing task, which combines constraints on features in the image space with other task-related constraints in the Cartesian and joint space.

Calculating the robot motion from the specified constraints can be very computationally expensive in systems which include a lot of actuated joints, such as humanoid robots. Therefore the quest for efficient algorithms is still ongoing. This thesis discusses one of the most ground-braking algorithms

for acceleration-based constrained hybrid dynamics (Vereshchagin 1989) while this work contains a fully recursive, linear-time, inverse and forward dynamics algorithm with partial acceleration constraints, it never got the attention it deserved. Even so, a counterpart has, to the best of the author's knowledge, never been developed. **This thesis extends the work of Vereshchagin 1989 for tree-like kinematic structures and explains how it can be used to solve an extended iTaSC motion specification.** The recursive algorithm uses Gauss's Principle of least constraint as the optimization metric at the acceleration level and realizes the minimization of the "acceleration energy". The algorithm is fully compatible with the iTaSC methodology and offers a recursive solution instead of the matrix-inversion method for Eq. (3.27). A `C++` implementation of the algorithm is under development as part of the kinematics an dynamics library (*KDL*) in the *OROCOS* project, (Bruyninckx 2001).

Creating software for multi-sensor, multi-robot applications from scratch for each new robotic system, or new application is a very involved task. Therefore reusable software components and formal representations of both the robotic system and the task to be executed are mandatory. **This thesis presents a component-based modular architecture for the next generation of robot controllers based on the extended iTaSC motion specification and coordination.** It implements the extended iTaSC methodology to address the issues in motion specification, sensing and world modeling. *Skills* are used to address the issues in motion coordination. The presented architecture enables the mandatory decoupling between the actual robotic system hardware and the control software architecture that determines the robotic system's desired motion. By using a component-based architecture which decouples the different information sources in the iTaSC motion specification, the developed specific components become highly reusable in different application settings, and can be used in a distributed system. The actual implementation is based on the Orocos project (Bruyninckx 2001), which is a very powerful programming framework for the execution of hard realtime control tasks, and allows the programmer to define application-specific interfaces for less experienced users.

To allow maximal reuse of motion specifications and skills which are built on top of them, the formal representation of a constraint-based motion specification is important. **This thesis makes the case for a formal representation of the "Robotics Scene Graph" in an existing standard file format, COLLADA which is commonly used in the 3D interactive industry.** The "Robotics Scene Graph" covers (i) the representation of kinematic and dynamic (lumped parameter) models of the *robots*, (ii) the possibility to attach to these robots sensors, motors, tools, shape, and task-relevant reference points or frames and (iii) the possibility to locate the robots with respect to objects in the world (which often are other kinematic models) that are relevant to the task.

A formal representation of the constraint-based motion specification for the "Robotics Scene Graph" based on the COLLADA standard is presented which uses the "extension" mechanism of the COLLADA standard to add the necessary information and tags to build the entire component-based software architecture for a specific application from a single standardized file.

The presented software framework as well as a lot of other robotic control software frameworks need powerful kinematic and dynamic algorithms which can be reused for different models of kinematic structures. **This thesis presents a software library that offers a generic implementation of various kinematic and dynamic algorithms.** For each of these implementations special care is taken (i) to use a generic API, such that using different algorithm implementations for the same purpose only changes the configuration procedure but not the usage of the algorithm in the running state of a robot control system; and (ii) to make all algorithm implementations safe to use in a hard realtime control setting.

10.2 Outlook

10.2.1 Methodological developments

In the shift from industrial robotics to domestic robotics not only the coordination of the desired motion in dynamically changing settings is important. Also the perception of what is happening in the neighborhood of the robotic system is essential. The underlying extended iTaSC methodology does not only support motion control specification but also estimation of modeled uncertainties. Hence the concept of *Skills* does not only map to motion coordination: the same concept should be used to coordinate the estimation and perception calculations and actions in the application.

Learning *Interaction models* at the continuous level as well as learning coordination *Skills* based on the proposed methodologies is very important to learn modular motion specifications and coordination. Applying the constraint-based approach to motion specification and motion coordination learning methodologies will result in learned motion configurations with a higher composability and reusability than learned motion configurations from overspecified learning problems.

The constraint-based motion specification methodology can include dynamic relations between two objects instead of just geometric relations. These dynamic relations can for example be the process models of a deburring or milling task. How to include these process models in the iTaSC methodology still has to be investigated.

While the presented methodologies only optimize the robot motion at each time instant, the overall resulting motion might however still be far from optimal. Therefore non-instantaneous optimization techniques should be combined with the presented ones to enable full optimization of the robot motion. (Decré et al. 2009) is a first step in enabling this kind of optimizations. Also, instead of minimizing a single objective function, more complicated multi-objective utility functions can be used to optimize the robot motion globally over time.

In the proposed software architecture all robot motion is still calculated at one central place, the *Scene*. To make decentralized coordination possible the motion intentions of other *Scene*s have to be included at the current *Scene* and have to be used as soft constraints if cooperation between multiple *Scene*s is necessary. The PROSA architecture (Van Brussel et al. 1998),(Valckenaers et al. 2003) provides a mechanism to enable this kind of coordination by providing the necessary primitives for resource allocation and intention propagation.

The standardization of the formal representation of the presented *Robotics Scene Graph* is a basis to create interchangeable motion specifications. The first stake-holders to discuss and adopt the suggested *Robotics Scene Graph* standard are the larger *open source* software projects: ROS (Willow Garage 2009), Player (Gerkey et al. 2007), Orocos (Bruyninckx 2001), and Orca (Brooks et al. 2004). The same holds for Europe's major robot manufacturers, ABB and KUKA, who are founding members of the AutomationML initiative (AutomationML 2008) which contains the COLLADA standard for similar purposes as discussed in this chapter. All these elements together bode well for the potential successful adoption of the suggested robotics platform standard.

10.2.2 Applications

The presented methodologies can be used for applications in a variety of robotic domains. In industrial robotics, skills are very useful to implement the coordination of mechanical assembly tasks of rigid body parts or objects made of flexible materials. Material removal or forming processes such as grinding, deburring, welding or sheet metal bending that are performed by robots need the skill to coordinate the robot motion in view of both process stability and the quality of the finished product, by taking into account the production process characteristics (statics and dynamics). In domestic robot applications, skills can be used to coordinate for instance household tasks such as (un)loading a dishwasher, cutting fruit, breaking an egg, etc. These tasks seem easy to do for humans but are highly complex to be performed by robots. Breaking down domestics task into their supporting skills based on the symbolic information, contained in a high-level abstract description of

the task, is one of the most important steps and challenges for this kind of applications. In biomedical applications skills can be used to coordinate the partially autonomous manipulation of surgical instruments by robots while the surgeon still maintains control over the procedure through physical human-robot interaction. Skills also provide the tools needed for dynamic registration. Other biomedical applications can be found in the area of exo-skeletons, which are multi-degree-of-freedom ortheses which have to be able to support a human during skills like walking, running, stair-climbing, etc.

10.2.3 Business opportunities

As described in (Prassler et al. 2009) a redesign of the current robotic system design process is needed, both for the hardware and for the software components.

Separating the Task controller, the Skill controller, and the iTaSC motion controller, with clear interfaces between them as described in Chapter 5, is not only an advantage from a software engineering point of view, but it creates opportunities for new commercial activities, with companies specializing in one (or more) of these three components (Gerkey 2009). For example, one company could build a business around paid subscriptions to mobile manipulation skills, providing customers with improved versions of *specifications* for the Skill controller at regular intervals, without having to know anything about how the skill is *executed* by the iTaSC motion controller.

References

Ambler, A. P., H. G. Barrow, C. M. Brown, R. H. Burstall, and R. J. Popplestone (1975). A versatile computer-controlled assembly system. *AI 6*, 129–156.

Ambler, A. P. and R. J. Popplestone (1975). Inferring the positions of bodies from specified spatial relationships. *AI 6*, 157–174.

Appell, P. (1900). Sur une forme générale des équations de la dynamique et sur le principe de Gauss. *J. für die Reine und Angewandte Mathematik 122*, 205–208.

Argall, B. D., S. Chernova, M. Veloso, and B. Browning (2009). A survey of robot learning from demonstration. *Robotics and Autonomous Systems 57*, 469–483.

Autodesk Inc. (1996). 3D Studio Max. `http://www.autodesk.com/3dsmax`. Accessed online 2 August 2009.

AutomationML (2008). Automation Markup Language. `http://www.automationml.org`. Last visited March 2010.

Barnes, M. and E. L. Finch (2008). COLLADA—Digital Asset Schema Release 1.5.0. `http://www.collada.org`. Last visited March 2010.

Ben-Israel, A. and T. N. E. Greville (1980). *Generalized Inverses: Theory and Applications* (Reprinted ed.). Huntington, NY: Robert E. Krieger Publishing Company.

Billard, A. and R. Siegwart (2004). Robot learning from demonstration. *Robotics and Autonomous Systems 47*(2-3), 65–67. Robot Learning from Demonstration.

Bolles, R. and R. Paul (1973). The use of sensory feedback in a programmable assembly system. Memo AIM-220, Artificial Intelligence Laboratory, Stanford University.

Brooks, A., W. Kadous, T. Kaupp, A. Makarenko, and A. Oreback (2004). Orca: Components for robotics.

`http://orca-robotics.sourceforge.net/`. Last visited March 2010.

Brooks, A., T. Kaupp, A. Makarenko, S. Williams, and A. Orebäck (2008). Orca: A component model and repository. In *Software Engineering for Experimental Robotics*, Springer Tracts in Advanced Robotics, pp. 231–251.

Brooks, R. A. (1982). Symbolic error analysis and robot planning. *ijrr 1*(4), 29–78.

Brooks, R. A. (1986). A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation 2*(1), 14–23.

Brooks, R. A. (1991, August). Intelligence without reason. In R. Myopoulos, John; Reiter (Ed.), *Proceedings of the 12th International Joint Conference on Artificial Intelligence*, Sydney, Australia, pp. 569–595. Morgan Kaufmann.

Brugali, D. and P. Scandurra (2009, December). Component-based robotic engineering (part i) [tutorial]. *IEEE Robotics and Automation Magazine 16*(4), 84–96.

Brugali, D. and A. Shakhimardanov (2010, march). Component-based robotic engineering (part ii). *Robotics Automation Magazine, IEEE 17*(1), 100 –112.

Bruyninckx, H. (2001). Open Robot COntrol Software. `http://www.orocos.org/`. Last visited 2010.

Bruyninckx, H. and J. De Schutter (1996). Specification of force-controlled actions in the "Task Frame Formalism": A survey. *IEEE Transactions on Robotics and Automation 12*(5), 581–589.

Bruyninckx, H., S. Demey, S. Dutré, and J. De Schutter (1995). Kinematic models for model based compliant motion in the presence of uncertainty. *The International Journal of Robotics Research 14*(5), 465–482.

Bruyninckx, H. and O. Khatib (2000). Gauss' Principle and the dynamics of redundant and constrained manipulators. In *Proceedings of the 2000 IEEE International Conference on Robotics and Automation*, San Francisco, CA, pp. 2563–2568. ICRA2000.

Bruyninckx, H., P. Soetens, and B. Koninckx (2003). The real-time motion control core of the Orocos project. In *Proceedings of the 2003 IEEE International Conference on Robotics and Automation*, Taipeh, Taiwan, pp. 2766–2771. ICRA2003.

Buss, S. R. and J.-S. Kim (2005). Selectively damped least squares for inverse kinematics. *Journal of Graphics Tools 10*(3), 37–49.

Conticelli, F. and B. Allotta (2001). Discrete-time robot visual feedback in 3D positioning tasks with depth adaptation. *IEEE/ASME Transaction on Mechatronics 6*(3), 356–363.

Corke, P. (1996, March). A robotics toolbox for MATLAB. *IEEE Robotics and Automation Magazine 3*(1), 24–32.

Coumans, E. (2005). Bullet physics library. `http://bulletphysics.org/`. Last visited March 2010.

Dassault Systèmes (1984). Catia. `http://www.3ds.com/products/catia/`. Accessed online 5 September 2009.

De Laet, T., H. Bruyninckx, and J. De Schutter (2010). Shape-based online multitarget tracking and localization algorithm for targets causing multiple measurements: Variational bayesian clustering and lossless data association. Internal report 10RP003, Department of Mechanical Engineering, Katholieke Universiteit Leuven, Belgium.

De Laet, T. and J. De Schutter (2007). Control schemes for constraint-based task specification in the presence of geometric uncertainty using auxiliary coordinates. Internal report 07RP001, Department of Mechanical Engineering, Katholieke Universiteit Leuven, Belgium.

De Laet, T., W. Decré, J. Rutgeerts, H. Bruyninckx, and J. De Schutter (2007). An application of constraint-based task specification and estimation for sensor-based robot systems. In *Proceedings of the 2007 IEEE/RSJ International Conference on Intelligent Robots and Systems*, San Diego, California, pp. 1658–1664. IROS2007.

De Sapio, V., O. Khatib, and S. Delp (2006). Task-level approaches for the control of constrained multibody systems. *Multibody System Dynamics 16*(1), 73–102.

De Schutter, J., T. De Laet, J. Rutgeerts, W. Decré, R. Smits, E. Aertbeliën, K. Claes, and H. Bruyninckx (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.

De Schutter, J. and H. Van Brussel (1988a). Compliant robot motion I. A formalism for specifying compliant motion tasks. *The International Journal of Robotics Research 7*(4), 3–17.

De Schutter, J. and H. Van Brussel (1988b). Compliant robot motion II. A control approach based on external control loops. *The International Journal of Robotics Research 7*(4), 18–33.

Decré, W., T. De Laet, J. Rutgeerts, H. Bruyninckx, and J. De Schutter (2007, September). Application of a generic constraint-based programming approach to an industrial relevant robot task with uncertain

geometry. In *IEEE International Conference on Computer as a Tool*, Warsaw, Poland, pp. 2620–2626.

Decré, W., R. Smits, H. Bruyninckx, and J. De Schutter (2009). Extending iTaSC to support inequality constraints and non-instantaneous task specification. In *Proceedings of the 2009 IEEE International Conference on Robotics and Automation*, Kobe, Japan, pp. 964–971.

Diankov, R. and J. Kuffner (2008, July). Openrave: A planning architecture for autonomous robotics. Technical Report CMU-RI-TR-08-34, Robotics Institute.

Dillmann, R., M. Kaiser, and A. Ude (1995). Acquisition of elementary robot skills from human demonstration. In *Proceedings of the International Symposium on Intelligent Robotic Systems*, Pisa, Italia, pp. 185 – 192.

Doty, K. L., C. Melchiorri, and C. Bonivento (1993). A theory of generalized inverses applied to robotics. *The International Journal of Robotics Research 12*(1), 1–19.

Ejiri, M., T. Uno, H. Yoda, T. Goto, and K. Takeyasu (1971). An intelligent robot with cognition and decision-making ability. In *Proceedings of the Second International Joint Conference on Artificial Intelligence*, London, UK, pp. 350–358.

Ernst, H. A. (1962). Mh-1, a computer-operated mechanical hand. In *AIEE-IRE '62 (Spring): Proceedings of the May 1-3, 1962, spring joint computer conference*, New York, NY, USA, pp. 39–51. ACM.

Espiau, B., F. Chaumette, and P. Rives (1992). A new approach to visual servoing in robotics. *IEEE Transactions on Robotics and Automation 8*(3), 313–326.

Euler, L. (1776). *Leonhardi Euleri Opera omnia. Series 2: Opera mechanica et astronomica*. Fussli Zurich. exact date unknown.

Featherstone, R. (1983). The calculation of robot dynamics using articulated-body inertias. *The International Journal of Robotics Research 2*(1), 13–30.

Featherstone, R. (2008). *Rigid Body Dynamics Algorithms*. Springer.

Feldman, J. A., G. M. Feldman, G. Falk, G. Grape, J. Pearlman, I. Sobel, and J. M. Tenebaum (1969). The stanford hand-eye project. In *Proceedings of the First International Joint Conference on Artificial Intelligence*, Washington, D. C., pp. 521–526.

Finkemeyer, B. (2004). *Robotersteuerungsarchitektur auf der Basis von Aktionsprimitiven*. Ph. D. thesis, Institut für Robotik, TU Braunschweig.

Finkemeyer, B., T. Kröger, and F. M. Wahl (2005). Executing assembly tasks specified by manipulation primitive nets. *Advanced Robotics 19*(5), 591–611.

Friedrich, H., O. Rogalla, and R. Dillmann (1998). Integrating skills into multi-agent systems. *J. of Intelligent Manufacturing 9*(2), 119–127.

Gantmacher, F. (1975). *Lectures in Analytical Mechanics*. Moscou: MIR.

Gauss, K. F. (1829). Uber ein neues allgemeines Grundgesatz der Mechanik. *J. für die Reine und Angewandte Mathematik 4*, 232–235.

Gerkey, B. (2009). Towards a robot app store [pdf document]. Retreived online from `http://www.scribd.com/doc/17384278/Towards-a-Robot-App-Store`. Last visited 2010.

Gerkey, B., R. Vaughan, A. Howard, and N. Koenig (2007). The Player/Stage project. `http://playerstage.sourceforge.net/`.

Gerkey, B., R. T. Vaughan, and A. Howard (2003). The Player/Stage project: Tools for multi-robot and distributed sensor systems. In *Proceedings of the 2003 International Conference on Advanced Robotics*, Coimbra, Portugal, pp. 317–323.

Gibbs, J. W. (1879). On the fundamental formulae of dynamics. *American Journal of Mathematics 2*, 49–64.

Gill, A. (1962). *Introduction to the theory of finite-state machines*. New York: McGraw-Hill.

Ginsburg, S. (1962). *Introduction to Mathematical Machine Theory*. Addison-Wesley series in computer science and information processing. Addison-Wesley.

Golub, G. and C. Van Loan (1989). *Matrix Computations*. The Johns Hopkins University Press.

Gourdeau, R. (1997, September). Object oriented programming for robotic manipulators simulation. *IEEE Robotics and Automation Magazine 4*(3), 21–29.

Hanafusa, H., T. Yoshikawa, and Y. Nakamura (1981). Analysis and control of articulated robot arms with redundancy. In *Proceedings of the 8th World Congress of the International Federation of Automatic Control*, Kyoto, Japan, pp. XIV:78–83.

Harel, D. (1987). State charts: A visual formalism for complex systems. *Science of Computer Programming 8*, 231–274.

Hartley, R. and A. Zisserman (2004). *Multiple View Geometry in Computer Vision* (Second ed.). Cambridge University Press, ISBN: 0521540518.

Hertz, H. (1894). *Die Prinzipien der Mechanik in neuem Zusammenhange dargestellt*. Leipzig, Germany: Barth. Reprinted in "The principles of mechanics," Dover, 1956.

Hertz, H. (1984). *Die Prinzipien der Mechanik. Einleiting*. Leipzig, Germany: Geest und Portig. With annotations by Josef Kuczera.

Hooker, W. W. and G. Margulies (1965). The dynamical attitude equations for an arbitrary n-body satellite having r rotational degrees of freedom. *J. Astronautical Sciences 12*(4), 123–128.

Joyeux, S., R. Philippsen, R. Alami, and S. Lacroix (2009). A plan manager for multi-robot systems. *The International Journal of Robotics Research*.

Kaiser, M., R. Dillmann, H. Friedrich, I. Lin, F. Wallner, and P. Weckesser (1996). Learning coordination skills in multi-agent systems. In *Proceedings of the 1996 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Osaka, Japan, pp. 1488–1495. IROS96.

Khatib, O. (1987). A unified approach for motion and force control of robot manipulators: The operational space formulation. *IEEE Journal of Robotics and Automation RA-3*(1), 43–53.

Khatib, O., O. Brock, K.-S. Chang, D. Ruspini, L. Sentis, and S. Viji (2004). Human-centered robotics and interactive haptic simulation. *The International Journal of Robotics Research 23*(2), 167–178.

Khatib, O., L. Sentis, and J.-H. Park (2008, March). A unified framework for whole-body humanoid robot control with multiple constraints and contacts. In *European Robotics Symposium*, Prague, Czech Republic, pp. 303–312.

Khronos Group (2006). COLLADA Document Object Model. `http://sourceforge.net/projects/collada-dom/`. Last visited March 2010.

Koenig, N. and A. Howard (2004). Design and use paradigms for gazebo, an open-source multi-robot simulator. In *Proceedings of the 2004 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Volume 3, Sendai, Japan, pp. 2149–2154 vol.3. IROS2004.

Laengle, T., T. C. Leuth, and U. Rembold (1995). A distributed control architecture for autonomous robot systems. In H. Bunke, T. Kanade, and N. H. (Eds.), *Modelling and planning for sensor based intelligent robot systems*, Volume 21 of *Series in Machine Perception and Artificial Intelligence*. World Scientific.

Lagrange, J. L. (1867). Mécanique analytique. In J.-A. Serret (Ed.), *Oeuvres*. Paris, France: Gauthier-Villars.

Latombe, J. C. (1991). *Robot motion planning*, Volume 124 of *Int. Series in Engineering and Computer Science*. Boston, MA: Kluwer Academic Publishers.

Latombe, J. C. (1999). Motion planning: A journey of robots, molecules, digital actors, and other artifacts. *The International Journal of Robotics Research 18*(11), 1119– 1128. Invited paper.

LaValle, S. M. (2006). *Planning algorithms*. Cambridge University Press.

Lilov, L. and J. Wittenburg (1977). Bewegungsgleichungen für Systeme starrer Körper mit Gelenken beliebiger Egenschaften. *Zeitschrift für Angewandte Mathematik und Mechanik 57*(3), 137–152.

Lötzsch, M., J. Bach, H.-D. Burkhard, and M. JÃijngel (2004). Designing agent behavior with the extensible agent behavior specification language XABSL. In D. Polani, B. Browning, and A. Bonarini (Eds.), *RoboCup 2003: Robot Soccer World Cup VII*, Volume 3020 of *Lecture Notes in Artificial Intelligence*, Padova, Italy, pp. 114–124. Springer.

Lozano-Pérez, T. (1976). The design of a mechanical assembly system. Technical Report Technical Report 397, MIT Artificial Intelligence Laboratory.

Lozano-Pérez, T. (1983). Robot programming. In *Proceedings of the IEEE*, Volume 71, pp. 821–841.

Lozano-Pérez, T. and R. A. Brooks (1985). An approach to automatic robot programming. Technical Report Technical Report 842, MIT Artificial Intelligence Laboratory.

Luh, J. Y. S., M. W. Walker, and R. P. C. Paul (1980). On-line computational scheme for mechanical manipulators. *Transactions of the ASME, Journal of Dynamic Systems, Measurement, and Control 102*(2), 69–76.

Maciejewski, A. A. and C. A. Klein (1985). Obstacle avoidance for kinematically redundant manipulators in dynamically varying environments. *The International Journal of Robotics Research 4*(3), 109–117.

Maciejewski, A. A. and C. A. Klein (1989). The singular value decomposition: Computation and applications to robotics. *The International Journal of Robotics Research 8*(6), 63–79.

MacIntyre, B. and S. Feiner (1998). A distributed 3D graphics library. In *SIGGRAPH '98: Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, pp. 361–370.

Maes, P. (1989). The dynamics of action selection. In *IJCAI'89: Proceedings of the 11th international joint conference on Artificial intelligence*, San Francisco, CA, USA, pp. 991–997. Morgan Kaufmann Publishers Inc.

Mason, M. T. (1981). Compliance and force control for computer controlled manipulators. *IEEE Transactions on Systems, Man, and Cybernetics SMC-11*(6), 418–432.

Mataric, M. J. (1997). Behavior-based control: Examples from navigation, learning, and group behavior. *Journal of Experimental and Theoretical Artificial Intelligence 9*, 323–336.

McCarragher, B. J. and H. Asada (1992). A discrete event controller using Petri nets applied to assembly. In *Proceedings of the 1992 IEEE/RSJ*

International Conference on Intelligent Robots and Systems, Raleigh, NC, pp. 2087–2094. IROS92.

McCarragher, B. J., G. Hovland, P. Sikka, P. Aigner, and D. Austin (1997). Hybrid dynamic modeling and control of constrained manipulation systems. *IEEE Robotics and Automation Magazine 4*(2), 27–44.

Mujtaba, S. and R. Goldman (1981). AL user's manual. Technical Report CS-TR-81-889.

Nakamura, Y. (1991). *Advanced robotics: redundancy and optimization*. Reading, MA: Addison-Wesley.

Nakamura, Y. and H. Hanafusa (1986). Inverse kinematic solutions with singularity robustness for robot manipulator control. *ASMEDSMC 108*, 163–171.

Nakamura, Y., H. Hanafusa, and T. Yoshikawa (1987). Task-priority based redundancy control of robot manipulators. *The International Journal of Robotics Research 6*(2), 3–15.

NetAllied Systems GmbH (2009). OpenCOLLADA. `http://opencollada.org`. Last visited March 2010.

Newton, I. (1871). *Philosophiae naturalis principia mathematica* (Reprinted ed.). Glasgow, Scotland: Maclehose. Eds. W. Thomson and H. Blackburn.

OMG (2009). Uml specification. `http://www.omg.org/spec/UML`.

OpenCV (2001). Open computer vision (opencv). `http://opencv.willowgarage.com/`. Last visited March 2010.

Paul, R. (1977, March). WAVE: A model-based language for manipulator control. *The Industrial Robot 4*, 10–17.

Paul, R. P. (1981). *Robot Manipulators: Mathematics, Programming, and Control*. Cambridge, MA: MIT Press.

Popov, E. P., A. F. Vereshchagin, and S. L. Zenkevich (1978). *Manipulyatsionnye roboty: dinamika i algoritmy*. Moscow: Nauka.

Popplestone, R., A. Ambler, and I. Bellos (1978). RAPT, A language for describing assemblies. *The Industrial Robot 5*(3), 131–137.

Prassler, E., H. Bruyninckx, K. Nilsson, and A. Shakhimardanov (2009). The use of reuse for designing and manufacturing robots. Technical report, Robot Standards project.

Radestock, M. and S. Eisenbach (1996). Coordination in evolving systems. In *Trends in Distributed Systems. CORBA and Beyond*, pp. 162–176. Springer-Verlag.

Radestock, M. and S. Eisenbach (2003). Coordinating components in middleware systems. *Concurrency and Computation: Practice and Experience 15*(13), 1205–1231.

Raibert, M. and J. J. Craig (1981). Hybrid position/force control of manipulators. *Transactions of the ASME, Journal of Dynamic Systems, Measurement, and Control 102*, 126–133.

Roberson, R. E. and R. Schwertassek (1988). *Dynamics of multibody systems*. Springer-Verlag.

Roberson, R. E. and J. Wittenburg (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*, Londen, U.K., pp. 46D.1–46.D9.

Roosendaal, T. (2002). Blender. `http://www.blender.org`. Last visited March 2010.

Ruoff, C. (1979). Teach - a concurrent robot control language. In *Computer Software and Applications Conference, 1979. Proceedings. COMPSAC 79. The IEEE Computer Society's Third International*, pp. 442–444.

Rutgeerts, J. (2007). *Task specification and estimation for sensor-based robot tasks in the presence of geometric uncertainty*. Ph. D. thesis, Department of Mechanical Engineering, Katholieke Universiteit Leuven, Belgium.

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*, 1174–1177.

Samson, C., M. Le Borgne, and B. Espiau (1991). *Robot Control, the Task Function Approach*. Oxford, England: Clarendon Press.

Saridis, G. N. and H. E. Stephanou (1977). A hierarchical approach to the control of a prosthetic arm. *IEEE Transactions on Systems, Man, and Cybernetics 7*(6), 407–410.

Schaal, S. (1999). Is imitation learning the route to humanoid robots ? *Trends in Cognitive Sciences 3*(6), 233–242.

Schraft, R. D. and C. Meyer (2006). Proceedings of the joint conference on robotics. In *Proceedings of the Joint Conference on Robotics*, Düsseldorf, Germany.

Shimano, B. (1979). Val: a versatile robot programming and control system. In *Computer Software and Applications Conference, 1979. Proceedings. COMPSAC 79. The IEEE Computer Society's Third International*, pp. 878–883.

Siciliano, B. and J.-J. E. Slotine (1991). A general framework for managing multiple tasks in highly redundant robotic systems. In *Proceedings of the 1991 International Conference on Advanced Robotics*, Pisa, Italy, pp. 1211–1216.

Silver, D. (1973). The little robot system. Technical Report AIM-273, MIT Artificial Intelligence Lab.

Smith, R. (2000). ODE: Open Dynamics Library. `http://www.ode.org`. Last visited March 2010.

Smits, R., T. De Laet, K. Claes, H. Bruyninckx, and J. De Schutter (2008). iTaSC: a tool for multi-sensor integration in robot manipulation. In *IEEE International Conference on Multisensor Fusion and Integration for Intelligent Systems*, Seoul, South-Korea, pp. 426–433. MFI2008.

Smits, R., D. Fioravanti, T. De Laet, B. Allotta, H. Bruyninckx, and J. De Schutter (2008). *Unifying Perspectives in Computational and Robot Vision*, Volume 8 of *Lecture notes in Electrical Engineering*, Chapter Image-Based Visual Servoing with Extra Task Related Constraints in a General Framework for Sensor-Based Robot Systems, pp. 187–204. Springer.

Soetens, P. (2006, May). *A Software Framework for Real-Time and Distributed Robot and Machine Control*. Ph. D. thesis, Department of Mechanical Engineering, Katholieke Universiteit Leuven, Belgium. `http://www.mech.kuleuven.be/dept/resources/docs/soetens.pdf`.

Soetens, P. and H. Bruyninckx (2005). Realtime hybrid task-based control for robots and machine tools. In *Proceedings of the 2005 IEEE International Conference on Robotics and Automation*, Barcelona, Spain, pp. 260–265. ICRA2005.

Strauss, P. S. and R. Carey (1992). An object-oriented 3D graphics toolkit. *ACM SIGGRAPH Computer Graphics 26*(2), 341–349.

Takase, K., R. P. Paul, and E. J. Berg (1981, April). A structured approach to robot programming and teaching. *Systems, Man and Cybernetics, IEEE Transactions on 11*(4), 274–289.

Taylor, R., P. Summers, and J. Meyer (1982). Aml: A manufacturing language. *The International Journal of Robotics Research 1*(3), 19–41.

Taylor, R. H. (1976). *Synthesis of manipulator control programs from task-level specifications*. Ph. D. thesis, Department of Computer Science, Stanford University, Stanford CA.

Valckenaers, P., H. Van Brussel, Hadeli, O. Bochmann, B. Saint Germain, and C. Zamfirescu (2003). On the design of emergent systems: an investigation of integration and interoperability issues. *Engineering Applications of Artificial Intelligence 16*, 377–393.

Van Brussel, H., J. Wyns, P. Valckenaers, L. Bongaerts, and P. Peeters (1998). Reference architecture for holonic manufacturing systems: PROSA. *Computers in Industry 37*(3), 255–274.

Vereshchagin, A. F. (1974). Computer simulation of the dynamics of complicated mechanisms of robot-manipulators. *Engineering Cybernetics 12*(6), 65–70.

Vereshchagin, A. F. (1975). Gauss principle of least constraint for modelling the dynamics of automatic manipulators using a digital computer. *Soviet Physics Doklady 20*(1), 33–34. Originally published in Dokl. Akad. Nauk SSSR, Vol. 220, No. 1, pp. 51–53, 1975.

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.

Vukobratović, M. (1973). How to control the artificial anthropomorphic systems. *IEEE Transactions on Systems, Man, and Cybernetics 3*, 497–507.

W3C Math Working Group. Mathematical markup language. `http://www.w3.org/Math/`.

Wampler, C. W. (1986). Manipulator inverse kinematic solutions based on vector formulations and damped least-squares solutions. *IEEE Transactions on Systems, Man, and Cybernetics 16*, 93–101.

Willow Garage (2009). Robot Operating System (ROS). `http://www.ros.org`. Accessed online 26 Januari 2010.

Wittenburg, J. (1977). *Dynamics of systems of rigid bodies*. Stuttgart, Germany: Teubner.

Yoshikawa, T. and A. Sudou (1993). Dynamic hybrid position/force control of robot manipulators—on-line estimation of unknown constraint. *IEEE Transactions on Robotics and Automation 9*(2), 220–226.

Curriculum vitae

Personal data

Ruben Smits
August 24 1982, Turnhout, Belgium
ruben.smits@mech.kuleuven.be
http://people.mech.kuleuven.be/~rsmits/

Education

- **2005 - 2010**: **PhD student** at the Katholieke Universiteit Leuven, Belgium.

- **May 2009**: **Visiting PhD student** at Willow Garage Inc., Menlo Park, California, USA.

- **2002 - 2005**: **Master of science in mechanical engineering** specialization mechatronics and machine design at the Katholieke Universiteit Leuven, Belgium.

- **2000 - 2002**: **Bachelor in engineering** at the Katholieke Universiteit Leuven, Belgium.

Publications

Articles in internationally reviewed scientific journals

De Schutter, J., De Laet, T., Rutgeerts, J., Decré, W., Smits, R., Aertbeliën, E., Claes, K., Bruyninckx, H. (2007). Constraint-based task specification and estimation for sensor-based robot systems in the presence of geometric uncertainty. *International Journal of Robotics Research, 26*(5), 433-455

Book chapters, internationally recognized scientific publisher

Smits, R., De Laet, T., Claes, K., Bruyninckx, H., De Schutter, J. (2009). iTaSC: a tool for multi-sensor integration in robot manipulation. In: Lee S., Ko H., Hahn H. (Eds.), *Multisensor Fusion and Integration for Intelligent Systems* (pp. 235-254). United Kingdom: Springer London Ltd.

De Laet, T., Smits, R., De Schutter, J., Bruyninckx, H. (2009). *Adaptive full scan model for range finders in dynamic environments.* In: *Experimental Robotics, Springer tracts in Advanced Robotics, 54*, (pp. 441–450). Springer Berlin / Heidelberg.

Smits, R., Fioravanti, D., De Laet, T., Allotta, B., Bruyninckx, H., De Schutter, J. (2008). Image-based visual servoing with extra task related constraints in a general framework for sensor-based robot systems. In: *Unifying perspectives in computational and robot vision* (pp. 187-204). United Kingdom: Springer London Ltd.

Papers at international conferences and symposia, published in full in proceedings

De Laet, T., Smits, R., De Schutter, J., Bruyninckx, H. (2009). *Adaptive full scan model for range finders in dynamic environments.* ISER2008, 2008 International Symposium on Experimental Robotics. Athens, Greece, Jul 14-17, 2008.

Smits, R., Bruyninckx, H., De Schutter, J. (2009). *Software support for high-level specification, execution and estimation of event-driven, constraint-based multi-sensor robot tasks.* In : Proceedings of the 14th International Conference on Advanced Robotics. International Conference on Advanced Robotics. Munich, June 22-26, 2009.

Decré, W., Smits, R., Bruyninckx, H., De Schutter, J. (2009). *Extending iTaSC to support inequality constraints and non-instantaneous task specification.* In : Proceedings of the 2009 IEEE International Conference on Robotics and Automation. the 2009 IEEE International Conference on Robotics and Automation. Kobe, Japan, 12-17 May 2009.

Smits, R., De Laet, T., Claes, K., Bruyninckx, H., De Schutter, J. (2008). *iTASC: a tool for multi-sensor integration in robot manipulation.* In : Proceedings of the Multisensor fusion and integration for intelligent systems. Multisensor fusion and integration for intelligent systems. Seoul, Korea, Aug 20-22, 2008 (pp. 445-452).

Smits, R., Bruyninckx, H., Meeussen, W., Baeten, J., Slaets, P., De Schutter, J. (2006). *Model based position-force-vision sensor fusion for robot compliant motion control.* In : Proceedings of the International Conference on Multisensor Fusion and Integration for Intelligent Systems. International Conference on Multisensor Fusion and Integration for Intelligent Systems. Heidelberg, Germany, Sep 3-6, 2006 (pp. 501-506).

Meeting abstracts, presented at international conferences and symposia, published or not published in proceedings or journals

Decré, W., Smits, R., Bruyninckx, H., De Schutter, J. (2009). *Extending iTaSC to support inequality constraints.* In : 28th Benelux meeting on systems and control. Benelux meeting on systems and control, Spa, Belgium, 16-18 March 2009 (147-147).

Smits, R., Bruyninckx, H. (2007). *Sensor-based control using the open robot realtime toolkit.* In : Book of abstracts of 26th Benelux Meeting on systems and Control. 26th Benelux Meeting on systems and Control, Lommel, Belgium, Mar 13-15, 2007 (21).

Arenberg Doctoraatsschool Wetenschap & Technologie
Faculteit Ingenieurswetenschappen
Departement Werktuigkunde
Afdeling Productietechnieken, Machinebouw en Automatisering (PMA)
Celestijnenlaan 300B B-3001 Leuven

KATHOLIEKE UNIVERSITEIT LEUVEN

ASSOCIATIE K.U.LEUVEN