KU LEUVEN

ARENBERG DOCTORAL SCHOOL
Faculty of Engineering Science

# Composing Concurrent Reactive Robot Skills

## Integration of Task Specification and Control with Symbolic Planning and CAD Information

**Yudha Prawira Pane**

Supervisors:
Prof. dr. ir. Joris De Schutter
Dr. ir. Wilm Decré
Dr. ir. Erwin Aertbeliën

Dissertation presented in partial fulfillment of the requirements for the degree of Doctor of Engineering Science (PhD): Mechanical Engineering

March 2022

# Composing Concurrent Reactive Robot Skills

Integration of Task Specification and Control with Symbolic Planning and CAD Information

**Yudha Prawira PANE**

Examination committee:
Prof. dr. ir. Patrick Wollants, chair
Prof. dr. ir. Joris De Schutter, supervisor
Dr. ir. Wilm Decré, supervisor
Dr. ir. Erwin Aertbeliën, supervisor
Prof. dr. ir. Herman Bruyninckx
Dr. ir. Vahid Mokhtari
Prof. dr.-ing. Ulrike Thomas
(TU Chemnitz)
Prof. dr. ir. Lars Tingelstad
(NTNU)

Dissertation presented in partial fulfillment of the requirements for the degree of Doctor of Engineering Science (PhD): Mechanical Engineering

March 2022

© 2022 KU Leuven – Faculty of Engineering Science
Uitgegeven in eigen beheer, Yudha Prawira Pane, Celestijnenlaan 300, 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, electronic or any other means without written permission from the publisher.

# Preface

My PhD is a long journey filled with adventures and excitements, but also obstacles and detours. I am fortunate enough to have people that helped me reach my destination and shaped me into a better person. First, I would like to thank Joris, my promotor, for his continuous support and belief in me during the difficult times. Joris is both an advisor and a role model that I've always looked up to. Among his generosities, I particularly appreciate the freedom of research that he entrusted to me. Coming into the end of second year, I found it really hard to find novel research that can be published. Joris allowed me to explore ideas beyond my initial thesis topic and collaborate with people from different expertises. From skill programming using CAD models to combining skills with symbolic task planning, even to a topic which is completely unrelated to my PhD, namely haptic gloves based on twisted string actuators. The freedom that Joris gave enabled me to finally make my own contributions to robotics research.

This work would not have been possible without my co-promotors, Wilm and Erwin. I am thankful to Wilm for always being available to assist me in my research through its ups and downs. Furthermore, thanks to his initiatives and managerial prowess, this PhD is filled with more than just conducting research and writing papers. I got to experience how it feels like to get my hands on exciting "toys" (aka expensive robots), participating in robotics competitions, and working on industry-relevant projects. Apart from that, I enjoyed our evening discussions of random topics, from the hypothetical "Lousy Robotics Inc." to financial management :). An equal amount of thanks to Erwin who were always willing to help turning my vague ideas into concrete research values. I admire Erwin as a full stack roboticist who can seemingly think of a solution for any robotics problem, be it problems related to perception, control, or software. Erwin's craftsmanship is something that I aspire to achieve during my lifetime. Joris, Wilm and Erwin, my deepest gratitude to all of you.

I would also like to thank my doctoral supervisory and examination committee:

Prof. Herman Bruyninckx, Prof. Ulrike Thomas, Prof. Lars Tingelstad, Dr. Vahid Mokhtari, Prof. Dirk Cattrysse and Prof. Patrick Wollants. My gratitude to Ulrike, Lars, Vahid and Herman for the invaluable feedback to the thesis. I especially thank Herman for being an inspiration with his ability to bring different perspectives on some of the most challenging robotics problems. Herman also inspires me to be critical in evaluating the hypes and trends in the robotics community.

I am proud to be a part of KU Leuven Robotics Group family, where scholars from different backgrounds, nationalities and cultures come to seek knowledge. It was a privilege to have shared an office with wonderful colleagues. My thanks to Enea for helping me shape my research direction. Cristian for being a comrade that share the same battlegrounds, from the robotics competitions to spending time together in the lab during paper deadlines. Vahid for his generous helps and the fruitful collaboration in integrating reactive skills with deliberative planning. Mohssen for showing me the world of "haptic gloves" and "twisted string actuators". I am also extending my thanks to other colleagues: Santiago, Gianni, Maxim, Kevin, Federico, Nico, Niko, Filip, Maria, Juan, Lander, Jonas, Gil, Lin, Marco, Bart, Umut, Ajay, Ali, Johan Philips, Fergal, Sergio, Markus, Andrea, Ruan, Johan, Murtaza. A special credit to Thomas for translating the abstract. Thank you all for making my stay in the group a beautiful chapter in my life, something that I will cherish forever.

During my PhD, I crossed paths with people from outside KU Leuven Robotics Group. I would like to thank Mathias, my Norwegian friend who exposed me to the idea that robot skills can programmed using CAD models. Thanks to Asad, Ali and Patricia from Flanders Make for the memorable collaborations and allowing me to visit and work at their lab. Thanks to my new colleagues at Magics Technology, in particular Gert for his help to get me familiar with the new working environment.

PhD can be a lonely job sometimes, therefore, I am lucky that I met fellow Indonesians in Leuven who made me feel closer to home. I would like to thank Mbak Indah (with Zahra and Zaky) who was like my own sister, Mas Bayu (with A'yun), Kang Bobby, Mas Yusro, Mbak Kezia, Mbak Gati, Bang Arief, Kang Islah, Mas Yodi who are some of the first people I met in Leuven. Shout out to Andi who kept me sane by being the best badminton partner, as well as Romo Bastian, Romo Willy, Kang Anton, Mas Azam, Mbak Novita, Tata. Other friends at PPI Leuven: Nicholas, Prima, Teh Isye, Kak Vivi, Mbak Lila, Mbak Isti, Haqi, Udin, Bindi, Kevin, Eagan. I lived in Leuven for so long that I've witnessed my friends came and left the town. So, my apologies for not mentioning all their names here. In addition, thanks to my best friends in Indonesia, Arijal, Amal and Fadian, who helped me passed the corona time with laughters.

I would like to dedicate this thesis to my parents, Mama and Papa, who are my biggest supporters. I can never repay everything that they have given to me. I only hope that I can continue to make them proud and happy. Many thanks and love to my little brother, Fadel, and my little sister, Suci who always motivate me to be a better "abang". And thank you Tia, for coming into my life and making it more colorful.

Last but not least, *Alhamdulillahirabbil Alamin ...* Thank you Allah for this wonderful blessing.

*Geel, March 2022,*
*Yudha Prawira Pane*

# Acknowledgments

The author gratefully acknowledges the financial support by Flanders Make projects ICON-FINROP (Fast and intuitive robot programming), SBO-MULTIROB (Rigorous approach for programming and optimal control of multi-robot systems) and SBO-PROUD (A hybrid model-based/learning approach for symbolic-level robot programming by user demonstration).

# Abstract

The general trend in industrial robotics sees an expansion of applications from traditional, repetitive tasks to challenging tasks requiring advanced sensing and higher intelligence. However, despite the technological advancements in robot hardware, sensors and tools, robot adoption in industry is still overshadowed by the fear of high deployment cost. This cost is partly due to the expensive programming, especially for complex tasks that involve a higher variation in the environment and objects to manipulate, and presence of disturbances. Consequently, the traditional programming methods are often no longer a viable solution.

This thesis aims to address this issue by providing a set of methods to tackle specific problems in programming robotic behaviors, i.e., "skills", that can perform complex tasks robustly. Common in all of these methods is the formulation of skills using constraint-based task specification. This approach specifies robotic tasks in terms of constraints as opposed to traditional approach where robots are programmed in terms of desired positions or paths. It provides two main advantages that this thesis tries to exploit: *i)* real-time reactivity by incorporating sensor input *ii)* ability to compose different behaviors that correspond to different aspects, such as the actual tasks, safety, and environment.

As a first contribution, this thesis presents a framework to specify composition of reactive and synchronized robot skills. First, this framework provides a skill model that captures deviating behavior during non-nominal conditions. The model also introduces the notion of skill progress as a measure of skill advancement towards its final state. Second, it presents composition patterns for combining skills concurrently such that certain composed behaviors are formed. The composed behaviors include skills with synchronized progresses as well as skills capable of reacting to disturbances.

As a second contribution, this thesis presents a framework that integrates skills

with symbolic task planning. While the previous contribution captures the reactivity at the continuous level, this contribution extends the reactivity at the discrete level. The integrated task planner does not only compute a nominal plan to achieve the given goal, but also recomputes a new plan when failures due to unpredictable disturbances occur. This new plan is generated and translated into reactive, composed skills during runtime. The result is an improved autonomy and robustness for robot applications in non-ideal environments.

As a third contribution, this thesis presents a framework to specify assembly tasks in CAD domain and to semi-automatically generate the corresponding skills. This framework leverages the information embedded in CAD data in order to infer suitable skill candidates and their appropriate parameters, given a manually specified task. The CAD domain specification offers a higher-level of abstraction, therefore reducing the needs for programmers to understand lower-level skill implementation.

Finally, this thesis provides experimental evaluations of the above contributions in industry relevant lab demonstrators. These demonstrators comprise assembly tasks of different workpieces where various level of uncertainties and disturbances are present. The demonstrators also show the applications of sensor-based skills in dual-arm scenarios, where tasks have to be performed concurrently.

# Beknopte samenvatting

Robotica in de industrie ondergaat een trend die gekenmerkt wordt door een uitbreiding van het toepassingsgebied van repetitieve taken naar uitdagende taken waarin geavanceerde perceptie en hogere intelligentie vereist zijn. Echter, ondanks de technologische vooruitgang in hardware, sensoren en werktuigen, worden robots slechts zelden aangewend in industriële toepassingen uit angst voor hoge uitrolkosten. Een belangrijke oorzaak van deze hoge kosten is de nood aan het programmeren van de (vaak) complexe taken waarbij rekening wordt gehouden met variaties in de omgeving van de robot en de objecten waarmee de robot interageert. Ook de aanwezigheid van verstoringen verhoogt de complexiteit van een robot taak. Als gevolg van dergelijke hoge programmeerkosten is de traditionele wijze van programmeren niet langer de aangewezen oplossing.

Deze thesis behandelt het bovenstaande probleem door een aantal methodes voor te stellen waarmee specifieke problemen, die zich stellen tijdens het programmeren van het gedrag van de robot, aangepakt worden. Deze gedragingen worden ook wel "vaardigheden" (Engels: skills) genoemd en stellen een robot in staat om complexe taken op een robuuste manier uit te voeren. De voorgestelde methodes formuleren de vaardigheden telkens door middel van een beperkingsgebaseerde beschrijving van de taak. Traditionele programmeermethodes beschrijven een taak voor een robot als functie van het te volgen traject of als functie van de gewenste eindpositie. De beperkingsgebaseerde programmeermethode beschrijft de taak van een robot als functie van beperkingen opgelegd aan de robot. Deze aanpak heeft twee belangrijke voordelen: *i)* Er is de mogelijkheid tot reactief gedrag in real-time door gebruik te maken van de input van sensoren. *ii)* Er is de mogelijkheid om gedragingen samen te stellen op basis van verschillende aspecten van een taak zoals het doel van de taak, veiligheid en de omgeving.

Een eerste bijdrage van deze thesis bestaat uit een raamwerk dat toelaat reactieve en gesynchroniseerde robotvaardigheden samen te stellen. Dit raamwerk heeft

twee voordelen. Ten eerste is er het model voor de vaardigheid dat het gedrag van de robot beschrijft, ook tijdens niet-nominale (afwijkende) omstandigheden. Het model introduceert ook het concept van voortgang tijdens een vaardigheid dat aangeeft hoe ver een vaardigheid gevorderd is ten opzichte van de eindsituatie. Ten tweede stelt het raamwerk methodes voor om vaardigheden gelijktijdig te combineren om samengestelde gedragingen te bekomen. Dit resulteert in samengestelde vaardigheden met gesynchroniseerde voortgang en vaardigheden die gepast reageren op verstoringen.

Een tweede bijdrage omvat een raamwerk waarin vaardigheden geïntegreerd worden met symbolische taakplanning. De bijdrage in de vorige paragraaf focust op reactief gedrag op het continue niveau. Deze tweede bijdrage breidt dit uit naar reactief gedrag op het discrete niveau. De geïntegreerde taakplanner stelt niet enkel een nominaal plan op om een bepaald doel te bereiken, maar herberekent dit plan ook wanneer het nominaal plan faalt omwille van onverwachte verstoringen. Dit nieuwe plan wordt vervolgens onmiddellijk vertaald naar reactieve en samengestelde vaardigheden. Het resultaat zijn meer autonome en robuuste robottoepassingen in niet-ideale omgevingen.

Een derde bijdrage van deze thesis bestaat uit een raamwerk voor het beschrijven van assemblagetaken in het CAD-domein. Het raamwerk genereert vervolgens semi-automatisch de bijhorende robotvaardigheden. Het raamwerk buit de informatie bevat in CAD-data uit zodat geschikte kandidaat-vaardigheden afgeleid kunnen worden. Daarnaast genereert het raamwerk ook de juiste parameters voor de vaardigheden op basis van een handmatig beschreven taak. De beschrijving in het CAD-domein biedt een nieuwe laag van abstractie en reduceert de nood aan programmeurs die de implementatie van de onderliggende vaardigheden moeten doorgronden.

Tot slot omvat deze thesis experimentele evaluaties van de bovenstaande bijdragen in een laboratoriumomgeving die relevant is voor de industrie. De experimenten bestaan uit assemblagetaken voor verschillende werktuigen met variërende niveaus van onzekerheid en waarin verstoringen mogelijk zijn. De experimenten tonen ook de mogelijkheden van sensorgebaseerde vaardigheden in tweearmige toepassingen, waarin twee robotarmen samenwerken en taken gelijktijdig uitvoeren.

# List of Abbreviations

**ABB** ASEA Brown Boveri. xx, 3, 7, 8

**AUTOPASS** AUTOmated Parts ASsembly System. 9

**BREP** Boundary Representation. 10

**CAD** Computer-Aided Design. xxiv, xxv, 8–10, 12, 14, 15, 78–86, 89–91, 94, 95, 97, 100–102, 104, 106–108, 110, 119–121

**CAM** Computer-Aided Manufacturing. 110

**COVID-19** Coronavirus Disease 2019. 1

**DOF** Degrees of Freedom. 18–20, 28

**DSL** Domain Specific Language. 19, 36, 56

**eTaSL** expressiongraph-based Task Specification Language. xxv, 15, 17, 19–22, 24–30, 35, 36, 38, 44, 51, 53, 56, 112, 115, 119

**eTC** expressiongraph-based Task Controller. 15, 61

**FANUC** Fuji Automatic NUmerical Control. 3

**FRI** Fast Research Interface. 100

**FSM** Finite State Machine. 42, 50, 53

**GUI** Graphical User Interface. 5, 8

**HQP** Hierarchical Quadratic Programming. xx, xxi, xxv, 25–30, 32, 112

**iTaSC** instantaneous Task Specification using Constraints. 19, 20, 35, 36, 53, 54, 112

**KRL** KUKA Robot Language. 3

**KUKA** Keller und Knappich Augsburg. x, 2, 3, 47

**LBR IIWA** Leichtbauroboter Intelligent Industrial Work Assistant. xxi, 2, 47, 51, 54

**LP** Linear Programming. 25

**LWR4** Light Weight Robot 4. 47, 51, 54

**MIP** Mixed Integer Programming. 113

**MPC** Model Predictive Control. 21, 115

**OPTIC** Optimising Preferences and Time-Dependent Costs. 69, 72

**OSQP** Operator Splitting Quadratic Program. 27

**OWL** Web Ontology Language. xxiii, 78, 94, 95, 98

**PDDL** Planning Domain Definition Language. 62, 68, 76, 110, 114, 119

**PTP** Point-to-Point. 2, 4, 18

**QP** Quadratic Programming. 19, 21–26, 64–67, 74

**rFSM** restricted Finite State Machine. 19, 97, 99, 107

**ROS** Robot Operating System. 100, 108

**RPS** Robot Programming Software. 7

**RSI** Robot Sensor Interface. 100

**SME** Small to Medium Enterprises. 1

**SoT** Stack of Task. 19–21, 27, 35, 36, 112

**STN** Simple Temporal Network. 113

**STRIPS** Stanford Research Institute Problem Solver. 62, 114

**TAMP** Task and Motion Planning. 58, 62

**TFF** Task Frame Formalism. 18, 53

**UML** Unified Modeling Language. 36

**URDF** Unified Robot Description Format. 20

**VKC** Virtual Kinematic Chain. 19

**WBC** Whole Body Control. 19–21

# List of Symbols

### General symbols

| | |
|---|---|
| $\lambda$ | Path length in a trajectory tracking task |
| $i, j, l$ | Symbols denoting counter/iteration |
| $k$ | Feedback control gain |
| $m, n$ | Symbols denoting natural numbers. A subscript is added for a more specific notation. |
| $n_f$ | Number of feature variables in $x$ |
| $n_r$ | Number of joint variables in $x$ |
| $n_s$ | Number of skill in an active skill set |
| $n_t$ | Number of task in a HQP optimization formulation |
| $t$ | Time variable |

### Optimization symbols

| | |
|---|---|
| $\bar{w}_{\mathrm{p}}$ | Weight factor that is applied to skill's progress constraints |
| $\bar{w}_{\mathrm{t}}$ | Weight factor that is applied to skill's task constraints |
| $\hat{\bar{w}}_{\mathrm{p}}$ | Modified weight factor that is applied to skill's progress constraints as a result of the weight scaling pattern |
| $\hat{\bar{w}}_{\mathrm{t}}$ | Modified weight factor that is applied to skill's task constraints as a result of the weight scaling pattern |
| $\mathbf{A}$ | Linear constraint matrix |
| $\mathbf{H}$ | Hessian matrix of a QP optimization |

| | |
|---|---|
| $\mathbf{L}$ | Lower bound vector of the box constraints |
| $\mathbf{L}_A$ | Lower bound vector of the linear constraints |
| $\mathbf{U}$ | Upper bound vector of the box constraints |
| $\mathbf{U}_A$ | Upper bound vector of the linear constraints |
| $\mathbf{W}_\varepsilon$ | Weight matrix corresponding to the slack variables |
| $\mathbf{W}_f$ | Weight matrix corresponding to the feature variable velocities |
| $\mathbf{W}_q$ | Weight matrix corresponding to the joint variable velocities |
| $\mu$ | Regularization factor of the QP Hessian |
| $\varepsilon$ | Slack variables |
| $\varepsilon^*$ | Optimal value of the slack variables |
| $p_\mathrm{p}$ | Priority level of a progress constraint |
| $p_\mathrm{t}$ | Priority level of a task constraint |
| $w$ | General symbol for weight |
| $w_{\mathrm{p_i}}$ | Weight value of the $i$-th progress constraint |
| $w_\mathrm{p}$ | Weight of a progress constraint |
| $w_{\mathrm{t_i}}$ | Weight value of the $i$-th task constraint |
| $w_\mathrm{t}$ | Weight of a task constraint |
| $x$ | Optimization variable in eTaSL |
| $x^*$ | Optimal value of the optimization variable |

**Robot kinematics and dynamics symbols**

| | |
|---|---|
| $\chi_f$ | Feature variables |
| $\chi_{f_{pv}}$ | Feature variable corresponding to the path coordinate |
| $\mathbf{J}$ | Jacobian matrix corresponding to the task constraints |
| $\tau_i$ | Torque expression about the $i$-axis where $i = x, y, z$ |
| $e$ | Task error function |
| $f_i$ | Force expression along the $i$-axis where $i = \{x, y, z\}$ |

| | |
|---|---|
| $K_a$ | Admittance matrix |
| $q$ | Robot joint variables |
| $T$ | General symbol for pose |
| $T_f$ | Final pose in a desired path |
| $T_i$ | Initial pose in a desired path |
| $T_{\mathrm{a}}^{\mathrm{b}}$ | Pose of frame a with respect to frame b |
| $W_{\mathrm{b}}^{\mathrm{a}}$ | Wrench applied to frame a and expressed in frame b |

**Skill modeling and composition symbols**

| | |
|---|---|
| $\alpha$ | Transient weight profile for runtime skill composition - increasing |
| $\beta$ | Transient weight profile for runtime skill composition - decreasing |
| $\eta$ | Sigmoid steepness parameter for the weight scaling pattern |
| $\gamma$ | Transient weight profile for runtime skill composition - restoring |
| $\kappa$ | Generic symbol for a controller |
| $\mathbb{S}$ | Skill set |
| $\mathbb{S}_a$ | Active skill set |
| $\mathbf{S}$ | Diagonal selection matrix in the weight scaling pattern |
| $\mathcal{T}$ | Generic symbol for a robotic task |
| CP | Composition pattern operation |
| $\mathsf{s}$ | Generic symbol for a skill |
| $\zeta$ | Activation signal of the weight scaling pattern |
| $c$ | Generic symbol for a constraint |
| $s$ | Progress variable of a skill |
| $t_0$ | Start time of the transient phase in a runtime skill composition |
| $t_f$ | End time of the transient phase in a runtime skill composition |
| $t_i$ | Intermediate time of the transient phase in a runtime skill composition |
| $w_{\mathrm{scaling}}$ | Sigmoid (hyperbolic tangent) function for the weight scaling pattern |

**Other symbols**

| | |
|---|---|
| $\mathbb{C}$ | Constraint set |
| $\mathbb{K}$ | Controller set |
| $\mathbb{R}$ | Real number set |
| $\rho$ | Overall preference to resolve conflict in multiple skills execution |
| $h^a$ | Heuristic binary function that is associated with action advancement for computing preference in conflicting skills |
| $h^m$ | Heuristic binary function that is associated with object manipulation for computing preference in conflicting skills |
| $h^s$ | Heuristic binary function that is associated to skill progress for computing preference in conflicting skills |

# Contents

**Abstract** v

**Beknopte samenvatting** vii

**List of Abbreviations** xi

**List of Symbols** xv

**Contents** xvi

**List of Figures** xx

**List of Tables** xxv

**1 Introduction** 1
- 1.1 Background . . . 1
- 1.2 Related Works . . . 4
  - 1.2.1 Skill-based robot programming . . . 4
  - 1.2.2 CAD-based robot programming . . . 8
- 1.3 Research Goal and Objectives . . . 10
- 1.4 Overall Approach of the Thesis . . . 11
- 1.5 Contributions . . . 13
  - 1.5.1 Contribution 1: A framework to program composition of reactive and synchronized robot skills . . . 13
  - 1.5.2 Contribution 2: A framework for integrating task planning and reactive skills to enable autonomous runtime skill composition . . . 13
  - 1.5.3 Contribution 3: CAD-based task specification and skill programming for assembly applications . . . 14
- 1.6 Thesis Outline . . . 15

**2 Constraint-based Task Specification for Sensor-based Robot Skills 17**
2.1 Overview of related works . . . 17
2.2 eTaSL Framework . . . 19
2.2.1 Overview of eTaSL . . . 19
2.2.2 Elements that constitute an eTaSL specification . . . 20
2.2.3 Optimization problem formulation and solver . . . 21
2.3 Extending eTaSL solver to support arbitrary number of priority levels . . . 24
2.3.1 Related work . . . 25
2.3.2 Formulation of eTaSL solver with HQP . . . 25
2.3.3 An example of the application of the HQP solver in resolving conflicting, concurrent tasks . . . 27

**3 Skill-based Programming Framework for Composable Reactive Robot Behaviors 33**
3.1 Introduction . . . 34
3.2 Related Work and Motivation . . . 35
3.3 Constraint-Based Skill Modelling . . . 37
3.3.1 Constraint and Monitor . . . 38
3.3.2 Specifying Skill Behavior with Constraints . . . 39
3.3.3 Modelling Skill Progress . . . 40
3.3.4 Classifying a Skill Based on its Progress . . . 41
3.4 Composing Behaviors with Composition Patterns . . . 42
3.4.1 Progress Synchronization Patterns . . . 42
3.4.2 Weight Scaling Pattern . . . 45
3.4.3 Examples of skill composition . . . 46
3.5 Experimental Validation . . . 47
3.5.1 Problem Scenario . . . 47
3.5.2 Implemented skills . . . 47
3.5.3 Composing the Skills . . . 50
3.5.4 Analyzing the Composition Behaviors . . . 51
3.6 Discussion and Conclusion . . . 54

**4 Autonomous Runtime Composition of Sensor-based Skills Using Concurrent Task Planning 57**
4.1 Introduction . . . 58
4.2 Related Work . . . 61
4.3 Constraint-based Composable Skills . . . 63
4.3.1 Skill Modeling and Specification . . . 63
4.3.2 Runtime Composition of Skills . . . 65
4.4 Task Planning . . . 68
4.5 Experimental Validation . . . 71
4.5.1 Experimental Setup and Task Scenario . . . 71

4.5.2 Experimental Results . . . . . 72
4.5.3 Conflict Resolution in Multiple Skills Execution . . . . . 74
4.5.4 Smooth Transient in Runtime Skill Composition . . . . 75
4.6 Discussion and Conclusion . . . . . 76

**5 A System Architecture for CAD-based Robotic Assembly with Sensor-based Skills 78**
5.1 Introduction . . . . . 79
5.2 Related Works . . . . . 81
5.3 Overview . . . . . 83
5.3.1 System Architecture . . . . . 83
5.3.2 Skill and Task . . . . . 85
5.4 Grounding From Task to Skill . . . . . 86
5.4.1 CAD-Based Ontology: From Task to Skill . . . . . 86
5.4.2 Planning and Inference Module . . . . . 88
5.5 Implementation . . . . . 94
5.5.1 Application Layer . . . . . 94
5.5.2 Process Layer . . . . . 95
5.5.3 Control Layer . . . . . 100
5.6 Test Cases . . . . . 100
5.6.1 Case 1: Assembly of Air Compressor . . . . . 101
5.6.2 Case 2: Assembly of Electrical Motor . . . . . 102
5.7 Discussion . . . . . 104
5.8 Conclusion . . . . . 106

**6 Discussion 109**
6.1 Connections between the contributions . . . . . 109
6.2 Conflict resolution of robotic tasks . . . . . 110
6.3 Coordinated execution of concurrent robot tasks . . . . . 113
6.4 Limitations of the proposed methods . . . . . 115
6.4.1 Limitation of reactive planning . . . . . 115
6.4.2 Limitation of the runtime replanning approach . . . . . 115
6.4.3 Limitation of CAD-level assembly specifications . . . . . 116

**7 Conclusion 118**
7.1 Main contributions . . . . . 119
7.1.1 A framework to program composition of reactive and synchronized robot skillls - Chapter 3 . . . . . 119
7.1.2 A framework for integrating task planning and reactive skills to enable autonomous runtime skill composition - Chapter 4 . . . . . 119
7.1.3 CAD-based task specification and skill programming for assembly applications - Chapter 5 . . . . . 120

7.2 Suggestions for future work . . . . . 120
7.2.1 Integration with motion planning approach . . . . . 120
7.2.2 An extended CAD-level assembly task specification . . . 121
7.2.3 Exploration of tactile sensing for improved manipulation skills . . . . . 121

**Bibliography** **123**

**List of publications** **137**

# List of Figures

1.1 Examples of a simple task (left) and a complex task (right). The simple task is to perform a linear motion that moves the robot's end-effector to a desired target pose in free space. Conventional robot programming methods are suitable for such tasks due to the rudimentary behavior. In contrast, the complex task is exemplified by an unknown-contour following task in which the robot has to maintain a desired contact force and react to human-induced disturbance. Such a task would be too difficult to program using conventional methods due to the composition of different behaviors and involvement of sensor inputs. . . . . 3

1.2 Examples of skill-based programming solutions offered by industry. Left: robot manufacturers' solutions represented by ABB and Franka Emika. Right: non-robot manufacturers' solutions represented by Artiminds Robotics, drag and bot and Ready Robotics. . . . . 8

2.1 A dual-arm setup for part grasping scenario to illustrate the applicability of the implemented HQP solver. Each arm is trying to reach a part in order to pick and assemble later. The parts are placed in a rack in such a way that they cannot be picked by both arms simultaneously. The desired paths to approach the parts are depicted with dashed lines. . . . . 28

2.2 Result of the tasks execution with different solver settings. First column: standard solver with a weight setting that favors both robots to stay on the paths, second column: standard solver with a weight setting that favors the prioritized arm to reach the end goal, third column: HQP solver. The first row shows the progress variables that parameterize the desired paths. The second and third row show the deviation plots for the first and second arm, respectively. These two rows represent the errors of staying on the desired paths, i.e., $\mathcal{T}_1$ and $\mathcal{T}_2$. The fourth row shows the Euclidean distances to the target position, with inset plots that magnify the converged distance for the prioritized arm. This row represents the errors of trying to achieve the end goals, i.e., $\mathcal{T}_3$ and $\mathcal{T}_4$. Finally, the final row shows the desired and traveled paths of the two arms. The right trajectories correspond to the first arm and the right trajectories correspond to the second arm. 32

3.1 Cooperative object transfer with dual arm setup . . . . . 37
3.2 A Cartesian motion skill that moves a robot end-effector to a target pose while following a given path (dashed line). The blue solid line signifies the path already travelled. . . . . 39
3.3 The structure of constraints in a skill specification . . . . . 40
3.4 The rotary compressor parts to be assembled. Note that the detail of the rotors' helical meshing is not visible. . . . . 49
3.5 Snapshots of the assembly process. Each robot performs a sequence of gripping and inserting the rotor. Human disturbance is acting on the LBR IIWA robot (iii). . . . . 51
3.6 The overall sequential and concurrent compositions for the rotors assembly task. Description about each block is provided in the bottom right part of the figure. The arrows denote sequential skill transitions. . . . . 52
3.7 The scaled skill weights during pre-picking task of the small rotor. The disturbance force and the skill progress are also shown. . . 55
3.8 The evolution of the skill progresses under leader-follower synchronization pattern. . . . . 55

4.1 A dual-arm setup for the experimental validation of our online replanning and acting framework. The experiment involves an industrial pump solenoid assembly task in a dynamic environment where unforeseen disturbances, e.g., an obstacle (detected with a Vive tracker) and a force-based disturbance by a human operator (detected with force sensors), might enter the workspace. In this framework, reactive robot skills are composable and computed at runtime by a task planner. When a disturbance is detected at execution time, the task planner computes online a composed behavior in order to react to the disturbance. . . . . 59
4.2 A three-layer architecture for online replanning and acting. . . 60
4.3 A Cartesian motion skill that moves a robot end-effector to a target pose by following a given path (dashed line). The blue solid line is the path already traveled. Task constraints restrict the task frame, i.e., the robot end-effector to stay within the desired path. Progress constraints drive the task frame to move from the initial pose $T_i$ to the end pose $T_f$ along the path. . . 64
4.4 Plans containing skills sequences (top) and Gantt chart representing skills execution (bottom) to illustrate concurrent skill compositions at runtime. . . . . 68
4.5 The weight scaling factor profiles to achieve smooth transient behavior at runtime composition of skills. The transient is divided into two phases: $(i)$ at $t_0 \leq t \leq t_i$, the weight factor $\alpha$ is applied to the newly introduced skill, and simultaneously the old active skill is scaled with weight factor $\beta$. At $t = t_i$, the priority is changed for the new skill; $(ii)$ at $t_i \leq t \leq t_f$, the old active skill weight is restored by applying a scaling factor $\gamma$. . . . . 69
4.6 An example of a problem with a durative action useful for enforcing concurrency. Temporally annotated conditions in the durative action avoid_collision, specify when the action can start, continue or end, and when it can produce effects immediately or at the end of the action. The avoid_collision action produces the collision_free condition necessary to start the approach_to_grip action, however, it only ends when the arm_at condition holds by the approach_to_grip action. The model ensures that two actions run concurrently. . . . . 70

4.7 The initial concurrent plan for the task of two solenoids assembly. The plan includes subplans represented as rectangles: total order (white), partial order (light gray). Actions are shown in orange color. subplan_0 involves subplan_0_0 and subplan_0_1 for the left and the right arms respectively, that can be executed in a partial order manner. Each of subplan_0_0 and subplan_0_1 consists of a sequence of totally-ordered actions. . . . . 72
4.8 Part of a new plan computed after detecting an obstacle to right_arm. The plan contains a concurrently composed skill, move_above and avoid_collision, shown in dark gray. . . . . 73
4.9 Snapshots of the two solenoids assembly task execution. The arms start from standby poses (i). In (ii), they execute the first actions when an obstacle suddenly enters and blocks the robots' path. Here, a new plan is computed which composes reactive skills to avoid the obstacle. In (iii), another disturbance occur in which a human operator pull the right arm from its nominal path. Again here, the task planner recomputes a new plan and composes a reactive admittance skill. In (iv), the left robot has already inserted one solenoid while the right arm is still performing the insertion. . . . . 75
4.10 The elbow joint velocities at runtime composition with and without the transient method. Discontinuous velocity is obtained when the skill is introduced without scaling the weights in the optimization problem. In contrast, a smoother velocity is achieved when the skill's weights are scaled. . . . . 76

5.1 The overall system architecture of the proposed system . . . . 84
5.2 The relational model describing a workpiece . . . . . 87
5.3 A sub-ontology modeling the relation between task and skill . . 88
5.4 Flowchart illustrating the steps in grounding from task to executable skill . . . . . 92
5.5 Relevant application parameters in a peg-in-hole problem applied to a rotor assembly . . . . . 93
5.6 Liaison diagram for the first assembly use case. Although the rotors are in contact with the lid, the chamfers allow us to use a place task defined between the housing and the lid. . . . . 94
5.7 The fuzzy sets used in the inference module. The left column shows the geometric parameters while the right one shows the process parameters. . . . . 96
5.8 The skill taxonomy as defined in the OWL ontology. . . . . 98
5.9 Example of the finite state machine for gripping and inserting a workpiece. The transition between two states is triggered by incoming event "e_event_name" . . . . . 99

5.10 CAD model of the air compressor. Note that the helical screw profile is not modeled. . . . . 102
5.11 The robot performing the assembly sequence of the compressor parts. From left to right: assembly of the small rotor, the big rotor, and the housing lid. . . . . 102
5.12 The measured forces and torques during insertion of the big rotor. Note that from 8 s, the rotor is in contact with the bottom. This was added to verify that the desired dithering is present. . . . 103
5.13 CAD model of the electrical motor. . . . . 104
5.14 KR16-2 performing the assembly sequence of the electrical motor. 104
5.15 The measured forces when inserting the rotor in the motor housing. Note that this example moves slower as it uses a lower gain to show *CylindricalInsert* behavior. The inset axes show the overshoot but eventual convergence to the desired insertion force. 105

6.1 Connections between this thesis contributions and other relevant parts shown in a broader view of robot skill programming and deployment. The contributions and their corresponding chapters are shown in rounded rectangles. The solid arrows signify how two entities are related. The dashed arrow signifies a potential connection that was not explored in this thesis. . . . . 111
6.2 Illustrating the limitation of the presented CAD-level assembly specification with a home switch assembly task. Top, left to right: the assembly steps as performed by a human. Bottom: magnification of the switch to show the feature pairs that are subject to CAD-level constraints. . . . . 117

# List of Tables


2.1 The list of tasks performed by the dual-arm setup in Figure 2.1. The HQP solver uses strict priority levels while the standard eTaSL solver uses weights. The "n/a" signifies no weights due to the task implementation as hard constraints. . . . . 29

3.1 Different skill classification based on their progress variable properties . . . . . 43
3.2 The task specification of two skills composed with the leader-follower synchronization pattern. The leader and follower skills are denoted with $\mathsf{s}_1$ and $\mathsf{s}_2$, respectively. . . . . 48
3.3 The task specification of two skills composed with the weight scaling pattern. The Cartesian motion skill is denoted with $\mathsf{s}_1$, while the admittance motion skill is denoted with $\mathsf{s}_2$. . . . . 49
3.4 Feature comparison between different skill-based programming frameworks . . . . . 53

4.1 Examples of a progressive and a non-progressive skills, their constraints specification and terminating condition. . . . . 64
4.2 Relevant information for the two assembly task scenarios. . . . 76

5.1 Examples of grounding maps between CAD constraints to the required skill constraints . . . . . 89
5.2 Geometric and Inferred Process Parameters for the Compressor Rotor Assemblies . . . . . 101
5.3 Geometric and Inferred Process Parameters for the Motor Assemblies . . . . . 105

# Chapter 1

# Introduction

## 1.1 Background

In a world where demands for customized and low-volume products are ever increasing, manufacturing companies are challenged to set up agile production systems that can quickly adapt to dynamic markets. Consequently, robots have become an increasingly important part of the companies' production lines. This is supported by a number of reports that show increasing annual sales of industrial robots, purchased by both large companies and SMEs accross different manufacturing sectors [28, 130].

In addition to the challenging market demands, various external factors such as economic downturns and uncertain government policies also affect manufacturing companies to various extents. One of the most prominent examples of these factors is the COVID-19 pandemic which is still ongoing at the time this thesis is written. The pandemic has transformed how manufacturing and supply chain operate all over the world. As suggested by several research studies, companies that can quickly stem the shortage and dislocations of workers by resorting to robotics and automation may have more competitive advantages and higher chances to endure the crisis [20, 22].

Despite the increasing trend of robot adoption by manufacturing industry, robot deployment is still confronted with a number of challenges. One of them is the high cost of deployment. This cost can be attributed to different aspects such as cost associated with the robot hardware and the supporting equipment as well as cost associated with comissioning the robotic systems [143]. One component that takes a large portion of commissioning cost is the cost associated with

robot programming [97].

Robot programming is typically performed by experts which can be difficult to find and also costly to hire. Furthermore, depending on the application complexity, programming can take weeks, even sometimes months [143]. For mass manufactured products with relatively long life cycle such as cars, such high cost can be justified. However for products with shorter life cycle and smaller lot sizes, which are often the cases for SMEs, this becomes a barrier that prevents robot adoption.

The high programming cost is in contradiction to the growing needs to automate tasks in increasingly challenging scenarios. Today, robots are demanded to perform tasks that are subject to disturbances, human interactions, and unstructured environments. Robotic systems need to be robust against variations in the environment and the workpieces they manipulate. Often, these tasks are not feasible to perform with blind motions, but require motions that adapt to sensor input instead. As these complexities increase programming cost, we therefore need to re-evaluate how robots should be programmed for such scenarios.

One problem with conventional robot programming methods is the high level of expertise needed in specifying the tasks. In conventional methods, robots are programmed in terms of low-level motion commands such as *point-to-point* (PTP) or impedance motion commands. While these methods are designed and optimized to program simple tasks, they are less effective in programming complex tasks that require reactive motions based on sensor inputs and combination of multiple behaviors. Figure 1.1 illustrates this problem with two example tasks: a free-space motion task that can be easily programmed with a traditional approach and an involved sensor-based task that requires a different programming approach.

The left part of Figure 1.1 shows simplistic robotic tasks for which conventional programming methods are well suited. Typically, these tasks can be performed by position control, where the paths or target positions are predetermined and static during execution. Although more advanced robots such as KUKA LBR IIWA provide more sophisticated impedance commands that can adapt the paths based on measured forces, their programming languages are not yet designed to easily incorporate other types of sensors nor program different types of reactive behaviors. Consequently, programming more complex tasks with traditional methods can require significantly more efforts and highly trained experts/specialists.

Another problem with robot programming is the lack of common programming language/methods. Each robot manufacturer offers their own programming

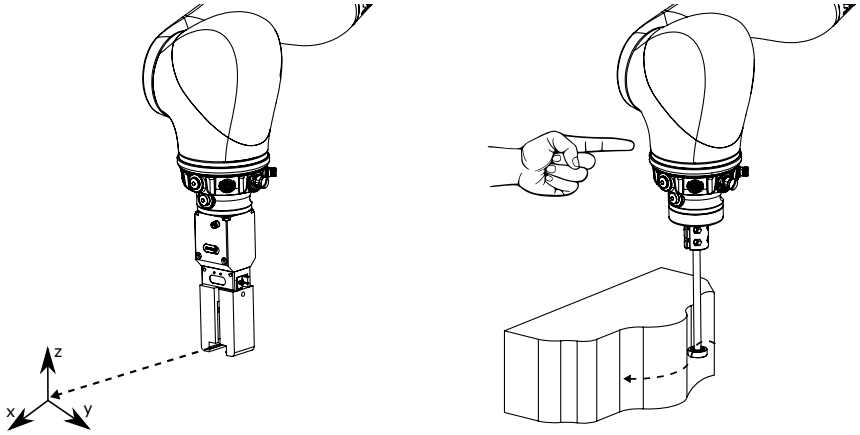



Figure 1.1: Examples of a simple task (left) and a complex task (right). The simple task is to perform a linear motion that moves the robot's end-effector to a desired target pose in free space. Conventional robot programming methods are suitable for such tasks due to the rudimentary behavior. In contrast, the complex task is exemplified by an unknown-contour following task in which the robot has to maintain a desired contact force and react to human-induced disturbance. Such a task would be too difficult to program using conventional methods due to the composition of different behaviors and involvement of sensor inputs.

language that is incompatible with the other robot vendors. For example, KUKA robots are programmed in KRL and Java (Sunrise) [85], ABB robots are programmed in RAPID [1], while FANUC robots are programmed in Karel [63]. Unfortunately, there is no single vendor that offers solutions to all robotic problems. For example, a vendor may offer robots that excel in heavy-payload tasks, but does not offer robots that are suitable for human-robot collaboration. Therefore, it would be a disadvantage for companies to commit to only one particular vendor. However, adoption of multiple vendors is often costly, due to the high learning curve in acquiring the expertise in native robot programming methods. The lack of common programming method also prevents reusability of robot programs. Consequently, deploying similar tasks to different robot vendors often translates to programming from scratch, with little to no reusable codes.

The aforementioned problems contribute to the relatively high cost and long duration to commission a robotic system. Therefore, a different approach to robot programming is desirable. In general, this thesis aims to alleviate the above problems by proposing methods to program sensor-based robot behaviors that

require a lower level of expertise than conventional methods, while improving the reusability of the programs.

## 1.2 Related Works

In order to transform robots into a flexible tool that non-experts can program and deploy with less effort, the robotics research community needs to invent innovative programming approaches. This calling is shared by many scholars in the community, resulting in a number of seminal works that encourage follow-up research. Following are the key ideas that inspire and set the foundation of the work in this thesis.

### 1.2.1 Skill-based robot programming

Recently, there is an emerging paradigm to program robotic tasks in terms of reusable skills instead of lower-level motion primitives. Generally speaking, *skill* can be defined as a robot capability to perform a *task*. Different skill-based programming frameworks have been developed by various robotics communities: universities, research institutes and companies. While the formal representation and specific implementation of the skills and the programming frameworks may differ, their main *raison d'être* remains the same: to lower the complexity in robot programming such that it becomes more accessible for non-experts. In general, a skill has the following characteristics:

1. A skill implements a specific robot capability to perform a task. This capability can be simplistic and with similar complexity to that of conventional robot motion commands, e.g., PTP motions, but can also be more complex such as contour following or force-based assembly.

2. The task corresponding to a skill is specified with a higher level semantics than conventional robot motion commands. For example, a task can be specified in terms of a desirable objectives that should be satisfied instead of reference trajectories.

3. A skill exposes a set of configurable parameters such that it can be reused for similar tasks. For example, a peg-in-hole skill may come with configurable insertion force, insertion depth and desired stiffness so that it can be reconfigured to different peg geometries and materials.

4. A skill has a set of monitoring mechanisms. A monitor evaluates certain conditions and triggers an event that can be used to, for instance, signal the success or failure of a skill.

5. Multiple skills can be composed in order to perform increasingly complex tasks. Typically, a composition can be specified to combine skills either in a sequential or concurrent manner*. Sequential composition results in skills that are executed one after another, while concurrent composition results in skills that are executed simultaneously.

In addition to the above, skill-based programming frameworks often come with a *graphical user interface* (GUI) that allows skills to be programmed without coding.

### Skill-based Programming Research in Academia

Skill-based robot programming has been a topic of interest for universities and research institutes. Different aspects of robot skills present non-trivial topics for research. Some of the frequently studied aspects include skill modelling [95, 131], methods to compose skills [114, 92], task specification and control frameworks for skill implementation [3, 77] and methods to combine higher-level semantics to skills [11].

Skill modelling deals with designing a formal representation of a skill. Some literature focuses on a conceptual skill representation that describes the minimum requirements of a skill. For example, [95] proposes an abstract skill model that consists of an execution program, configurable parameters, and monitors that check the pre- and post-conditions. The model is quite general and does not specifically address the underlying task specification nor the controller. Instead, this decision is left to the users to decide an implementation that is seen fit. Other modelling efforts put more emphasis on the underlying task specification and execution rather than the abstract representation. For example, the skill models proposed in [93, 90] commit to a specific approach for the task specification and control aspect. By taking this decision, the skill model becomes less general. However, in return, some issues such as composability of the skills can be treated in a more concrete manner.

*Throughout this thesis, concurrency means that two or more tasks are in progress at the same time. Here, we do not make any assumption on whether the tasks are being processed with a single or multiple processors. Other domains such as parallel computing often distinguish between concurrency and parallelism, where the latter might be more suited for a certain context [91].

The skill composition aspect deals with methods to combine skills such that certain desired composed behaviors can be achieved. These compositions can be as simple as ordering multiple skills in a sequence such that one skill is executed after the preceding skill terminates. For example, a composed skill that performs an assembly task may consist of a sequence of skills to grasp the part to be assembled, bring the part close to its mate and assemble the part. A more complex case is when skills are composed concurrently, i.e., when multiple skills have to be executed simultaneously. Such composition presents a more difficult challenge for execution since the skills may use the same resources, e.g., sensors or actuators. For example, a trajectory following skill that is composed with a collision avoidance skill may actuate the same robot joints. As a result, when an obstacle is present in the trajectory, the composed behaviors will be conflicting. Therefore, the underlying task specification and control framework should be capable of handling these conflicting behaviors by appropriately distributing the allocation of resources, e.g., by computing the contribution of each skill to the actuation input.

Task specification concerns methods to describe the task as a set of robot-readable objectives that the skill should achieve. Meanwhile, the task controller deals with translation of the specification into robot control inputs. Together, the task specification and control results in motions that make the robot "skillful". Most of skill-based programming frameworks advocate a vendor-independent approach, such that the same task specification can be reused for robots from different manufacturers. This way, skill programmers can be liberated from having to master robot-specific programming languages. There are different methods to achieve vendor-independent task specification. For example in [106], the tasks are specified in terms of motion primitives that can be performed by a motion planner such as MoveIt! [21]. Motion planners compute reference paths that steer the robot from an initial to a target configuration while satisfying constraints such as avoiding collisions and joint limits. The generated paths are robot-agnostic and can in turn be executed using standard robot motion APIs.

Another alternative is to specify the tasks using a constraint-based task specification approach. Instead of specifying in terms of desired paths/trajectories, tasks are specified in terms of constraints. Different behaviors or modes can be expressed as constraints. For example, a constraint to avoid collision with the environment and a constraint to reach a target pose. Typically, constraint-based task specification is also implemented together with the complementary constraint-based controller. Given a set of constraints and an objective function, the constraint-based controller is responsible to generate the optimal robot control input. Depending on how the controller is formulated, the control input can be velocity resolved [3], acceleration resolved [135] or torque resolved [117]. Similar to a motion planner, these control inputs can be executed by

standard industrial robots' motion APIs, therefore making this approach vendor-independent. Examples of skill-based programming frameworks that adopt a constraint-based approach for their underlying task specification and controller are presented in [90, 93].

Combining higher-level semantics to robot skills is also an active research topic. Higher-level semantics contains symbolic information such as objects being manipulated or the required tools. One motivation of this research topic is to enable reasoning about the low-level motion, such that the overall robotic system becomes more intelligent and autonomous. For example, [128] shows that skills can be composed and configured automatically given a high level task description and by reasoning on the knowledge base.

### Skill-based Programming Frameworks by Industry

The potential of the skill-based programming paradigm has caught the attention of industry as the number of companies that offer skill-based frameworks is increasing. Skill-based robot programming in industry is mainly provided by either robot manufacturers or companies specializing in developing robot programming solutions. The latter typically do not manufacture their own robots. Rather, they offer stand-alone products that can be integrated with various robot brands and, sometimes, a complete integrated solution that already includes robots from existing manufacturers.

Examples of skill-based robot programming frameworks by robot manufacturers are ABB's Wizard Easy Programming [2] and Franka Emika's Desk [38] (see Figure 1.2, left). A drawback of these frameworks is that they are designed to work exclusively with the vendors' own robots. Nevertheless, these frameworks have lowered the threshold for non-experts to program robot skills since they do not need to understand nor code in the robot programming languages.

Vendor-independent frameworks are provided by companies who specialize on commercializing robot programming software. A few examples are Artiminds Robotics's RPS [5], Ready Robotics's Forge-OS [104] and drag and bot's OS [32] (see Figure 1.2, right). These solutions are designed to be robot-agnostic such that the same skills can be reused and reconfigured to different robot brands. Some of these frameworks, such as Artiminds' RPS and drag and bot's OS, generate native robot codes that can be uploaded to and executed by the robot controller. Consequently, there is no need for an external controller to run the skills during the task execution phase. Other frameworks such as Ready Robotics's Forge-OS do not generate native code, but instead send motion commands to the robot controller. The skills are executed by an external

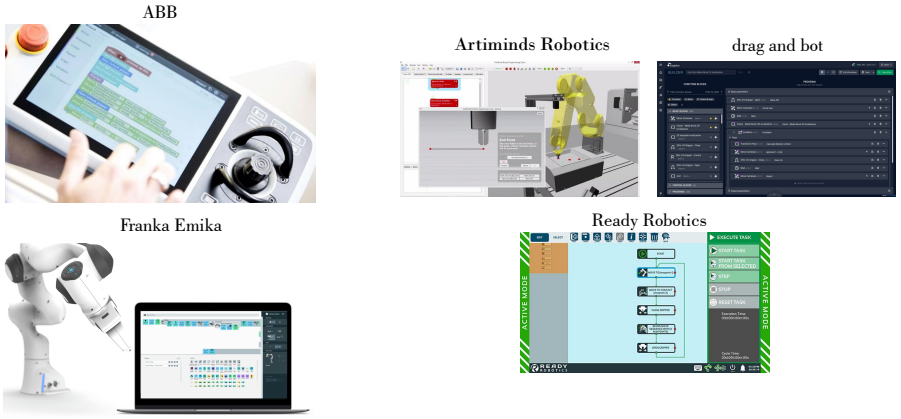



Figure 1.2: Examples of skill-based programming solutions offered by industry. Left: robot manufacturers' solutions represented by ABB and Franka Emika. Right: non-robot manufacturers' solutions represented by Artiminds Robotics, drag and bot and Ready Robotics.

controller that needs to be connected to the robot controller during production phase.

All aforementioned industrial solutions offer code-free methods to program robots using visual programming via dedicated GUIs. Programmers can instantiate skills using existing templates or create new custom skills for more specific applications. They also offer the facility to compose skills. However, the compositions are typically relatively simplistic or restricted to discrete level, e.g., by pure sequencing or applying state machines. One particular limitation of industrial solutions is the lack of support for concurrent skill composition, in which multiple skills have to performed simultaneously.

### 1.2.2 CAD-based robot programming

Computer-aided design (CAD) has long been used as a standard in industry to create digital models of products. As CAD software become more capable and advanced, the product models contain increasingly complex information that not only contains geometric data, but also materials, assemblies and processes. This has initiated research works that aim at assisting and automating some aspects of robot skill programming by leveraging the information embedded in CAD models.

The idea of using CAD models in skill programming has found a variety of use cases; two notable examples are path generation and assembly. In path generation, geometric information of the workpieces, robots and workcells are extracted from the CAD models such that motion planners can compute collision-free paths to perform the given tasks. This is commonly applied to industrial applications such as welding [66], deburring [86] and additive manufacturing [31]. Furthermore, commercial software such as Siemens Kineo [119] are already available in the market and used in real production lines.

In assembly use cases, the ultimate goal is to achieve a fully automated assembly system that takes CAD models as an input and generates a complete sequence of robot skills to perform the assembly tasks. This is a challenging problem that requires interaction of different aspects such as assembly planning to compute feasible assembly sequences, world modelling and sensing to locate the robots and workpieces to be assembled, as well as refinement and execution of robot skills to perform the assemblies. Due to its complexity, existing literature typically only focuses on a specific part of the overall system.

Algorithms for assembly planning have been addressed in literature since as early as 1970s, with pioneering works such as AUTOPASS [72] and more recent works such as HighLAP [133]. Assembly planning results in an assembly sequence that is typically modelled as a graph where each edge connects two vertices representing parts to be mated. Assembly graphs are often computed without taking into account robot limitations and the edges are only annotated with symbolic actions. Since actions in assembly graphs are too abstract and therefore can not be executed directly by the robots, a refinement to concrete skills is necessary.

Refining abstract actions to executable skills is influenced by many factors such as the robot types, workpieces, and toolpieces. Therefore, the refinement process should ideally take all these relevant factors into account to increase the probability of successful task execution. Furthermore, the CAD format also influences how the refinement is performed. For example a STEP format can store exact mathematical models of geometric features such as surfaces and edges, whereas mesh format such as STL lacks this capability. Some proprietary CAD formats such as Solidworks' SLDASM facilitate even higher fidelity models that also store the specific assembly constraints between geometric features.

Existing literature exhibits different views and methodologies to handle the refinement from abstract actions to concrete skills. For example in [134], Thomas, et al. propose to refine assembly graphs into skills that are specified with the task frame formalism [19]. They apply their work to mesh-based CAD models that do not contain any assembly constraints information. Perzylo, et al. developed a framework that makes use of a more refined CAD model

based on BREP [98]. The BREP-based model can be annotated with assembly constraints information such as a concentric constraint between two cylindrical surfaces. This makes refinement from abstract assembly actions to concrete skills easier since each assembly constraint can be mapped into an appropriate set of robot motion constraints [122].

Despite a long history of research and development, the use of CAD models for assembly is not yet at the same level of maturity as that of path generation and the overall industry uptake is still quite limited. This is mainly due to the difficulty in refining from CAD-level assembly specification to appropriate robot skills. Assembly is a very complex task whose execution is heavily influenced by the context at hands. For example, similar assembly tasks may require completely different robot skills depending on the insertion clearance or the presence of chamfers. Loose clearance assemblies can be performed by position-based skills, while tight clearance assemblies may require sensor-based skills that control the contact forces. Even when the appropriate skill is known, configuring it with the correct parameters is also a challenging problem that is again context-dependent. This is in contrast to path generation use cases where the task is usually executed by a trajectory following skill that moves the robot in free-space with minimal to no contact with the environment.

## 1.3 Research Goal and Objectives

The overall goal of this thesis is:

**To reduce the commissioning effort and improve the robustness of robotic applications in less structured, dynamic environments where geometric uncertainties and unpredictable disturbances can be present.**

In order to achieve this goal, the following objectives are defined:

1. **Objective 1:** To build up a set of skills that shows improved robustness of skill execution and extend the skill composability. This set of skills allows: *i) continuous reactive behaviors* by adapting the robot motion according to sensor signals in real-time in order to handle non-nominal conditions, e.g., presence of disturbances. *ii)* composability, i.e., combining multiple skills concurrently in order to form a desirable composed behavior.

2. **Objective 2:** To endow robots with *discrete deliberative behaviors* in order to deal with a high variability in the environment and disturbances that require further adaptation beyond the continuous-level behaviors. By

delegating this adaptation capability to the discrete level, the complexity at the continuous level can be reduced. The desired discrete deliberative behavior should be specified using a planning domain definition language and the resulting plan should be able to be executed in terms of continuous reactive skills.

3. **Objective 3:** To reduce the level of expertise required in programming robot skills by separating task specification and execution. Concretely, the thesis aims to facilitate task specification in a higher-abstraction layer such that non-experts in robotics and control are able to specify a task to achieve and instantiate the corresponding skills without having a deep knowledge in the lower-level skill implementation. This task specification should be implemented in a workpiece-centric representation that contains sufficient information to enable generation of robot-executable specifications in a semi-autonomous manner.

## 1.4 Overall Approach of the Thesis

The advantages offered by the skill-based paradigm makes it a promising approach to achieve the objectives stated above. Therefore, this paradigm is adopted as the core subject in this thesis. Furthermore, this thesis favors constraint-based task specification as the underlying approach to implement the skills. This choice is motivated by the fact that constraint-based skills facilitate reactive and composable robot behaviors, therefore improving the robustness of the task execution with respect to uncertainties and disturbances as well as enabling a wider variety of tasks.

Covering the whole aspect of skill-based programming in detail would be a daunting subject to address in a PhD thesis. In order to make noteworthy contributions, this thesis aims at developing specific solutions that focus on some of the key problems in programming and deploying constraint-based skills. These problems are identified from the limitations present in the state of the art literature. Following are the steps taken as the overall approach in this thesis.

- **Propose a constraint-based skill model that allows for reactive behaviors in non-nominal conditions**. A conservative assumption in most of the existing literature is that the skills are typically performed in nominal conditions with little to no disturbances. This assumption may no longer be valid for realistic scenarios. Therefore, in line with **Objective 1**, this thesis proposes a model of constraint-based skills that allow reactive deviations from their nominal behaviors in order to handle disturbances.

- **Design methodologies to compose constraint-based skills**. As the tasks to perform become increasingly complex, robots may need skills that are not confined to singular behaviors. For example, tasks that are performed in the presence of disturbances require skills that simultaneously exhibit nominal and reactive behaviors. In addition, some tasks may also require skills to run synchronously in order to achieve certain objectives such as when two robots cooperatively transfer an object. These examples require the facility to compose multiple skills concurrently while adhering to well-defined composition rules. Therefore, as a further step to achieve **Objective 1**, this thesis should provide methodologies to compose skills, i.e., how to combine constraint-based skills in order to achieve a desired composed behavior.

- **Integrate low-level reactive skills with a high-level deliberative planner**. A composition of skills can endow robots with reactive behaviors that can handle disturbances at the continuous level such as dynamic obstacles. However, in realistic scenarios, the occurence of disturbances can be unpredictable, therefore preprogramming the appropriate skill compositions in advance may become inconvenient or even infeasible. Relevant examples are human-robot interaction applications, where human intentions and actions are difficult to estimate and predict. Disturbances may appear suddenly in the workspace, rendering the currently executed skills unsuitable anymore. The solution to this problem requires that the skill compositions are adaptable during the task execution. Furthermore, it is desirable that this adaptation is computed automatically, such that the robots can handle a variety of disturbances in a more autonomous manner. In order to address this issue, this thesis proposes an integration of reactive skills with a high-level deliberative planner. The planner serves to endow robots with deliberative behaviors at the discrete level by automatically reconfiguring the skills upon failures. This overall approach is a step to accomplish **Objective 2**.

- **Develop a method to program skills based on CAD-level task specification**. As an approach to achieve **Objective 3**, this thesis proposes the use of CAD models and CAD interface to reduce the effort in skill programming such that it is more accessible to non-experts. This direction is motivated by the ubiquity of CAD models in industry and industry familiarity with CAD modeling and interfaces. Instead of specifying the tasks in lower-level constraints such as forces or motions, this thesis explores a way to specify the tasks in higher-level CAD constraints. From the CAD-level task specification, the suitable skills are inferred and parameterized in a semi-automated manner by leveraging the geometric information embedded in the CAD models.

## 1.5 Contributions

This thesis is composed of three main contributions; each presents a solution that corresponds to the aforementioned objectives. In addition to the three main contributions, this thesis also presents a formulation and software implementation of a solver that supports hierarchical tasks. This additional contribution extends the solver library of the underlying constraint-based task specification framework used in this thesis.

### 1.5.1 Contribution 1: A framework to program composition of reactive and synchronized robot skills

The first contribution presents a constraint-based skill model and methodologies to compose skills. In contrast to the existing work on constraint-based skills that treat the constraints uniformly, the proposed model introduces the notion of skill progress and makes distinction between *task* and *progress* constraints. The proposed model allows to specify a variation of deviating behaviors during non-nominal condition, e.g., in the presence of disturbances. For example, the deviating behavior can be tuned to weigh more towards keeping the robot on the constrained path or keeping the robot on track with the desired progress. Furthermore, the framework introduces *composition patterns* that can be used to combine skills concurrently to form a composed skill. Each composition pattern corresponds to a specific composed behavior such as progress synchronization and superposition of reactive behaviors.

Combination of the constraint-based skill model and the composition patterns facilitate programming of reactive and synchronized skill compositions. As a result, an extended composability is achieved and the robustness of the skill execution with respect to non-nominal conditions is improved. This contribution is in line with **Objective 1**.

### 1.5.2 Contribution 2: A framework for integrating task planning and reactive skills to enable autonomous runtime skill composition

The second contribution concerns **Objective 2**. It presents a framework that integrates symbolic task planning as a deliberative layer to automatically compose constraint-based skills. Here, the role of task planning is twofold. Firstly, it computes a feasible plan, i.e., a sequence of actions that are refined into a sequence of constraint-based skills that achieves a given goal. Secondly,

it recomputes a new plan whenever failures, caused by unforeseen disturbances, are detected. This replanning is performed during execution time without interrupting the ongoing skills. A model of planning actions based on temporal planning in PDDL2.1 is proposed to ensure concurrency. The new plan consists of concurrent actions that are refined into composed skills. A composed skill comprises both a nominal skill and reactive skills to ensure disturbance rejection behaviors. Discontinuity in the control input due to the suddenly changed skill composition is mitigated with a continuous task transition method, resulting in smooth transients.

The main advantage offered by the framework is that it endows robots with a deliberative capability to automatically reconfigure the skills in scenarios where occurences of disturbances are not known in advance. This capability can help robots in dealing with a large variety of disturbances and different conditions that may happen during the task execution. Furthermore, the high level description in PDDL also makes it easier to reconfigure the task, therefore reducing the overall commissioning effort. Combined with the advantage of constraint-based skills that can deal with continuous level disturbances in real-time, the framework improves the overall robustness of the robotic applications.

### 1.5.3 Contribution 3: CAD-based task specification and skill programming for assembly applications

The third contribution is aligned with **Objective 3**. It presents a system architecture and software implementation to semi-automatically generate sensor-based skills from CAD-level task specification for assembly applications. The system architecture consists of three layers:

- **application layer** facilitates the specification of assembly tasks in a higher-level CAD domain
- **process layer** performs *skill selection* and *skill parameterization* for the given task specification. The former deals with finding suitable skills by reasoning on a knowledge database implemented in Web ontology language (OWL). The latter deals with generating the appropriate skill parameters based on the CAD-model of the workpieces.
- **control layer** executes the instantiated skills with a constraint-based controller.

This contribution allows programming skills based on CAD-level specification while automating as much as possible the low-level programming by leveraging

on the CAD data and constraint-based task specification. As a result, it lowers the barrier for non-experts to program robot skills that otherwise requires a deep level of expertise in low-level robot motion programming.

## 1.6 Thesis Outline

The outline of this thesis is given as follows.

- **Chapter 2** presents a preliminary background on expressiongraph-based Task Specification Language and Controller (eTaSL/eTC) [3]. This is the underlying constraint-based task specification and control framework used to implement the robot skills. This chapter starts by giving a concise overview of constraint-based task specification literatures. Subsequently, it dives deeper into the theoretical background of eTaSL. Finally, this chapter also presents an extension to the eTaSL solver to support hierarhical tasks as an additional contribution of this thesis.
- **Chapter 3** presents a framework for programming composition of reactive and synchronized robot skills, i.e., **Contribution 1**. It starts by explaining a model of constraint-based skills and introduces the notion of skill progress. Subsequently, methods to compose skills concurrently that produce reactive and synchronized behaviors are discussed. Finally, it presents an evaluation of the framework in a dual-arm assembly task of an industrial compressor.
- **Chapter 4** presents a framework of integrated task planning and reactive skills for autonomous runtime skill composition, i.e., **Contribution 2**. First, it recaps the skill model and presents a method to perform runtime skill composition that ensures smooth transients in the control input. Second, it explains the symbolic task planner that automatically computes the skill composition. In order to ensure that the solver builds fully concurrent actions, a temporal planning model for symbolic actions is proposed and described. As an experimental evaluation, a dual-arm assembly task of solenoid valves into an industrial pump in the presence of disturbances is presented.
- **Chapter 5** presents a system architecture for CAD-based task specification and skill programming for assembly applications, i.e., **Contribution 3**. The chapter starts by presenting the overall system architecture and describing each building block. Next, the proposed methods to refine from CAD-level task specification to executable skill instances are discussed. Subsequently, the chapter explains the software implementation

of the system architecture. Finally, evaluation of the implemented system in a compressor assembly task performed by two different robot setups is shown and the results are analyzed.

- **Chapter 6** discusses several insights obtained from the development of this thesis' contributions. It also discusses how the proposed methods are connected and their distinct roles in programming robotic tasks. Furthermore, it describes the limitations of the proposed methods.
- **Chapter 7** concludes the thesis by recapitulating the objectives, summarizing the contributions and providing suggestions for the future work.

# Chapter 2

# Constraint-based Task Specification for Sensor-based Robot Skills

This chapter presents theoretical preliminaries on constraint-based task specification. First, a brief introduction to constraint-based task specification and an overview of existing works in robotics literature are discussed. Subsequently, the focus is shifted to *expressiongraph-based task specification language* (eTaSL), a state-of-the-art constraint-based task specification and control framework that is selected as the underlying framework to implement robot skills in this thesis. As an additional contribution, this chapter also presents an extension to eTaSL solver in order to allow specification of tasks with arbitrary levels of priorities.

## 2.1 Overview of related works

Constraint-based task specification presents a different paradigm to program robot motions compared to conventional robot programming methods. Instead of specifying motions in terms of target trajectories or poses, motions are specified in terms of constraints. These constraints can capture different aspect of the overall application such as constraints related to environments, robot limitations and the tasks themselves [78].

The set of constraints form a "high-level" task specification that is then automatically translated into "low-level" robot motion commands such as desired joint velocities or accelerations. Since task specification programming is decoupled from low-level controller programming, this paradigm allows programmers to focus more on designing the specification of the problems at hand and be less bothered with the low-level control implementation. A task specification can be under or over-constraining with respect to the available *degrees of freedom* (DOF), making computation of a unique solution non-trivial. Thus, the translation to robot motion command is often formulated as a constrained optimization problem that minimizes a certain cost function such as weighted square of joint velocities [3] [50].

Constraint-based task specification offers a number of advantages compared to conventional programming methods; two main advantages are: *i)* improved flexibility, *ii)* higher reusability of the robot program, and *iii)* reactivity with respect to online sensor measurements. Flexibility is improved in the sense that changes in the scenarios can be simply addressed by reconfiguring constraint parameters or adding/removing constraints. In contrast, traditional programming methods that uses *point-to-point* (PTP) motions often require reprogramming the whole trajectory. Reusability is acquired thanks to the composability property of constraints that makes it possible to separate constraints related to the actual tasks and the robot descriptions such as kinematic models and joint limits. As a result, the same task specification can be reused across different robots and workcells. Meanwhile, reactivity to variations and disturbances observed by the sensors can be achieved. This is due to the possibility to incorporate sensor information and to compute the optimal robot motion command that satisfies the constraints in real-time.

Research related to constraint-based task specification has started since decades ago. In 1970s, Ambler and Popplestone already show how the desired relative pose between two bodies can be inferred from a "high-level" specification of geometric constraints between features such as "fit" and "against" [6]. However, the paper does not provide concrete applications in robotics. One of the earliest work that incorporates constraint-based task specification for robotic tasks is a paper by Mason [78]. The paper presents a methodology to specify compliant robot motion and introduces the notion of *natural* and *artificial* constraints. In [19], Bruyninckx and De Schutter provides a formalism to synthesize compliant motion specification based on the compliant frame introduced in [78]. The formalism is later widely recognized as the *task frame formalism* (TFF). The paper also explains situations in which the TFF is not suitable for. One common trait in these publications is that the authors mainly address the task specification aspect and isolate it from the underlying controller implementation.

The above seminal papers lay the foundation for more recent works that offer

more general and refined solutions. In more recent papers, the task specification aspect is usually addressed along with the control/execution aspect, forming a more complete framework. In [113, 120] De Schutter, et al. present *instantaneous Task Specification using Constraints* (iTaSC), a framework that facilitates task specification for general sensor-based robot systems and estimation of geometric uncertainties. In iTaSC, a task is modeled with six DOF *virtual kinematic chains* (VKC)s. The controller is formulated as a first order dynamic, velocity-resolved control scheme using weighted pseudo-inverse method to handle conflicting constraints. The *stack of task* (SoT) framework of Mansard et al. uses a graph representation of computational entities to specify the task [75, 77]. The authors adopt the null-space projection method that allows strict priority levels between tasks. Such method ensures that lower priority tasks will never violate higher priority tasks. Another notable work is the *whole body control* (WBC) framework by Sentis and Khatib [58, 116, 117]. Similar to SoT, the framework uses the null-pace projection method, however, with a torque-resolved control scheme. The work is originally motivated by humanoid robot applications, which typically group constraints into three priority levels corresponding to the safety, operational tasks, and postures constraints.

One of the more recent framework is eTaSL by Aertbeliën and De Schutter [3], which can be viewed as a successor of iTaSC. eTaSL improves upon iTaSC by having a better separation between the task specification and execution as well as a more flexible *domain specific language* (DSL). Furthermore, in contrast to iTaSC that follows a strict methodology to describe a constraint with a six DOF VKC, eTaSL relaxes this requirement which makes it easier to specify tasks. A more detailed overview of eTaSL and the underlying mathematical formulation is presented in the next section.

## 2.2 eTaSL Framework

### 2.2.1 Overview of eTaSL

eTaSL is an integrated framework of constraint-based task specification and execution which is intended for continuous-level control applications. As such, it distinguishes itself from discrete-level executive frameworks such as rFSM [60] and SMACH [15]. The task specification aspect is facilitated with a DSL which is based on Lua [48]; the task execution is handled by translating the specification into a QP problem that is solved by an off-the-shelf solver.

eTaSL enables programming of sensor-based robot controls. Furthermore, eTaSL deals not only with the nominal tasks, but it can also accurately define how to

deviate under influence of disturbances and variations. For example, in [138], eTaSL is used to specify a reactive approach motion that allows deviation within a manifold.

One of the design features in eTaSL is the *separation of concerns*. With regards to the task specification, this allows programmers to separate robot-related, environment-related and task-related aspects of the specification. The robot-related specification encompasses constraints imposed by the robot hardware such as maximum velocities, collision properties and joint limits. The information can be extracted from a robot description file such as URDF [52]. The environment-related specification includes constraints imposed by the robot surroundings such as workcell volume and obstacles. Finally, the task-related specification consists of constraints specific to the tasks at hand such as reaching a target pose or applying a desired force. This way, eTaSL offers a flexible way to adapt the robot application by only changing a subset of the robot program.

Similar to iTaSC, eTaSL retains the use of *feature variables* as a distinctive attribute, in comparison to other frameworks such as SoT and WBC. The feature variables serve as auxiliary decision variables in addition to robot joint variables. They are useful to construct non-physical entities such as virtual kinematic chains. A typical application of feature variables is to provide additional degrees of freedom in a path such that it allows a reactive devation in non-nominal conditions.

### 2.2.2 Elements that constitute an eTaSL specification

Followings are the entities that form an eTaSL task specification.

1. **Variables** consist of: *i)* the time variable *ii)* joint variables that correspond to the actuated joints of robots *iii)* feature variables as auxiliary variables to specify additional DOF. Both joint and feature variables can be explicitly defined and assigned by names.

2. **Expressions** are functions of variables that are stored in a directed acyclic graph called *expression graphs*. These expressions can be functions that correspond to physical entities such as a robot tool frame, but they can also correspond to non-physical entities such as parameterized trajectories. One of the key features of eTaSL is the ability to efficiently calculate the derivatives of expressions with respect to the variables by means of automatic differentiation [103]. This aides the automatic generation of a controller during task execution.

3. **Constraints** are imposed on expressions and can be specified as either position-level or velocity-level. eTaSL supports both equality and inequality constraints by assigning target values and desired bounds, respectively. Furthermore, programmers can set a certain level of importance to a constraint. In contrast to other frameworks such as SoT* [77] and WBC that use strict priority levels, eTaSL allows both priority levels and weights. Priority levels are integer numbers; they ensure that constraints with lower priority never violates those with higher priority. Weights are scalar numbers; they allow trade-offs between constraints in the same priority level.

4. **Monitors** handle the discrete-level feedback of the task execution. Monitors continuously evaluate eTaSL expressions and trigger events once their conditions are met, e.g., when a measured force exceeds a predefined threshold. Monitors can be used to notify successful and failed execution of the task to a higher-level coordinator or scheduler, but also other types of discrete events.

5. **Input and Output Channels** serve as a mechanism to feed incoming data such as sensor signals to the eTaSL controller and query the values of expressions (and their derivatives), respectively. The framework itself does not deal with the specific type of sensor or the details of sensor acquisition. One can generally define input and output channels and connect external component to them.

6. **Context** is a data structure that stores the whole task specification. During execution, this context is translated into an instantaneous optimization problem that is fed to an underlying numerical solver.

### 2.2.3 Optimization problem formulation and solver

This subsection explains the translation of the task specification constraints to a QP optimization problem with *numerical constraints*. The eTaSL controller is a task execution controller that *instantaneously* computes the most optimal control input given a set of constraints and it solves a local optimization problem. This distinguishes it from controllers that use a global optimization approach such as motion planner or predictive approach such as *model predictive control* (MPC).

*The SoT framework as presented in [77] uses solely priority levels to resolve conflicting tasks. However, the software implementation of SoT [40] provides additional possibility to use weights.

At each timestep, eTaSL translates the task specification into a numerical optimization problem of the following form:

$$\underset{x}{\text{minimize}} \quad x^\top \mathbf{H} x \tag{2.1a}$$

$$\text{subject to} \quad \mathbf{L}_A \leq \mathbf{A}x \leq \mathbf{U}_A, \tag{2.1b}$$

$$\mathbf{L} \leq x \leq \mathbf{U} \tag{2.1c}$$

This is a QP optimization problem, i.e., a quadratic objective function subject to linear equality or inequality constraints. The optimization variable $x$ is a vector of the following elements

$$x = [\dot{q}^\top \dot{\chi}_f^\top \varepsilon^\top]^\top$$

where $\dot{q} \in \mathbb{R}^{n_r}$ is the robot joint variable velocities, $\dot{\chi}_f \in \mathbb{R}^{n_f}$ is the feature variable velocities and $\varepsilon$ is slack variables that are introduced to facilitate soft constraints. $\mathbf{H}$ denotes the Hessian of the QP optimization problem. $\mathbf{L}_A$ and $\mathbf{U}_A$ denote the lower and upper bound vector, respectively, of the linear constraints described by a matrix $\mathbf{A}$. The rows of the matrix $\mathbf{A}$ contain the Jacobians of the task constraints. The vectors $\mathbf{L}$ and $\mathbf{U}$ are lower and upper bounds on the state variables $\dot{q}$ and $\dot{\chi}_f$. These can typically be handled more efficiently than general linear constraints by the underlying numerical solver. This type of constraints is typically used to limit the joint and feature variables velocities.

The specification in terms of variables, constraints and expressions is automatically translated into above numerical problem in the following way. A task constraint is satisfied by reducing the error according to a first order dynamical system. For each scalar constraint, the general formulation of the task error dynamic is:

$$\frac{d}{dt} e_i(q, \chi_f, t) = -k e_i(q, \chi_f, t) \tag{2.2}$$

where $k \in \mathbb{R}_{\geq 0}$ denotes the proportional feedback gain and $i$ denotes the constraint index. The task error expression $e$ is a function of joint positions $q$, feature variables $\chi_f$ and time $t$. The formulation of the task error $e$ is adopted from the task function approach proposed by Samson, et al. [107], except that it explicitly specifies the dependency on $t$.

In order to construct the vectors and matrices of the QP optimization, the total time derivative of $e$ in (2.2) is expanded as follows:

$$\frac{\partial e_i}{\partial t} + \sum_{j=1}^{n_r} \frac{\partial e_i}{\partial q_j}\dot{q}_j + \sum_{l=1}^{n_f} \frac{\partial e_i}{\partial \chi_{f,l}}\dot{\chi}_{f,l} = -ke_i \tag{2.3}$$

The first partial derivative term $\partial e_i/\partial t$ can be interpreted as a feedforward term to avoid tracking errors for task functions that explicitly depend on time. The second and the third partial derivative terms, $\partial e_i/\partial q_j$ and $\partial e_i/\partial \chi_{f,l}$ , denote the elements of the Jacobian of $e_i$ with respect to the joint and feature variables.

Slack variables $\varepsilon_i$ can be introduced to above equation to obtain lower priority constraints. Rewriting equation (2.3) gives us:

$$\mathbf{J}_i \begin{bmatrix} \dot{q} \\ \dot{\chi}_f \end{bmatrix} = -ke_i - \frac{\partial}{\partial t}e_i + \varepsilon_i \tag{2.4}$$

where $\mathbf{J}_i$ denotes the row Jacobian of $e_i$ with respect to the joint and feature variables. If no slack variable is used, the constraint is called a *hard constraint* (higher priority). If a slack variable is used, the constraint is called a *soft constraint* (lower priority). Inequality constraints are handled in the same way, by replacing the equality in (2.3) and (2.4) with an inequality. Furthermore, by replacing the term $-ke_i$ with a desired velocity $v_{\text{des}}$, a velocity constraint is obtained.

Equation (2.4) is used to construct the linear constraint (2.1b). Concretely, the row Jacobian $\mathbf{J}_i$ is used to fill out a row of the constraint matrix $\mathbf{A}$ such that each row $\mathbf{A}_i$ is:

$$\mathbf{A}_i = \begin{bmatrix} \mathbf{J}_i & \mathbf{I}_i^\top \end{bmatrix} \tag{2.5}$$

where $\mathbf{I}_i$ is a vector whose $i$-th element is 1 and 0 everywhere else. The feedback $-ke_i$ and feedforward $-\partial e_i/\partial t$ terms are used to fill out the lower and upper bound vectors $\mathbf{L}_A$ and $\mathbf{U}_A$, whose elements are equal to each other in case of an equality constraint.

The Hessian matrix $\mathbf{H}$ of (2.1a) is formulated as

$$\mathbf{H} = \begin{bmatrix} \mu\mathbf{W}_q & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mu\mathbf{W}_f & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mu\mathbf{I} + \mathbf{W}_\varepsilon \end{bmatrix} \tag{2.6}$$

where $\mathbf{W}_q$, $\mathbf{W}_f$ and $\mathbf{W}_\varepsilon$ are diagonal weight matrices corresponding to the joint velocities, feature variable velocities and the slack variables, respectively. Each diagonal element of $\mathbf{W}_\varepsilon$ corresponds to the weight of a soft constraint (2.4). $\mu$ is a small numerical value introduced to ensure positive definiteness.

The standard implementation of eTaSL as described in [3] uses a numerical solver qpOASES [35] to compute the QP problem solution. The paper limits the number of priority levels of a constraint into two levels. The higher priority level is for hard constraints that are primarily reserved for safety purposes such as the robot joint limits. For such constraints, the slack variables $\varepsilon_i$ in (2.4) is left out. The lower priority level, i.e., soft constraints, can be used for other, non-safety purposes. For this type of constraints, weights can be used to set the relative importance between the constraints.

## 2.3 Extending eTaSL solver to support arbitrary number of priority levels

This section presents an extension to eTaSL solver in order to facilitate constraints with arbitratry level of priorities. This extension is motivated by robotic applications that involve multiple conflicting, non-safety tasks to be performed simultaneously with different levels of importance. In such a scenario, tuning the proper weights for the tasks can be inconvenient. For instance, for some tasks, it is desirable to have the task error to be as small as possible in order to achieve a satisfactory execution, such as in trajectory tracking or positioning tasks. Here, other less "important" tasks should ideally not compromise the performance of the aforementioned tasks, i.e., a hierarchy between tasks is desirable. But safety related tasks, such as workspace limits, should always be enforced, even when they compromise other tasks. Therefore, it becomes important to be able to specify a hierarchy between tasks.

The standard solver of eTaSL only supports two priority levels: higher priority level for the safety tasks and lower priority level for non-safety tasks. In the standard solver, the relative importance of non-safety tasks are set with weights. If the weights are not properly tuned, the solver may achieve numerical optimum, but does not obtain the desired behavior of the tasks. A typical solution to this problem is to set the weight ratios to be sufficiently large, such that they approximate strict priority levels. While this strategy works for small degrees of hierarchy, it quickly becomes numerically impractical for a larger degrees of hierarchy. Especially, this problem occurs in applications that involve multiple robots executing tasks concurrently with possibility of conflicts, e.g., due to a

shared workspace. This motivates the implementation of a solver that facilitates arbitrary level of priorities as an added value for eTaSL.

### 2.3.1 Related work

In the robotics literature, one of the earliest work to address hierarchical tasks is by Siciliano and Slotine [118]. The authors use the null-space projection method to facilitate arbitrary levels of task priorities. However, the framework does not support tasks with inequality constraints. Since then, more general frameworks that treat inequality constraints have been proposed [76, 33, 53]. Escande et al. and Kanoun et al. formulate their frameworks based on *hierarchical quadratic programming* (HQP) [33, 53]. The theoretical foundation to solve HQP problem was already studied in the 1970s, resulting in an approach known as *lexicographic optimization* where a *quadratic programming* (QP) solution is solved at every sequential optimization stage [12]. More recent work by Escande et al. [34] integrates the task function approach of Samson [107] and improves the computation time with respect to the previous work. Some authors also propose to formulate the hierarchical optimization based on *linear programing* (LP) as an alternative to QP. For example, in [109], a formulation to efficiently solve hierarchical $\ell$-1 norm minimization using exact penalty functions is presented.

### 2.3.2 Formulation of eTaSL solver with HQP

An extension to eTaSL solver using the HQP formulation inspired by [34] is as follows. Consider two different tasks to perform simultaneously, namely $\mathcal{T}_1$ and $\mathcal{T}_2$, where $\mathcal{T}_1$ has a higher priority over $\mathcal{T}_2$. Both tasks are assumed to be non-safety tasks, therefore they are not composed of hard constraints. The HQP solver computes the solution in multiple stages, starting from the task with the highest to the lowest priority level. Here, we shall use the convention where a higher priority level is represented with a lower integer number. Thus, the solution to task $\mathcal{T}_1$ is found by solving the QP problem below

$$\underset{x_1}{\text{minimize}} \quad x_1^\top \mathbf{H}_1 x_1 \tag{2.7a}$$

$$\text{subject to} \quad \mathbf{L}_{A_1} \leq \mathbf{A}_1 x_1 \leq \mathbf{U}_{A_1}, \tag{2.7b}$$

$$\mathbf{L}_1 \leq x_1 \leq \mathbf{U}_1 \tag{2.7c}$$

where $x_i = [\dot{q}^\top \dot{\chi}_f^\top \varepsilon_i^\top]^\top$ is the optimization variables for the $i$-th task, $\mathcal{T}_i$. Solving this QP problem gives us the optimal slack variables for $\mathcal{T}_1$, i.e., $\varepsilon_1^*$. Next, we construct an auxiliary vector $x_1^*$ that contains the variables joint and feature velocities together with the now constant solution $\varepsilon_1^*$:

$$x_1^* = [\dot{q}^\top \dot{\chi}_f^\top \varepsilon_1^{*\top}]^\top \tag{2.8}$$

Subsequently, the solution to $\mathcal{T}_2$ is computed by solving a similar, but augmented QP problem:

$$\underset{x_2}{\text{minimize}} \quad x_2^\top \mathbf{H}_2 x_2 \tag{2.9a}$$

$$\text{subject to} \quad \mathbf{L}_{A_2} \leq \mathbf{A}_2 x_2 \leq \mathbf{U}_{A_2}, \tag{2.9b}$$

$$\mathbf{L}_2 \leq x_2 \leq \mathbf{U}_2, \tag{2.9c}$$

$$\mathbf{L}_{A_1} \leq \mathbf{A}_1 x_1^* \leq \mathbf{U}_{A_1} \tag{2.9d}$$

Note that, in addition to its own row constraints (2.9b) and (2.9c), the QP of $\mathcal{T}_2$ also contains row constraints of $\mathcal{T}_1$ (2.9d) , but taking into account the optimal slack variables obtained from the preceding QP solution. The optimal variables $\dot{\chi}_{f_1}^*$ and $\varepsilon_1^*$ appear as constants in the $\mathcal{T}_2$'s QP problem, leaving only the joint variable velocities $\dot{q}$ to optimize in the constraint equation (2.9d). Consequently, the solution to $\mathcal{T}_2$ is computed without affecting $\mathcal{T}_1$, i.e., the solution to $\mathcal{T}_2$ lies in the null space of $\mathcal{T}_1$.

For a more general scenario with $n_t$ number of tasks, each with different priority level, then the HQP formulation for the $i$-th priority level, where $1 \leq i \leq n_t$ is as follows.

$$\underset{x_i}{\text{minimize}} \quad x_i^\top \mathbf{H}_i x_i \tag{2.10a}$$

$$\text{subject to} \quad \mathbf{L}_{A_i} \leq \mathbf{A}_i x_i \leq \mathbf{U}_{A_i}, \tag{2.10b}$$

$$\mathbf{L}_i \leq x_i \leq \mathbf{U}_i, \tag{2.10c}$$

$$\mathbf{L}_{A_j} \leq \mathbf{A}_j x_j^* \leq \mathbf{U}_{A_j}, \forall j \in 1, \ldots, i-1 \tag{2.10d}$$

Note that the lower the priority level, the more additional constraint equations that have to be taken into account. For a task in $i$-th priority level, there are $i-1$ additional row constraints that have to be included.

One advantage of eTaSL's well-separated task specification and execution is that it eases the implementation of additional solvers without modifying eTaSL

core source code. This way, programmers can select the most appropriate solver for the specified task depending on the application requirements. From the technological perspective, this is a different approach compared to other frameworks such as SoT, where a dedicated solver was developed specific for robot control usage. In contrast, eTaSL, and thus this thesis, leverages the maturity of off-the-shelf, general purpose numerical solvers developed by the optimization community. Consequently, aspects such as numerical reliability and treatment of inequality constraints are delegated to the off-the-shelf solvers, rather then dedicated methods as in [34]. In this thesis, the HQP solver is implemented using OSQP, a numerical solver based on operator splitting technique [124].

### 2.3.3 An example of the application of the HQP solver in resolving conflicting, concurrent tasks

#### Scenario

To demonstrate the use of priorities in a practical application, and compare the priorities with the use of weights in a numerical example, we describe the following application: a dual-arm robot system performs concurrent tasks as illustrated in Figure 2.1. The two robot arms perform similar tasks, namely approaching parts in order to grasp them. These parts are placed on a rack such that the arms can not pick them simultaneously because they might collide into each other. The arm has completed the approach motion and is committed to grasping when the gripper is close enough to the part, i.e., when the Euclidean distance between the gripper and part frames is lower than or equal to a predefined threshold. After the approach phase, the arm performs the next tasks, i.e., closing the gripper and lifting the part.

The simulation shows how the HQP solver can resolve conflicting tasks by assigning strict priority levels. The desired overall behaviors are as follows: *i)* each arm should try to reach a part by following a desired path *ii)* when the arms are about to collide, the first arm should be given a preference such that it keeps on progressing while the second arm moves away to avoid collision *iii)* the arm must keep its end-effector on the desired paths as best as possible during advancing and retracting motion. The main motivation for the second behavior is to avoid being trapped in a local optimum without any arm being able to reach the parts. Meanwhile, a practical motivation for the last behavior is to achieve a predictable deviating motion when the arm is reacting to disturbances.

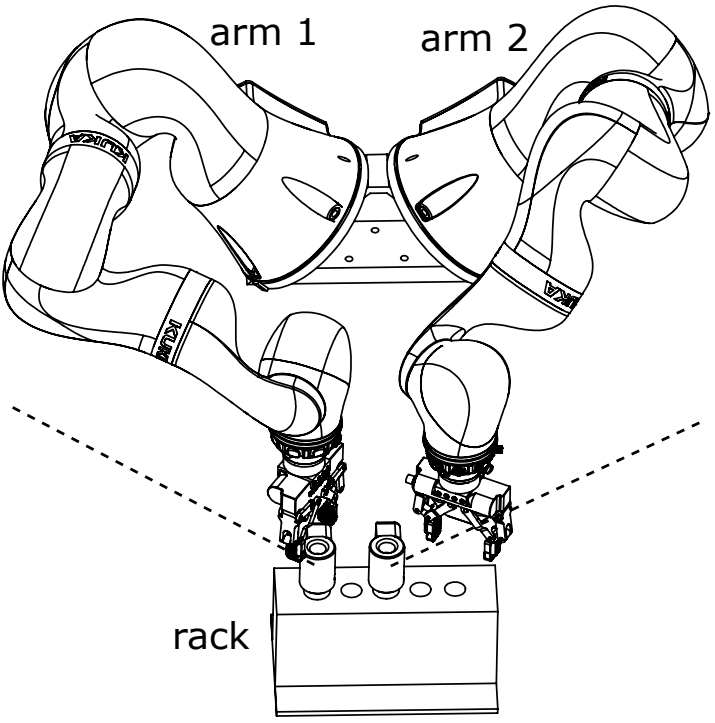



Figure 2.1: A dual-arm setup for part grasping scenario to illustrate the applicability of the implemented HQP solver. Each arm is trying to reach a part in order to pick and assemble later. The parts are placed in a rack in such a way that they cannot be picked by both arms simultaneously. The desired paths to approach the parts are depicted with dashed lines.

### Methodology

We model the above application in eTaSL as follows. Each desired path is formulated as a target frame expression which is parameterized with a progress variable. This progress variable measures how much the robot has progressed along the desired path. It is implemented with eTaSL's feature variable and is constrained to follow a specified motion profile. This way, eTaSL can optimize the progress variable as an auxiliary DOF, enabling the robot to move reactively along the path.

We divide the overall behavior into five tasks with different levels of importance. At the highest level is a collision avoidance task that should never be violated in

Table 2.1: The list of tasks performed by the dual-arm setup in Figure 2.1. The HQP solver uses strict priority levels while the standard eTaSL solver uses weights. The "n/a" signifies no weights due to the task implementation as hard constraints.

| Task | HQP solver | | Standard solver | | | |
|---|---|---|---|---|---|---|
| | | | stay on path | | endgoal | |
| | priority | weight | priority | weight | priority | weight |
| $\mathcal{T}_0$ | 0 | n/a | 0 | n/a | 0 | n/a |
| $\mathcal{T}_1$ | 1 | 1 | 1 | $10^4$ | 1 | $10^4$ |
| $\mathcal{T}_2$ | 2 | 1 | 1 | $0.5 \times 10^4$ | 1 | 100 |
| $\mathcal{T}_3$ | 3 | 1 | 1 | 1 | 1 | 1 |
| $\mathcal{T}_4$ | 4 | 1 | 1 | 0.01 | 1 | 0.01 |

any circumstance ($\mathcal{T}_0$). This task is specified by imposing inequality constraints on the distances between the two robots' links for which collisions may occur. These distances should not be lower than the predefined thresholds. At one level below, the task is to keep the first arm's end-effector on its desired path ($\mathcal{T}_1$). This task is specified by introducing an equality constraint to the end-effector frame to track the target frame. A similar task is specified for the second arm at the subsequent level. The next task is the actual approach motion along the desired path for the first arm. This task is specified by imposing an equality constraint to the progress variable to follow a predefined motion profile ($\mathcal{T}_3$). Similarly, at the lowest level, this task is specified for the second arm ($\mathcal{T}_4$).

In accordance with these levels of importance, we implement the above behavior using the standard and HQP solvers. Using the standard solver, task $\mathcal{T}_1$ is placed at the top priority level, i.e., implemented with hard constraints while the remaining tasks are placed at the second priority level, i.e., implemented with soft constraints. We further assign different weights to the tasks at the second priority level. These weights are tuned to approximate strict hierarchy such that the tasks with lower importance do not significantly influence the tasks with higher importance. The standard solver is tested with two different weight settings: one setting favors both robots to stay on the paths while the other favors the prioritized robot to reach the end goal. For the HQP solver, each of the five tasks is placed at a distinct priority level. The most important task is placed at priority level 0 and the least important task is placed at priority level 4. Table 2.1 lists the corresponding priority level and weight of each task.

### Simulation results

The results of the task execution using both the standard and HQP solvers are presented in Figure 2.2. In the first and second columns, plots from the standard solver are presented. The standard solver can emulate hierarchical tasks, however, with noticable degradation in the task execution qualities. The first column shows the results for the weight setting that favors both arms to stay on the paths. This is done by setting the weight of $\mathcal{T}_2$ close to that of $\mathcal{T}_1$. However, this sacrifices the final errors as both arms achieve steady states where the smallest distance to the part is around 5 mm. For tasks that require highly accurate positioning, such as grasping of small parts, this performance may be inadequate. In contrast, the second column shows the results for the weight setting that favors the prioritized arm to reach the end goal. This is done by setting the weight of $\mathcal{T}_1$ much higher than $\mathcal{T}_2$. As a result, the position error for the first arm is reduced to around 1 mm since now the second arm retracts further. However, this causes more deviation from the desired paths, especially for the second robot, as evident from the bottom right plot.

The results of the HQP solver are shown in the third column. For the HQP solvers, all tasks except the collision avoidance have weight of 1. As evident from the plots, the HQP solver satisfies the desirable behaviors described previously. When conflict occurs, collision is avoided by allowing the first arm to proceed while retracting the second arm along the desired path. The top left subfigure shows the progress variable of the less prioritized arm decreases, causing the arm to retract. Thanks to the strict hierarchy in the tasks, the target can be reached without causing large deviation from the desired paths.

### Closing remarks

The simulation shows the benefit of using a hierarchical solver that supports arbitrary level of priority for complex scenarios involving conflicting tasks. The implemented HQP solver can ensure execution qualities of certain tasks do not get influenced by tasks below them. Furthermore, the extended solver could help ease efforts to tune proper weights for conflicting tasks. While the standard eTaSL solver can emulate strict priorities to a certain extent, determining the appropriate weights can be more challenging as the level of hierarchy increases. As seen from Table 2.1, approximating priorities using weights often requires setting the relative difference between weights significantly large (a factor of 100 in this particular example). This can lead to a numerical problem due to the limited numerical accuracies of off-the-shelf solvers. The extended solver could alleviate these problems by facilitating arbitrary levels of priorities, while

retaining the use of weights to balance tasks in the same priority level. The software implementation of the solver is publicly available in [144, 145].

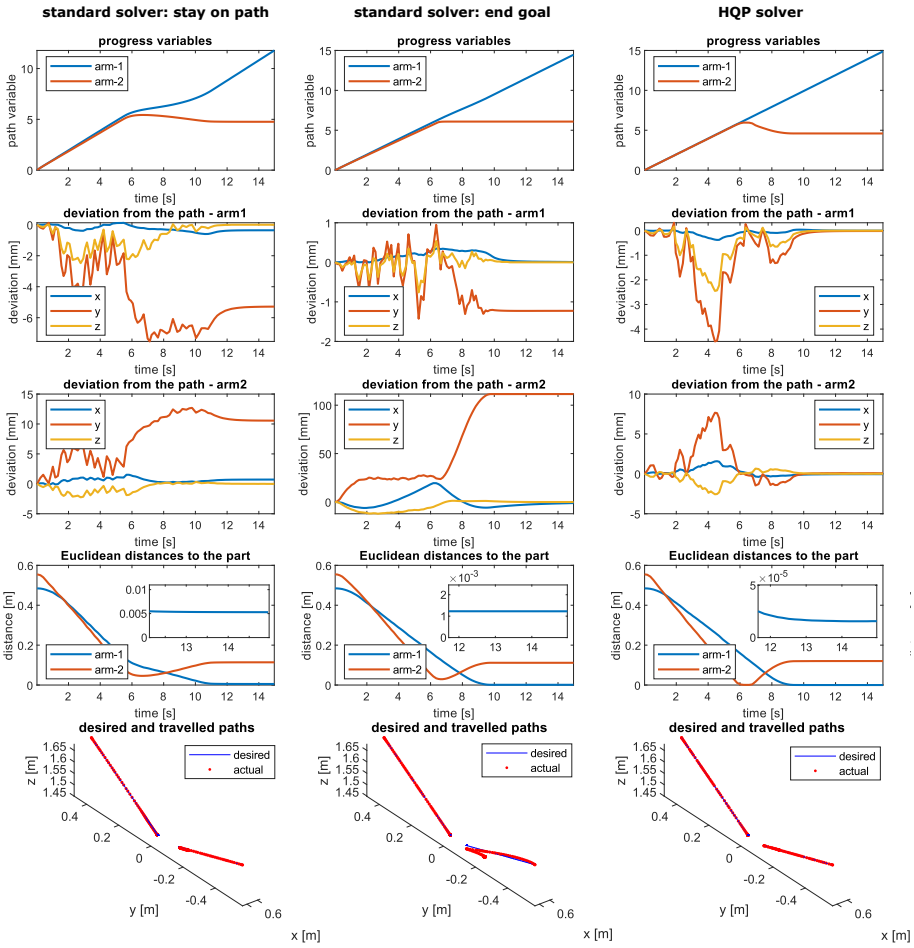



Figure 2.2: Result of the tasks execution with different solver settings. First column: standard solver with a weight setting that favors both robots to stay on the paths, second column: standard solver with a weight setting that favors the prioritized arm to reach the end goal, third column: HQP solver. The first row shows the progress variables that parameterize the desired paths. The second and third row show the deviation plots for the first and second arm, respectively. These two rows represent the errors of staying on the desired paths, i.e., $\mathcal{T}_1$ and $\mathcal{T}_2$. The fourth row shows the Euclidean distances to the target position, with inset plots that magnify the converged distance for the prioritized arm. This row represents the errors of trying to achieve the end goals, i.e., $\mathcal{T}_3$ and $\mathcal{T}_4$. Finally, the final row shows the desired and traveled paths of the two arms. The right trajectories correspond to the first arm and the right trajectories correspond to the second arm.

# Chapter 3

# Skill-based Programming Framework for Composable Reactive Robot Behaviors*

## Abstract

This chapter introduces a constraint-based skill framework for programming robot applications. Existing skill frameworks allow application developers to reuse skills and compose them sequentially or concurrently. However, they typically assume that the skills are running independently and in a nominal condition. This limitation hinders their applications for more involved and realistic scenarios, e.g., when the skills need to run synchronously and in the presence of disturbances. This chapter addresses this problem in two steps. First, we revisit how constraint-based skills are modeled. We classify different skill types based on how their progress can be evaluated over time. Our skill model separates the constraints that impose task consistency and the constraints that make the skills progress, i.e., reaching their end conditions. Second, this chapter introduces composition patterns that couple skills concurrently such that they are executed in a synchronized manner and reactive to disturbances.

*This chapter is based on: Pane, Y., Aertbeliën, E., De Schutter, J., and Decré, W. Skill-based Programming Framework for Composable Reactive Robot Behaviors. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 7087-7094, Las Vegas, USA (2020). Adaptations have been made in order to maintain consistent terminologies and notations throughout the thesis as well as to provide examples of skill compositions. An accompanying video is available online at: https://youtu.be/U1gkm5lrz2Y

The effectiveness of our framework is evaluated on a dual-arm robotics setup that performs an industrial assembly task in the presence of disturbances.

## 3.1 Introduction

The demand for mass customization has forced manufacturing processes to be more agile than ever before. As a consequence, robotic systems, as one of the key enabling technologies, need to be more configurable and adaptable to a wide range of situations. Furthermore, since many aspects from the environment cannot be modeled accurately, in many cases robots also need to be **reactive** with respect to unforeseen disturbances. That is, utilizing online information, e.g., sensor measurements to adapt their trajectory in order not to violate constraints such as collision avoidance.

Programming a complex reactive behavior requires a different approach than the traditional methods of simply specifying point-to-point (PTP) or impedance motion. This is especially important because robot programming is often performed by *application developers* who typically have less expertise in robotics. They also need to develop applications in a shorter time, thus programming with simple motion primitives is too laborious and undesirable.

One of the solutions proposed in the literature is skill-based robot programming [95] [131] [127] [90]. Skills give robots the capability to perform a certain behavior, e.g., trajectory following or force-based insertion. Skills are implemented by encapsulating the underlying motion controller and exposing a set of configurable parameters to the users. This way, application developers can easily reuse the skills without having to deal with writing the reactive behavior themselves.

In this chapter's context, we define the term **composability** as the ability for application developers to compose existing skills in order to yield a predictable combined and desired behavior. One of the main limitation of the current skill-based programming frameworks is their limited composability. This in turn reduces the reusability of existing skills. This problem is partly due to the limitation of the underlying skill specification and the relatively rudimentary composition patterns, i.e., the possible ways of composing skills. For instance, existing frameworks allow application developers to specify skills[†] and compose them to be executed concurrently or sequentially. But they do not facilitate to specify more complex composition such as imposing synchronization constraints

[†]Throughout this thesis, the notion of specifying a skill means specifying the corresponding task to be performed. In the context of constraint-based skills, this is equal to specifying the task to perform in terms of constraints.

between concurrent skills or composing skills to handle non-nominal situations such as collision or human-induced disturbances.

In this chapter, we present a new skill programming framework based on a **constraint-based approach**. Our framework differs from existing solutions because it not only allows the programmers to change the skill parameters, but also allows them to apply and specify composition constraints to create coupled behavior between the skills. These composition constraints are presented as patterns; they are semantically well-defined, thus can be understood and reused by application developers. The contributions of this work are: *i)* a constraint-based skill model that separates the *task* and *progress* constraints, *ii)* reusable composition patterns for composing skill behaviors in non-trivial manners, *iii)* evaluation of the proposed framework in the context of dual arm robotic assembly task.

The remainder of the chapter is organized as follows. In Section 3.2, we present an overview of the previous work. Section 3.3 describes how we model constraint-based skills. The different composition patterns are presented in detail in Section 3.4. We evaluate the framework in an industrial assembly task and report the results in Section 3.5. Section 3.6 concludes the chapter.

## 3.2 Related Work and Motivation

Methods to generate real-time, reactive robot motion while executing a nominal task constitute an extensive research area within the robotics community. A variety of techniques have been proposed to address different applications. In the context of collision avoidance behavior, approaches based on potential function are widely-used [57]. Efforts to combine local and global planning have resulted in mixed methods, such as elastic strips [18]. In [18], it is possible to compose a nominal behavior with non-nominal ones such as following a path while avoiding obstacles and maintaining a posture by assigning priority levels.

Meanwhile, in the context of reacting to a contact situation, constraint-based techniques such as the task frame formalism (TFF) are more often used [19]. This formalism allows for specifying force/torque-controlled tasks along and around independent task frame axes. More recently, there has been significant progress in constraint-based programming, resulting in, among others, the iTaSC [113] [120], eTaSL [3], and stack-of-tasks (SoT) frameworks [77].

The iTaSC framework can explicitly model geometric uncertainty to improve the robustness of the task execution with respect to disturbances. It also proposes the use of feature coordinates to enable more general constraint

expressions. However, with iTaSC, the programmer is forced to formulate a six DoF virtual kinematic chain (VKC). This restriction is not necessary in eTaSL, thus the singularity problem inherent in the VKC can be avoided. In eTaSL, the separation of concerns between the task specification and the solver is properly addressed, such that the same specification can be executed with different types of solvers. The SoT is an alternative framework that differs from eTaSL and iTaSC in the way the optimization problem is solved. In particular, a null-space projection technique is used such that lower priority constraints will not affect the higher priority ones.

One main advantage of the constraint-based approach is the composability of constraints. Programmers can develop robot skills by composing constraints instead of composing trajectories. When the constraints are conflicting, different weights or priority levels can be assigned. Another advantage is that, in most cases, the skills can be executed in real-time, and therefore are suitable for industrial robot tasks.

Due to its versatility, the constraint-based approach is adopted as the underlying method in developing skill-based programming frameworks. For example, in [131], Thomas et al. propose LightRocks, a domain specific language (DSL) for programming sensor-based skills. The skill is specified based on an extended TFF. The programmers can compose skills using the UML/P state chart model.

Another example is the framework developed by Nägele et al. [90] [44] where the skill specification is based on iTaSC. It facilitates to instantiate new skills from already existing instances by following the prototype pattern. More complex skills can be composed hierarchically, both in a sequential and concurrent manner.

Other frameworks achieve flexibility using path planning or ad hoc motion generation instead of a constraint-based reactive approach. These approaches can achieve a considerable amount of flexibility but with a slower reactivity. An additional drawback is a reduced composability between skills since they do not share a common underlying specification. One example is proposed by Pedersen et al. [95], who showed that skills can either be executed using a TFF-based controller or by an off-the-shelf motion planner [96]. The framework allows non-experts to compose skills sequentially, forming a task. A transition occurs when the post-conditions of the source skill and the pre-conditions of the target skill hold. Another example is from Stenmark et al. [127] [125]. Their work does not particularly focus on composability of skills, but rather on how to parameterize the skills in an intuitive way. They also evaluated their prototype tool for programming dual-arm tasks by non-robot experts [125].

The above mentioned frameworks reduce the effort in skill programming

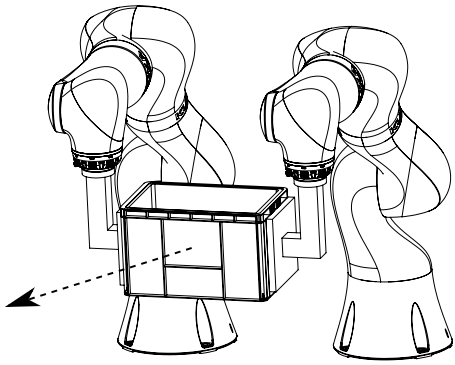


Figure 3.1: Cooperative object transfer with dual arm setup

by abstracting away the low-level coding. However, the relatively simple composition rules limit the level of complexity that can be achieved for the robot behavior. In our approach, composing two skills does not simply mean merging their constraints, but also introducing additional constraints that describe the coupled behavior between the composed skills. This is particularly relevant when the execution of skills is subject to non-nominal conditions or disturbances. For example, consider the problem of transferring a box with two robots as illustrated in Figure 3.1. Each robot executes a sequence of skills to approach the box and bring it to a target pose. In the nominal situation, the two skills can be executed independently at the same time. However, in a non-nominal situations such as in the presence of human-induced disturbance, each robot should adapt its skill's progress to maintain a proper grip. Here, the simple composition strategy will fail. Alternatively, one may argue that such a problem can be solved by a single skill that controls both robots, forming a closed kinematic chain with the object. However, this implies the need to create a skill that is application-specific, hence not so reusable. The perspective of this research is to provide application developers with a set of generic skills, while facilitating the creation of more complex skills by means of compositions.

## 3.3 Constraint-Based Skill Modelling

This section presents the underlying model that describes how a constraint-based skill is specified and structured. The model serves as a guideline for *skill developers*, i.e., robotics experts who implements skills. The developed skills can then be configured and reused by the application developers who are the

domain-experts. This way, the application developers can focus on defining the tasks at hand and selecting the suitable skills. In this work, we adopt eTaSL to specify the skill's constraints. This choice is due to its strong separation of task specification and execution and its relatively mature development. Furthermore, one of the advantages of eTaSL is the support of feature variables. These are auxiliary variables that are used to specify degrees of freedom in the skill. Such variables are useful not only for specifying how a skill progresses, but also how it deviates from the nominal path. We explain this in more detail below.

### 3.3.1 Constraint and Monitor

A skill is mainly composed of a set of *constraints* and *monitors*. The constraints and monitors are acting upon *expressions*. The expressions can depend on *input* signals such as sensor values. These expressions can also be specified as *outputs*. The skill developers typically spend most of their effort in specifying and testing these two components. Ideally, the developed skill should represent a well-defined reactive robot behavior, specifying not only the nominal behavior but also how to deviate in case of disturbances. This way, it is easier for the application developers to predict the resulting motion and select suitable skills for their use cases.

In eTaSL, a constraint is imposed on a task error function $e$:

$$e = f(q, \chi_f, t) \tag{3.1}$$

where $q \in \mathbb{R}^{n_r}$ denotes the $n_r$ joint variables , $\chi_f \in \mathbb{R}^{n_f}$ denotes the $n_f$ feature variables and $t \in \mathbb{R}$ is time. Skill developers can specify a constraint by imposing target values (equality) or bounds (inequality) on the expresion either in position or velocity level.

In a more formal notation, a constraint $c \in \mathbb{C}$ can be represented as a tuple $c := \langle e, \kappa \rangle$ where $e$ is the task error expression and $\kappa \in \mathbb{K}$ is the assigned controller. In eTaSL, the controller steers $e$ such that it converges with the following dynamics:

$$\dot{e} = -ke \tag{3.2}$$

The control gain $k$ can be set to obtain a desirable time constant. This error dynamics consist of a feedback term $-ke$; a feedforward term $\partial e/\partial t$ exists when the task error $e$ explicitly depends on time $t$.

Monitors continuously evaluate an eTaSL expression and trigger events once certain conditions are met, e.g. when the pose error between is lower than a given threshold. They are useful for end conditions of skills, such that transition to the subsequent skills can be performed.

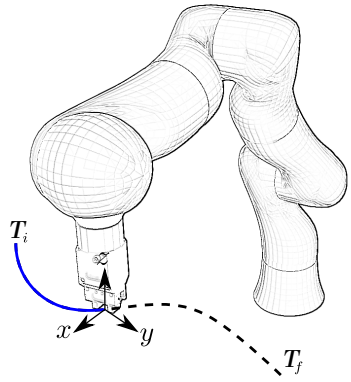



Figure 3.2: A Cartesian motion skill that moves a robot end-effector to a target pose while following a given path (dashed line). The blue solid line signifies the path already travelled.

From the point of view of the application developers, skills are simply configurable and composable building blocks. The detailed implementation of the skill-specific constraints and monitors are abstracted away from them. Instead, they are exposed to the constraints and monitors' parameters, which can be adapted to suit the target application.

### 3.3.2 Specifying Skill Behavior with Constraints

Constraints are the main component of a skill as they determine the robot's overall behavior. This work makes a distinction between skill constraints that are either *task* or *progress* constraints. Progress constraints result in behaviors that advance the skill towards completion, i.e., it drives the skill from its initial to goal condition. Meanwhile, the task constraints are the remainder of a skill's constraints. Task constraints typically result in behaviors of that that keep the skills on the task space, i.e., keep the execution of the skill within acceptable boundaries. In other words, task constraints describe conditions that have to hold continuously throughout the task. Together, these two constraint classes result in a robot that performs the skill-specific behavior while trying to get closer to its goal state. As an illustration, consider a path following skill as depicted in Figure 3.2. The skill moves the robot end-effector from initial pose $T_i$ to a target pose $T_f$. Here, the task constraints will keep the robot end-effector on the path. At the same time, the progress constraint will enforce the end effector to move along the path by following a given progress profile. This

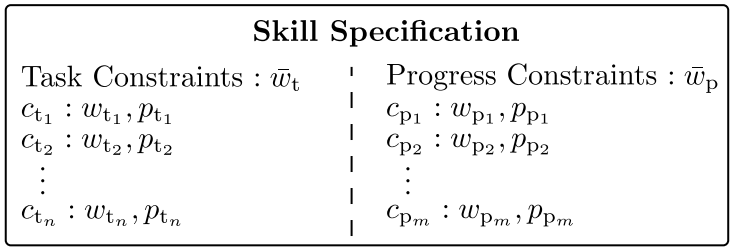



Figure 3.3: The structure of constraints in a skill specification

separation of constraint types in a skill specification is in contrast to previous works that treat all task constraints uniformly. Such a distinction is beneficial since it introduces a degree-of-freedom within the task space. The skill can optimize this degree-of-freedom to resolve non-nominal situation such as the presence of disturbances or conflicting objectives. Furthermore, it also allows us to compose skills by synchronizing their progresses, as will be shown in the subsequent section.

The task and progress constraints can have different weights and priorities. If the task constraints' weight is larger than the progress constraints' weight, then the robot prefers to remain in the task space and sacrifices its progress whenever a disturbance occurs. Conversely, when the progress constraints have larger weight or higher priority, then the robot prefers to stay on track with its nominal progress while deviating from its task space. This way, different deviating behavior can be achieved by varying the two constraints weights or assigning different priority levels to them.

Figure 3.3 outlines the constraints structure in a skill specification. The skill may be composed of $n$ task constraints and $m$ progress constraints. Each task constraint $c_{\mathrm{t}_i} \in \mathbb{C}$ has its own weight $w_{\mathrm{t}_i}$ and priority $p_{\mathrm{t}_i}$. A constraint $c_{\mathrm{t}_i}$ will not violate $c_{\mathrm{t}_j}$ when $p_{\mathrm{t}_i} > p_{\mathrm{t}_j}$, regardless of their weights. The weights difference will only have an effect when the two constraints are in the same priority level. The relative importance between the task and progress constraints can be adjusted by setting $\bar{w}_\mathrm{t}$ and $\bar{w}_\mathrm{p}$. These weight factors will be multiplied with the weight of each task and progress constraint, respectively. This way, the relative importance within the same constraint group remains the same.

### 3.3.3 Modelling Skill Progress

A progress constraint is imposed on the *progress variable*, $s = [0, 1]$, which is a normalized scalar expression that measures the skill's advancement in

continuous time. The skill starts when $s = 0$, while $s = 1$ implies the skill has reached its end goal. We assume that $s$ is an increasing function in the nominal situation, i.e., in the absence of disturbances. As an example, consider again the path-following skill of Figure 3.2. Assuming the robot tracks the path by following a target pose generated by a motion profile, the progress variable is simply $s = \frac{\chi_{f_{pv}}}{\lambda}$ where $\chi_{f_{pv}}$ denotes the coordinate along the path and $\lambda$ denotes the path length. A progress constraint can be specified such that $s$ increases with a constant speed. As the motion profile is a function of the path coordinate $\chi_{f_{pv}}$, task constraints will enforce the robot end-effector to track the generated pose while satisfying the constant progress speed in a nominal situation. In general, the progress variable of a given skill can be mathematically expressed as a function of the joint and feature variables $s = f(q, \chi_f)$.

Although the separation of task and progress constraints is useful to achieve different behaviors, specifying a progress constraint is not applicable to all skills. This is because not every skill has an explicit measure of advancement. For example, skills that solely performs reactive behavior upon disturbance such as collision avoidance or admittance motion do not actually "make progress" as time elapses. For these types of skill, progress constraints do not exist.

### 3.3.4 Classifying a Skill Based on its Progress

As different skills produce different behaviors, how their progresses are defined may be different. Based on the property of their progress variables, we analyze and classify different types of skills. Such classification is needed in determining whether two skills can be composed together using the patterns proposed in Section 3.4.

The different skill types along with their examples are summarized in Table 3.1. The left column (Type-I) describes the first category of skills whose progress can be expressed as a function of the optimization variables $q$ and $\chi_f$. We say that their $s$ values are instantaneously *observable* and *controllable*. The second group (Type-II) are skills that, in nominal condition, have increasing progresses, but the values are neither observable nor controllable. These skills typically finish upon detecting certain events. An example is guarded motion, i.e., a skill that approaches an object with unknown location and stops as soon as contact is made. As the skill runs, it becomes closer to the object, thus the progress increases. But due to the unobservable end state, its value can not be quantified. Finally, the last group (Type-III) belongs to skills that do not have defined progresses, therefore lacking the progress constraints. These skills typically do not produce any motion in a nominal condition, but perform reactive motion in the presence of disturbance. An example is an admittance motion skill where

the robot only moves when force is applied. Since these types of skills do not make progress by themselves, they are commonly composed concurrently with progressing skills.

## 3.4 Composing Behaviors with Composition Patterns

More complex reactive behaviors can be programmed by means of composing skills. This work focuses more on composability in continuous level while assuming that the discrete-level composition can be simply specified with well-known coordination models such as finite-state machine (FSM) or behavior trees.

We consider more involved concurrent behaviors in which the composed skills are not just running independently, but they are governed by coordination schemes, namely *composition patterns*. Each pattern results in a different composition behavior with well-defined semantics such that application developers can reuse them for their target applications.

Formally, a composition pattern $\texttt{CP}$ can be viewed as a function that maps two skills into a new composed skill[‡]:

$$\texttt{CP} : \mathbb{S} \times \mathbb{S} \to \mathbb{S} \tag{3.3}$$

where $\mathbb{S}$ denotes the skill set. Application developers can create increasingly more complex behaviors by composing multiple patterns. For example, given two different composition patterns $\texttt{CP}_1$ and $\texttt{CP}_2$, more complex compositions can be created:

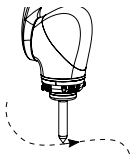
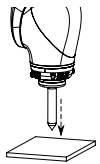
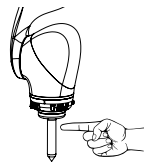
$$\texttt{CP}_1 \circ \texttt{CP}_2 : (\mathbb{S} \times \mathbb{S}) \times \mathbb{S} \to \mathbb{S} \tag{3.4}$$

### 3.4.1 Progress Synchronization Patterns

These patterns are motivated by tasks that require time-domain synchronization of robot behaviors. In such tasks, the active skills not only run simultaneously, but they have to reach their goals in a synchronized manner. In general, these patterns handle such problems by imposing constraints on the skills' progress variables. Here, we propose two synchronization patterns.

[‡]Note that it is not the intention to describe the composition patterns using mathematical operators. E.g. associative and commutative properties do not hold necessarily.

Table 3.1: Different skill classification based on their progress variable properties

| Type-I: Skills with controllable progress | Type-II: Skills with uncontrollable progress | Type-III: Non-progressing skills |
|---|---|---|
|  |  |  |
| • The progress variable is instantaneously observable and controllable<br>• Varying the progress variable moves the skill along its task space<br>• Example: Cartesian motion with motion profile | • The progress variable can not be instantaneously determined and controlled<br>• Reaching the goal condition is only known upon an event detection<br>• Example: guarded motion | • The progress variable can not be defined<br>• Complement the progressing skill to actually achieve a task<br>• Examples: admittance, collision avoidance |

### Leader-Follower Synchronization

This mode is useful when a task requires one skill to follow another skill's progress. The skill that is assigned a "leader" role may progress independently with respect to the "follower" skill. When the leader skill is perturbed, the follower skill has to adjust its progress variable accordingly to maintain a synchronized motion. In eTaSL, the constraint that realizes this pattern is imposed on the follower's progress variable $s_{\text{follower}}$:

$$c = \langle e = s_{\text{follower}} - s_{\text{d}}, \kappa_{\text{proportional}} \rangle \tag{3.5}$$

where $\kappa_{\text{proportional}} \in \mathbb{K}$ is a controller that satisfies (3.2). The desired progress $s_{\text{d}}$ corresponds to the leader's progress value and may explicitly depends on time, i.e., containing a feedforward term.

This pattern can be viewed as a function that is applied to two skills $\mathtt{s}_1, \mathtt{s}_2 \in \mathbb{S}$:

$$\texttt{ComposeLeaderFollower} : \mathtt{s}_1 \times \mathtt{s}_2 \rightarrow \mathtt{s}_3 \tag{3.6}$$

where the leader, $\mathtt{s}_1$, can be either Type-I or Type-II skill while the follower, $\mathtt{s}_2$, is a Type-I skill to ensure controllability. Composing the two skills results in a composite skill, $\mathtt{s}_3$, which inherits the leader skill's type.

### Peers Synchronization

In contrast to the previous mode, where the follower skill follows the leader skill's progress, this mode does not assign a "leader" nor "follower" role. Both skills have to converge together to a common progress state, in such a way that one skill adapts when the other is perturbed. This pattern is realized by imposing a constraint on each skill's progress variable $s_i$:

$$c = \langle e = s_i - s_{\text{peer}}, \kappa_{\text{proportional}} \rangle \tag{3.7}$$

The progress of skill-$i$ evolves such that it follows its own feedforward term and reacts to the difference to its peer's progress value $s_{\text{peer}}$. In our implementation, the $s_{\text{peer}}$ is assumed to only consist of a feedback term. This behavior is similar to the consensus algorithm presented in [59] applied to a simple two agents scenario. The cooperating agents try to converge to a common information state – a progress variable in this case. This pattern operates on two skills:

$$\texttt{ComposePeers} : \mathtt{s}_1 \times \mathtt{s}_2 \rightarrow \mathtt{s}_3 \tag{3.8}$$

where the composed skills $\mathsf{s}_1$ and $\mathsf{s}_2$, and the resulting skill $\mathsf{s}_3$ all belong to Type-I.

### 3.4.2 Weight Scaling Pattern

The last composition pattern is intended for composing a progressing skill with a non-progressing counterpart. This pattern is useful for combining a nominal behavior with reactive behaviors when their constraints are not compatible with each other's. For example, consider the composition of the Cartesian motion skill of Figure 3.2 with an admittance motion skill. We want a combined behavior such that the robot moves its end-effector along the path, but reacts whenever a human-induced force is applied by stopping its progress.

Combining the two skills by simply executing their constraints together does not result in the desired behavior. On one hand, the Cartesian motion skill imposes the following task constraint:

$$c_{\text{cartesian}} = \langle e = T_{\text{ee}}^{\text{world}} T_{\text{path}}^{-1}, \kappa_{\text{proportional}} \rangle \tag{3.9}$$

where $T_{\text{ee}}^{\text{world}}$ is the robot's end-effector pose expression w.r.t. the world frame and $T_{\text{path}}$ is the target pose. On the other hand, the admittance skill imposes:

$$c_{\text{admittance}} = \langle e = K_a \hat{T}_{\text{ee}}^{\text{ee}} - W_{\text{ee}}^{\text{ee}}, \kappa_{\text{proportional}} \rangle \tag{3.10}$$

where $K_a$ is the admittance matrix and $W_{\text{ee}}^{\text{ee}}$ is the measured wrench applied by the human to the robot's end-effector expressed in the end-effector frame. $\hat{T}_{\text{ee}}^{\text{ee}}$ denotes the end-effector pose represented as a vector of Cartesian position and Euler angles. The admittance skill moves the robot end-effector with a scaled velocity proportional to the applied wrench. When the force is absent, the Cartesian motion skill enforces the robot to move along the path while the admittance skill enforces the robot to stop. These conflicting constraints result in an undesirable behavior.

This composition pattern addresses this problem by means of varying the skills' weights. In a nominal condition with no disturbance, the pattern assigns a higher weight to the progressing skill. In contrast, in the presence of disturbance, the reactive, non-progressing skill gets a higher weight. The weights allocation is done by following a smooth function to avoid unwanted jerky motions.

To this end, the weights are scaled with a sigmoid function that changes value according to an activation signal $\zeta \in \mathbb{R}$. In the case of composing a Cartesian motion skill with an admittance skill, the activation signal can be the norm of the measured wrench $\zeta = ||W_{\text{ee}}^{\text{ee}}||$. In this work, we choose a hyperbolic tangent function $w_{\text{scaling}} = \tanh(\eta\zeta)$, where $\eta$ is a scalar parameter to adjust

the sigmoid's steepness. The skills' weight factors are then scaled accordingly, resulting in new weight factors:

$$\begin{bmatrix} \hat{\bar{w}}_{\mathrm{t}}^{\mathsf{s}_1} \\ \hat{\bar{w}}_{\mathrm{p}}^{\mathsf{s}_1} \\ \hdashline \hat{\bar{w}}_{\mathrm{t}}^{\mathsf{s}_2} \end{bmatrix} = \left[\begin{array}{c:c} (1 - w_{\mathrm{scaling}})\mathbf{S} & 0 \\ \hdashline \mathbf{0}_{1\times 2} & w_{\mathrm{scaling}} \end{array}\right] \begin{bmatrix} \bar{w}_{\mathrm{t}}^{\mathsf{s}_1} \\ \bar{w}_{\mathrm{p}}^{\mathsf{s}_1} \\ \hdashline \bar{w}_{\mathrm{t}}^{\mathsf{s}_2} \end{bmatrix} \tag{3.11}$$

The superscripts $\mathsf{s}_1$ and $\mathsf{s}_2$ signify that the weight factors belong to the progressing and non-progressing skills, respectively. $\mathbf{S} \in \mathbb{R}^{2\times 2}$ is a diagonal, selection matrix whose elements can be set to '0' or '1' in order to select whether the scaling is applied to the task or progress constraints. This pattern can be expressed as a function that is applied to two skills:

$$\texttt{ComposeWeightScaling} : \mathsf{s}_1 \times \mathsf{s}_2 \rightarrow \mathsf{s}_3 \tag{3.12}$$

where $\mathsf{s}_1$ can be either Type-I or Type-II skill, while $\mathsf{s}_2$ is a Type-III skill. Applying this composition pattern results in $\mathsf{s}_3$ that inherits $\mathsf{s}_1$'s type.

### 3.4.3 Examples of skill composition

In order to illustrate the applications of the proposed composition patterns, we provide examples below.

#### Example 1: synchronizing two robots with the leader-follower synchronization pattern

This example shows how the synchronization pattern is used to achieve a coordinated tasks execution. The robots are expected to start and finish their tasks at the same time. A practical use case is two robots that perform coordinated pick-and-place tasks using Cartesian motion skills, e.g., as ilustrated in Figure 3.1. Here, the leader-follower composition pattern is used to achieve a composed Cartesian motion skill. The programmer can set a desired tracking speed to the leader skill, while the follower skill will adapt its speed to synchronize its progress with the leader's progress. This approach allows the application developers to program such coordinated motions without explicitly programming synchronized reference trajectories. The underlying constraints that constitute the task specification is listed in Table 3.2.

**Example 2: adding a reactive behavior to a nominal skill with the weight scaling pattern**

In this example, the weight scaling pattern is used in programming a composite skill that consists of a nominal Cartesian skill and admittance motion skill. The resulting composite skill allows the robot to follow the desired trajectory and to react to a disturbance force by deviating from its nominal path. The reference trajectory is expressed as a function of a feature variable that is constrained to follow a trapezoidal motion profile. This deviation behavior can be adjusted to either favor staying on the desired path or maintaining the progress. The task specifications are listed in Table 3.3.

## 3.5 Experimental Validation

This section reports the results of an assembly task performed by skills that were composed and executed with the proposed framework. The goal of the experiment is to show how the framework is applied to specify skills and their compositions in the presence of disturbances.

### 3.5.1 Problem Scenario

We consider an assembly task of an industrial compressor which consists of two rotors as shown in Figure 3.4. The experiment is performed by a dual-arm setup consisting of a KUKA LBR IIWA and a LWR4 robots that are positioned in proximity. The two robots' workspace overlaps such that collisions can occur. Each robot assembles one rotor into the compressor housing and they both execute the task simultaneously in order to minimize idle time. In addition to collisions between the robots, another non-nominal disturbance is the presence of a human exerting force to the robots during the execution time.

### 3.5.2 Implemented skills

#### Cartesian Motion

The Cartesian motion moves a designated end-effector frame from an initial to a target pose. In order to obtain a smooth motion, a trapezoidal motion profile is defined to generate a Cartesian path as a function of a feature variable $T_{\text{path}} = f(\chi_{f_{pv}})$. $T_{\text{path}}$ is equal to the target pose when $\chi_{f_{pv}} = t_{\text{end}}$, where $t_{\text{end}}$

Table 3.2: The task specification of two skills composed with the leader-follower synchronization pattern. The leader and follower skills are denoted with $\mathsf{s}_1$ and $\mathsf{s}_2$, respectively.

| | Specification | Notes |
|---|---|---|
| $\mathsf{s}_1$: definition of expressions | $T_{\text{des}_1}, t_{\text{end}_1} = \text{f}_{\text{mp}}(\chi_{f_{pv_1}}, d_1, v_{\max_1}, a_{\max_1})$ | $T_{\text{des}_1}$ is the desired/target pose, $t_{\text{end}_1}$ is the total duration of the nominal motion profile, $\text{f}_{\text{mp}}$ corresponds to the trapezoidal motion profile function, $d_1$ is the path length, $v_{\max}$ and $a_{\max}$ are the maximum Cartesian velocity and acceleration; |
| | $s_1 = \frac{\chi_{f_{pv_1}}}{t_{\text{end}_1}}$ | $\chi_{f_{pv_1}}$ is the feature variable |
| $\mathsf{s}_1$: constraints | $c_{\text{t}_1} : \langle e = T_{\text{ee}_1} T_{\text{des}_1}^{-1},\ \kappa_{\text{proportional}} \rangle$ | |
| | $c_{\text{p}_1} : \langle e = \chi_{f_{pv_1}} - t,\ \kappa_0 \rangle$ | $\kappa_0$ corresponds to a controller where the proportional gain (see (3.2)) is set to 0. This way, only the velocity of $\chi_{f_{pv_1}}$ is constrained. |
| $\mathsf{s}_2$: definition of expressions | $T_{\text{des}_2}, t_{\text{end}_2} = \text{f}_{\text{mp}}(\chi_{f_{pv_2}}, d_2, v_{\max_2}, a_{\max_2})$ <br> $s_2 = \frac{\chi_{f_{pv_2}}}{t_{\text{end}_2}}$ | similar to the specification of $\mathsf{s}_1$ |
| $\mathsf{s}_2$: constraints | $c_{\text{t}_1} : \langle e = T_{\text{ee}_2} T_{\text{des}_2}^{-1},\ \kappa_{\text{proportional}} \rangle$ <br> $c_{\text{p}_1} : \langle e = \chi_{f_{pv_2}} - t,\ \kappa_0 \rangle$ | similar to the specification of $\mathsf{s}_1$ |
| Composition pattern | $s_d = \text{make_constant}(s_1)$ | Here, we first form $s_d$, a desired value derived from the leader's progress, i.e., with the make_constant function. This function returns a value that instantaneously coincides with $s_1$, i.e., it returns an expression where the partial derivatives of $s_d$ are zero, such that the composition constraint does not influence the leader's progress. This way, only the follower's progress $s_2$ will be optimized. |
| | $c : \langle e = s_2 - s_d,\ \kappa_{\text{proportional}} \rangle$ | |

Table 3.3: The task specification of two skills composed with the weight scaling pattern. The Cartesian motion skill is denoted with $\mathsf{s}_1$, while the admittance motion skill is denoted with $\mathsf{s}_2$.

| | Specification | Notes |
|---|---|---|
| $\mathsf{s}_1$: definition of expressions | $T_{\text{des}}, t_{\text{end}} = \text{f}_{\text{mp}}(\chi_{f_{pv}}, d, \text{v}_{\max}, \text{a}_{\max})$<br>$s = \frac{\chi_{f_{pv}}}{t_{\text{end}}}$ | The expressions are identical to the previous example in Table 3.2 |
| $\mathsf{s}_1$: constraints | $c_{\text{t}_1} : \langle e = T_{ee} T_{\text{des}}^{-1},\ \kappa_{\text{proportional}} \rangle$<br>$c_{\text{p}_1} : \langle e = \chi_{f_{pv}} - t,\ \kappa_0 \rangle$ | The constraints are identical to the previous example in Table 3.2 |
| $\mathsf{s}_2$: definition of expressions | $\zeta = \lvert\lvert W_{\text{ee}}^{\text{ee}} \rvert\rvert$ | the activation signal computed from the measured wrench |
| $\mathsf{s}_2$: constraints | $c_{\text{t}_1} : \langle e = K_a \hat{T}_{\text{ee}}^{\text{ee}} - W_{\text{ee}}^{\text{ee}}, \kappa_{\text{proportional}} \rangle$ | The admittance motion is a non-progressing skill, therefore it only consists of task constraints. |
| Composition pattern | $\left[\hat{w}_{\text{t}_1}^{\mathsf{s}_1} \hat{w}_{\text{p}_1}^{\mathsf{s}_1} \hat{w}_{\text{t}_1}^{\mathsf{s}_2}\right] = \text{f}_{\text{weight_scaling}}(\zeta)$ | The $\text{f}_{\text{weight_scaling}}$ corresponds to the smooth scaling profile provided in (3.11). |

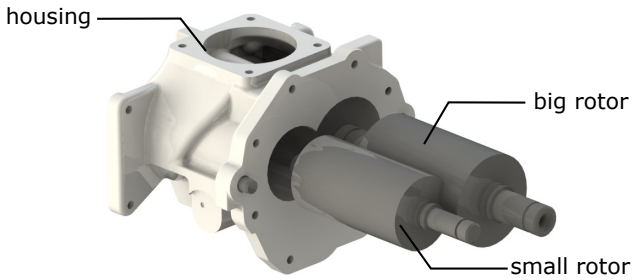



Figure 3.4: The rotary compressor parts to be assembled. Note that the detail of the rotors' helical meshing is not visible.

is the total duration of the motion profile. A progress constraint is imposed to $\chi_{f_{pv}}$ such that its value increases as time elapses. This skill belongs to the Type-I category, with a progress function that can be instantaneously evaluated and controlled.

#### Force-Based Insertion

Due to the tight clearance, the rotors assemblies are difficult to perform using a position-based skill. Therefore, they need to be assembled using a sensor-based skill that continuously checks the contact force. We specify the skill using the task constraints as described in [19]. This skill also belongs to the Type-I category.

#### Collision Avoidance

The collision avoidance skill is specified by imposing inequality constraints to the closest distance between two pairs of shape primitives. These shape primitives approximate the robot links and they are usually modeled with simple geometry, e.g., boxes or capsules to reduce computation cost. Due to its non-progressing behavior, this skill belongs to the Type-III.

#### Admittance Motion

Another Type-III skill that we implemented is a reactive skill that performs admittance behavior by reacting to an external force disturbance. In our experiment, this skill is used to reject the disturbance applied by a human.

### 3.5.3 Composing the Skills

By knowing the expected skill behaviors and their types, the application developers can compose them with the appropriate patterns. We developed the application by first creating a sequential composition of the nominal skills, i.e., the skills that each robot executes without considering disturbances. The sequences are specified with the FSM model proposed in [60]. The second step is to compose the nominal skills with the reactive skills to address the human disturbance. To this end, weight scaling pattern is used for combining the admittance with the Cartesian motion behaviors.

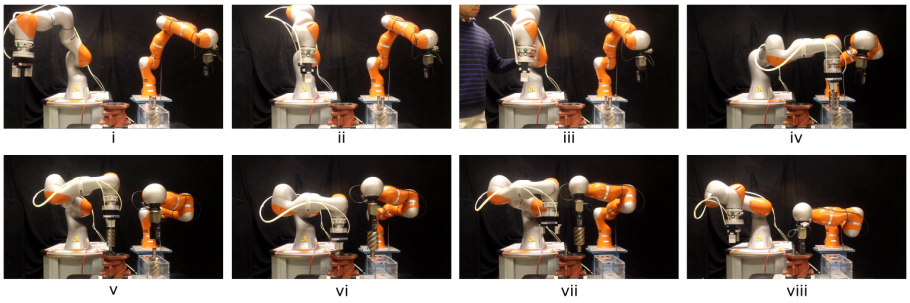



Figure 3.5: Snapshots of the assembly process. Each robot performs a sequence of gripping and inserting the rotor. Human disturbance is acting on the LBR IIWA robot (iii).

The rotors need to be assembled one at a time to avoid collision. Therefore, when the two robots are about to collide, one needs to be prioritized over the other. In eTaSL, this can be realized by assigning different priorities or weights between the two robots' skills. Here, we opted for applying different weights such that the LBR IIWA has more priority than the LWR4 robot. Furthermore, for each progressing skill, larger weights are assigned to the task constraints relative to the progress constraints. This way, the robot stays close to its task space in the presence of disturbances or when the other robot pushes it back. The resulted composed skills are visualized as a simplified state diagram shown in Figure 3.6. The diagram shows the sequential transitions between skills as well as pairs of skills that are subject to composition patterns. In order to demonstrate the synchronization behavior, we impose a leader-follower pattern to the pre-picking task of the LWR4 with the pre-insert task of the LBR IIWA. The experiment shows that the composed skills successfully perform the assembly tasks. Snapshots of the robots performing the tasks are shown in Figure 3.5. A video showing this experiment is available online at: https://youtu.be/U1gkm5Irz2Y.

### 3.5.4 Analyzing the Composition Behaviors

#### Combination of the Cartesian Motion and Admittance Behaviors

The weight scaling pattern is used in combining the nominal Cartesian motion and the admittance behavior during the pre-picking subtask of the small rotor. Here, the pattern uses norm of the measured wrench as the activation signal. This causes a change in the weights between the two skills, as shown in Figure 3.7

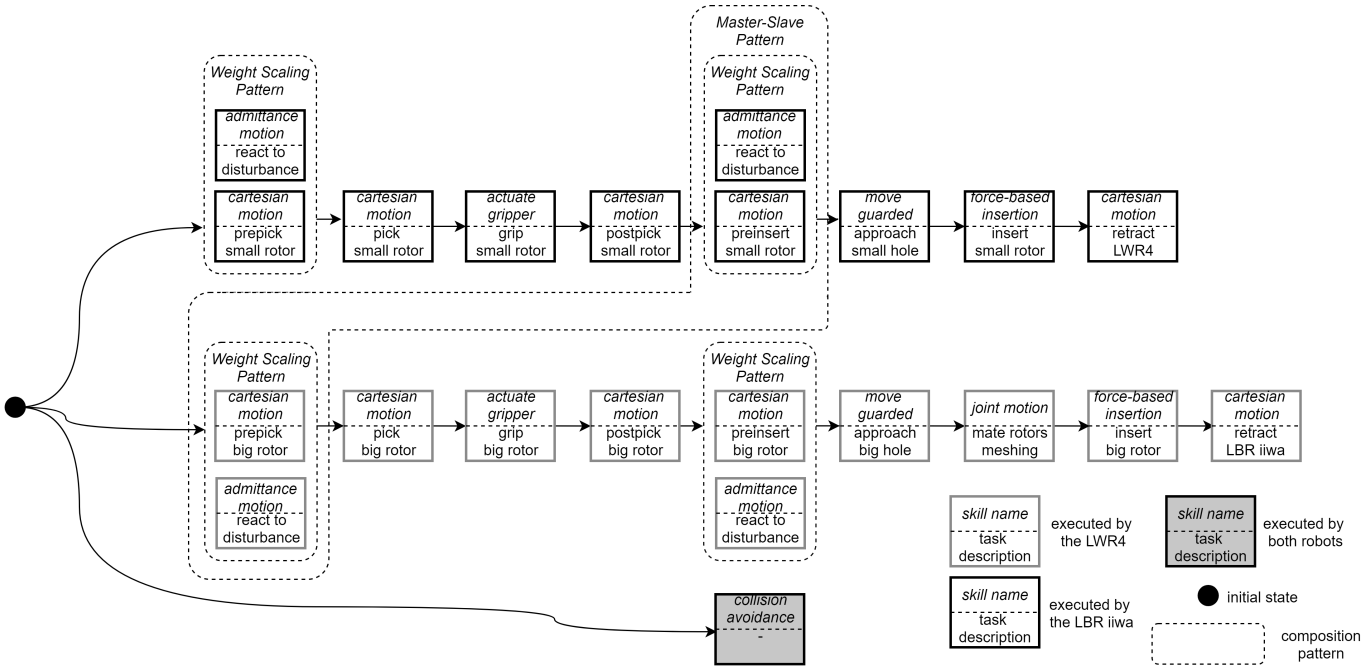



Figure 3.6: The overall sequential and concurrent compositions for the rotors assembly task. Description about each block is provided in the bottom right part of the figure. The arrows denote sequential skill transitions.

Table 3.4: Feature comparison between different skill-based programming frameworks

| Feature | Our framework | Pedersen [95] | Thomas [131] | Stenmark [127] | Nagele [90] |
|---|---|---|---|---|---|
| Skill implementation | eTaSL | TFF, motion planner | TFF | iTaSC, robot primitives | iTaSC |
| Continuous-level composition | concurrent, coupled | n/a | n/a | n/a | concurrent, independent |
| Discrete-level composition | FSM | sequencer | statechart | FSM, sequencer | statechart |
| Constraints resolution | priority, weight | n/a | n/a | n/a | priority |
| Task & progress constraints separation | yes | no | no | no | no |
| Domain-specific language | no | no | yes | no | yes |

(normalized for readability). The disturbance were applied twice: between $t = 3.4$ to 6.2 and $t = 10.7$ to 12.3 seconds. Note that we did not compensate for robot's dynamic force, therefore the measured force values are not zero outside the disturbance period. As the force increases, the Cartesian motion skill's weight reduces while the admittance's weight increases. When the force is large enough, the robot gets pulled from its nominal progress, as shown in the bottom plot of Figure 3.7.

#### Synchronization Behavior of two Cartesian Motion Skills

The pre-insert task of the small rotor is synchronized with the pre-pick task of the big rotor using the leader-follower pattern (Figure 3.5.v). Here, the pre-pick task of LWR4 is assigned the "leader" role while the pre-insert task of LBR IIWA is assigned the "follower" role. Their progress evolution is plotted in Figure 3.8. These two skills started at a different time. The pre-picking task is already active from the beginning of the experiment, but is unable to progress for some time (between t = 13 to 25 seconds) due to the more prioritized LBR IIWA obstructing its path. As soon as it can proceed, the progress increases again and the LBR IIWA makes the transition from post-pick to pre-insert tasks ($t = 25.6$ seconds). Since the pre-insert task is subject to the leader-follower synchronization, it quickly tracks the leader's progress. The plot shows that the synchronization is achieved with small progress error.

## 3.6 Discussion and Conclusion

This chapter proposed a skill-based programming framework that enables application developers to compose reactive behaviors that handle disturbances and synchronization constraints during the task execution. A skill specification that separates the task and progress constraints was modeled. Different skill types were classified based on how their progress is determined. Non-trivial composition of skills can be performed by applying reusable composition patterns.

A comparison between our framework and the state of the arts [95] [131] [127] [90] is outlined in Table 3.4. Our framework's main advantage is the higher composability in continuous level that allows to specify concurrent and coupled behaviors. This composability is either not addressed or not formalized in [95] [131] [127]. Other frameworks are more flexible in the skill implementation. For example, [127] allows to use both iTaSC and robot-specific motion primitives to realize a skill. However, such flexibility also introduces problems when multiple

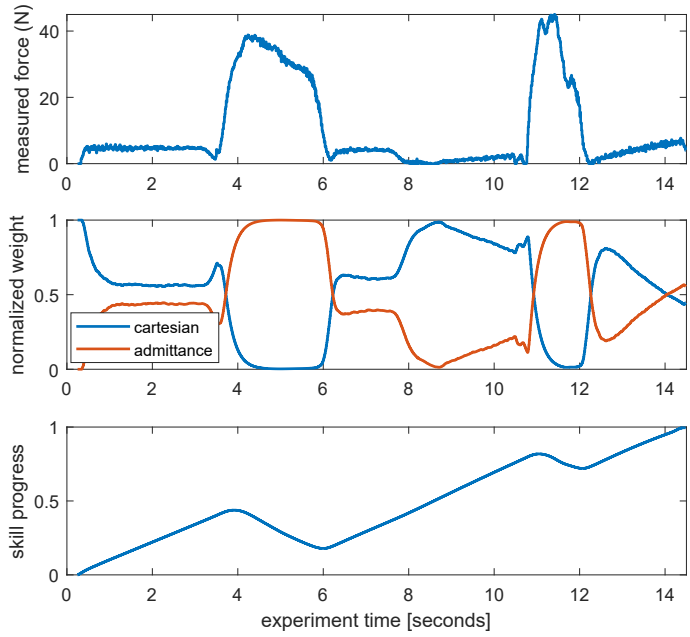



Figure 3.7: The scaled skill weights during pre-picking task of the small rotor. The disturbance force and the skill progress are also shown.

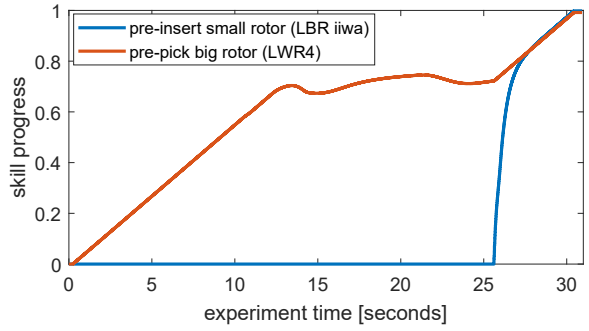



Figure 3.8: The evolution of the skill progresses under leader-follower synchronization pattern.

skills are composed and executed together. When there are conflicts between the skills, it is more difficult to resolve which skills should be prioritized. Due to the stronger ties with the underlying constraint-based task specification framework eTaSL, our work can use both priorities and weights to address this issue. As a topic for future work, we recommend the introduction of a domain-specific language (DSL) to support formal verification, as available in frameworks [131] and [90].

A test case of compressor rotors assembly performed by a dual-arm robot setup was evaluated to verify the effectiveness of the framework. The assembly task was subject to non-nominal conditions including the collision between the two robots, perturbation by a human, and a synchronization constraint between selected skills. Desired deviating behavior was obtained by properly assigning different weights/priorities to the composed skills. The composition patterns facilitated the combination of nominal and reactive skill behaviors as well as synchronization of the skills' progresses. The experiment showed successful execution of the assembly task.

# Chapter 4

# Autonomous Runtime Composition of Sensor-based Skills Using Concurrent Task Planning*

## Abstract

Constraint-based robot programming allows for implementing sensor-based skills that react to disturbances on the one hand, and composable skills that dynamically create and reconfigure complex robot behaviors on the other hand. In this work, we address a class of problems where composing appropriate skills prior to execution is inconvenient due to unpredictable events of disturbances. To this end, we propose an autonomous replanning and acting framework that computes and executes reactive composed skills at runtime. We formulate a

*This chapter is based on: Pane, Y., Mokhtari, V., Aertbeliën, E., De Schutter, J., and Decré, W. Autonomous Runtime Composition of Sensor-based Skills Using Concurrent Task Planning. In *IEEE Robotics and Automation Letters* (2021). Yudha Pane initiated the research and closely collaborated with Vahid Mokhtari in developing the methodology and defining the experimental scenario. Vahid implemented the task planner software and wrote the task planning specifications in PDDL2.1. Yudha performed the skill implementations, implemented the runtime composition method, integrated the software and ran the experiments. Yudha is the primary writer of the article, with a significant contribution from Vahid. Minor adaptations have been made in order to maintain consistent terminologies and notations throughout the thesis. An accompanying video is available online at: https://youtu.be/58BhGmGL3kI

model of constraint-based skills and present methods for executing concurrent skill compositions at runtime without interrupting the ongoing task execution. In order to autonomously compute reactive composed skills at runtime, we propose a temporal description of planning actions, corresponding to robot skills, that ensures concurrency at the task planning level. We validated our work on a dual-arm robot system that performs an industrial assembly task while reacting to obstacle and human-induced disturbances at runtime.

## 4.1 Introduction

Autonomous robots, in order to expand their applications beyond structured environments, need to be endowed with low-level reactive skills and high-level deliberative planning capabilities. Task and Motion Planning (TAMP) [24] is an extensively studied approach to autonomous robotics that finds a feasible task-motion plan by simulating the actions' execution. In TAMP, low-level skills are typically executed using a motion planner. Recent results have shown that TAMP can successfully find solutions in complex scenarios while avoiding collisions [39]. However, most of the TAMP frameworks assume ideal scenarios in which non-deterministic conditions can be simulated and their occurrences are predictable. These assumptions are often too restrictive for real-world scenarios, where disturbances can be difficult to model and predict.

We argue that the robustness of the task execution w.r.t. disturbances can be improved by replacing a motion planner with reactive and composable skills. Motion planning is known to have the advantage of finding collision free paths while avoiding the problem of local optima. However, the generated path is typically static and does not take into account real-time adaptation to disturbances. Some work addressed adapting to disturbances by executing the path with a reactive controller, such as one proposed in [102]. Motion planning applications are also limited to point-to-point positioning tasks. In contrast, reactive skills can perform tasks that prescribe the dynamics of the adaption to disturbances, useful, for example, in tight-tolerance assembly.

In this work, we propose to implement robot skills using constraint-based programming. In addition to reactivity, constraint-based skills are composable [92], that means individual skills can be combined into one skill in order to achieve a composed behavior. Here, we are interested in concurrent skills composition, i.e., skills that must be executed simultaneously. One example is combining a trajectory tracking skill with a collision avoidance skill, resulting in a composed skill that tracks a trajectory and avoids obstacles simultaneously. We address a class of problems where composing appropriate skills prior to

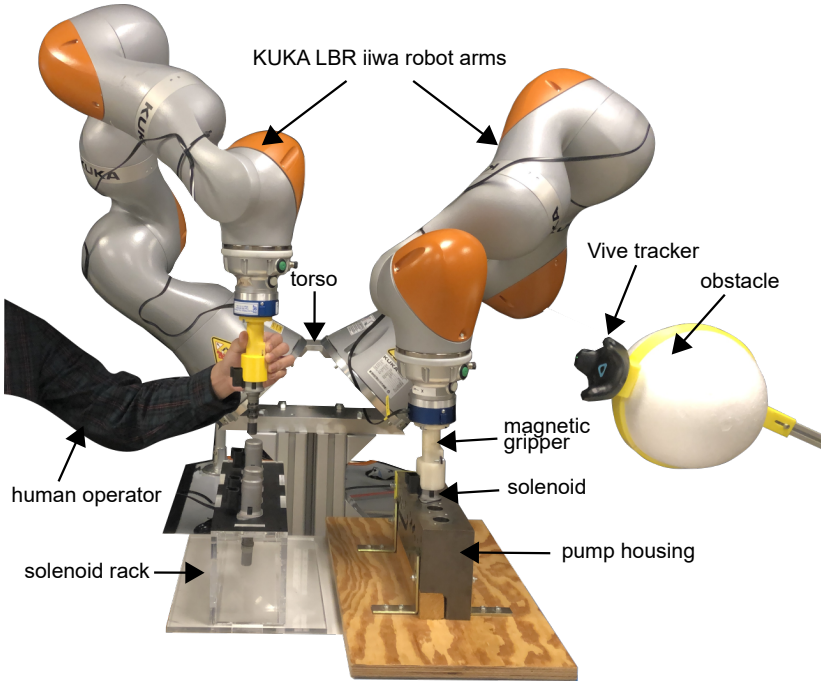



Figure 4.1: A dual-arm setup for the experimental validation of our online replanning and acting framework. The experiment involves an industrial pump solenoid assembly task in a dynamic environment where unforeseen disturbances, e.g., an obstacle (detected with a Vive tracker) and a force-based disturbance by a human operator (detected with force sensors), might enter the workspace. In this framework, reactive robot skills are composable and computed at runtime by a task planner. When a disturbance is detected at execution time, the task planner computes online a composed behavior in order to react to the disturbance.

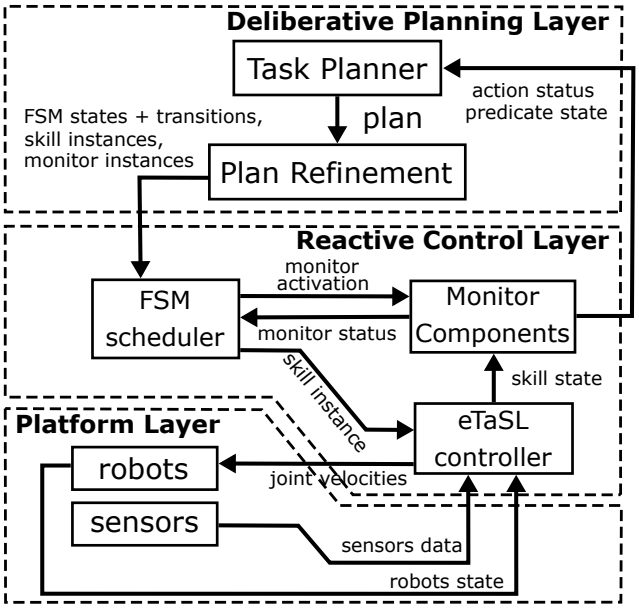



Figure 4.2: A three-layer architecture for online replanning and acting.

execution is infeasible due to unpredictable events of disturbances. Relevant examples are human-robot collaboration scenarios, where occurrences and effects of human actions are difficult to predict (see Figure 4.1). We propose a framework, combining reactive skills with a task planner, that automatically constructs a sequence of skills in order to perform a task in the presence of disturbances at runtime. In this framework, skills are specified and executed using eTaSL [3], a constraint-based robot programming language and controller. Task planning is not only meant to conventionally compute a sequence of actions to fulfill the task, but also to compose reactive skills at runtime. That means, concurrent actions corresponding to skill compositions must be ensured at the task planning level. For this purpose, we adopt a temporal planning model for encoding symbolic actions which enforces a temporal planner to build fully concurrent actions as composed skills.

Figure 4.2 provides an architectural overview of our framework, based on a three-layer architecture for autonomous deliberative robots [64], that combines (*i*) a deliberative planning layer, (*ii*) a reactive control layer, and (*iii*) a platform layer with sensory-motor modules. Given a symbolic description of a task problem, the task planner computes a nominal action plan to achieve the task. Each action is refined into a constraint-based skill and then a finite state machine

(FSM) scheduler loads each skill into an eTaSL controller (eTC) for execution and activates monitors for each skill [3]. When a disturbance is detected, making the execution of the current skill infeasible, a replanning is performed to compute a new plan including a reactive composed skill to deal with the disturbance. This work leverages on the composability property of constraint-based skills, that means skill compositions are computed and introduced at runtime. For example, adding a collision avoidance skill into an already running task without interrupting the task execution.

The chapter makes the following contributions: *i)* A framework of combined task planning and constraint-based skills to compute and execute reactive composed skills at runtime; *ii)* A constraint-based skill specification with runtime composition methodologies; *iii)* A temporal description of planning actions that ensures concurrency at the task planning level.

In the rest of the chapter, we study and discuss related work (Section 4.2). We present our skill model, specification and runtime composition methodology (Section 4.3). We address the symbolic task planning (Section 4.4). We present the experimental evaluation on a dual arm assembly task and discuss the results and conclude the chapter (Sections 4.5 and 4.6).

## 4.2 Related Work

Constraint-based programming [113] is studied in the literature to generate real-time, reactive robot motions. Instead of specifying trajectories, a robot behavior is programmed by specifying constraints that are imposed on the robot kinematic expressions, e.g., a constraint to specify how to reach a desired pose. A robot motion is then generated by solving an optimization problem that computes an optimal actuation command, e.g., joint velocities or accelerations, while satisfying the constraints. This approach also allows for specifying behaviors that react to sensor inputs, enabling a motion to adapt to force or other sources of sensor signals. Some of the well-known methods are iTaSC [113] and eTaSL [3].

Constraint-based programming has been used as an underlying method in developing robot skills [92, 90]. The composability property of constraints offers flexibility to compose skills to achieve complex behaviors. Although existing skill-based frameworks have shown various ways to compose skills, including sequential and concurrent compositions, they assume the compositions remain static throughout the execution [92, 90]. Runtime skill composition translates to sudden change in the null space of the Jacobian matrix, which is known to cause a control input discontinuity [56]. Efforts to mitigate this problem have resulted in methods that are known as continuous tasks transition algorithms

[50] [70]. We extend our previous skill-based framework [92] to facilitate skill compositions at runtime by adopting an approach based on [50] that ensures continuous joint velocities.

On the other side of robot acting is the high-level reasoning, a core deliberation function for intelligent robots to increase their autonomy and flexibility through the construction of sequences of actions for achieving some prestated objective [49, 42]. Manipulation planning has been extensively studied in robotics to compute a sequence of elementary robot motions and grasp and ungrasp actions that reach a goal state from a start state [4]. Manipulation planning relies heavily on geometric reasoning and motion planning in high-dimensional configurations spaces [55]. In [110], authors combine manipulation planning with constraint-based controllers in dynamic environments where two robotics arms hand over an object in the presence of a human disturbance. They model sequential manipulation tasks into a constraint-graph that embodies robot dynamics and time-variance, based on which constraint-based controllers are derived. Despite its robustness, this work does not address autonomous and runtime skill compositions. Moreover, the challenge for this work still remains at scaling up to problems of large size.

The most relevant high-level reasoning techniques for manipulation planning are based on STRIPS [36] and PDDL [79]. These planners take a symbolic description of a problem and a set of actions descriptions, which characterize the pre- and post-conditions of each action, and produce a sequence of discrete actions to achieve a goal. Subsuming manipulation planning into high-level reasoning is burdensome because geometric constraints on robot motions are difficult to express at the symbolic level. Thus robots require to combine symbolic planners with continuous, collision-free motion planning to compute paths to perform discrete actions. This integration problem is addressed as Task and Motion Planning (TAMP) in the literature [24, 123, 67]. However, existing TAMP frameworks often assume static and predictable domains where the executive reactiveness is disregarded. Symbolic task planning and sensor-based reactive execution has also been addressed in the literature. In [136], a task planning and control framework is presented in which symbolic actions are implemented as reactive behaviors into two separate modes determined by the absence or presence of unanticipated obstacles. The reactive behaviors are then mapped into physical joint-level feedback controllers. Nevertheless, this work does not address composability and runtime compositions of reactive behaviors, which are the main contributions of our work. In [115], authors present a scheduler as an interface between a symbolic plan and constraint-based skills. Given a task specification describing how actions should be executed, the scheduler decides when to configure the controller at runtime. This work also does not address the problem of runtime composition and the discontinuity

problem in the control input.

Partial-order planners have been widely used for generating concurrent plans [41]. A partially-ordered plan provides that two unordered actions can be executed in either order, however, it does not enforce the concurrent execution of actions [61]. To ensure concurrent execution, it is essential to explicitly model dependencies between concurrent actions. Temporal planning [37] is a technique that allows for modeling and reasoning about duration and concurrency of actions. Actions might have interdependencies that determine which actions overlap and can be executed simultaneously. Concurrency is also addressed in multi-agent planning [17], however, identifying a multi-agent planner in this work was far from straightforward.

## 4.3 Constraint-based Composable Skills

eTaSL is used as the underlying framework for specifying and executing the skills [3]. This choice is motivated by eTaSL's clear separation between skill specification and execution and its mature software development. Another reason for opting eTaSL is its support of feature variables, which are auxiliary optimization variables that can be used to specify additional degrees-of-freedom. They are useful for specifying complex skills where we can also optimize virtual entities in addition to physical robot joints.

### 4.3.1 Skill Modeling and Specification

A skill represents a well-defined robot behavior and is composed of a set of constraints on kinematic expressions. The expressions are represented as functions of input signals, such as joint positions or sensor measurements. A kinematic expression $e$ is a task function of robot joint variables $q \in \mathbb{R}^{n_r}$, feature variables $\chi_f \in \mathbb{R}^{n_f}$, i.e., variables for specifying additional degrees-of-freedom, and time $t \in \mathbb{R}$,

$$e = f(q, \chi_f, t) \tag{4.1}$$

where $n_r$ and $n_f$ are the number of joint and feature variables, respectively. The task functions can be used to express equality (target values) or inequality (bounds) constraints that compose a skill. Each task function represents a task error that is minimized according to a first order control law:

$$\dot{e} = -ke \tag{4.2}$$

where the feedback gain $k$ can be set to obtain a desirable time constant.

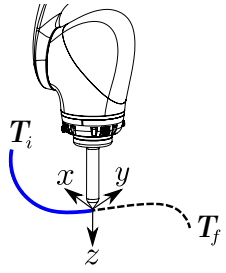
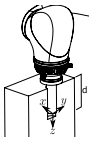
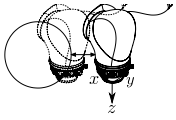



Figure 4.3: A Cartesian motion skill that moves a robot end-effector to a target pose by following a given path (dashed line). The blue solid line is the path already traveled. Task constraints restrict the task frame, i.e., the robot end-effector to stay within the desired path. Progress constraints drive the task frame to move from the initial pose $T_i$ to the end pose $T_f$ along the path.

Table 4.1: Examples of a progressive and a non-progressive skills, their constraints specification and terminating condition.

| Progressive skill: peg-in-hole | Non-progressive skill: collision avoidance |
|---|---|
|  • Task constraints: $f_x = 0$N, $f_y = 0$N $\tau_x = 0$ Nm, $\tau_y = 0$ Nm<br>• Progress constraints: $f_z = F_{\mathrm{d}}$ N<br>• Terminating conditions: insertion_distance $\geq d$ |  • Task constraints: distance(link, obstacle) $\geq d_{\min}$ m<br>• Progress constraints: none<br>• Terminating conditions: none |

A skill is executed by the eTaSL controller by translating its constraints into a quadratic programming (QP) problem. A dedicated solver then solves the QP problem and computes optimal robot joint velocities $\dot{q} \in \mathbb{R}^{n_r}$ and feature variables velocities $\dot{\chi}_f \in \mathbb{R}^{n_f}$ in real-time.

By adopting the skill modeling as proposed in our previous work [92], we distinguish between *progress* and *task* constraints that compose a skill. Progress constraints are defined as constraints that advance a skill from an initial state to a goal state. Task constraints are defined as the remaining constraints that constitute a skill. Figure 4.3 illustrates the different roles of task and progress constraints in a Cartesian motion skill. Here, progress constraints enforce the task frame, i.e., the robot end-effector, to move from an initial pose $T_i$ to an end pose $T_f$. Simultaneously, the task constraints keep the task frame to stay on the desired path.

The separation between constraints introduces a degree-of-freedom (DOF) which can move within the space that is restricted by the task constraints. This DOF

is implemented using the feature variables [3]. The skill can optimize this DOF to result in a behavior that advances the progress and reacts to disturbances or multiple conflicting skills.

A skill developer can assign a relative importance between the task and progress constraints using weights. If the weight of a task constraint is larger than the weight of a progress constraint, the robot prefers to remain in the space restricted by the task constraints and sacrifices its progress whenever a disturbance occurs. Conversely, when progress constraints have larger weights, the robot prefers to maintain a desired progress while deviating from the space restricted by the task constraints. The deviating behavior can be achieved by varying the constraints' weights. As a result, more predictable reactive behaviors can be specified in non-nominal situations.

Based on our notion of task and progress constraints, we define skills as *progressive* and *non-progressive*. A progressive skill is composed by both task and progress constraints. This type of skill results in a behavior that advances to a desired state and terminates once the state is reached. A non-progressive skill is only composed of task constraints without any terminating condition. Non-progressive skills typically only produce reactive motions in the presence of disturbances. An example is a collision avoidance skill where the robot only moves when there is an obstacle in proximity. Since non-progressive skills do not make progress by themselves, they are commonly composed concurrently with progressive skills. Table 4.1 shows an example of a progressive and a non-progressive skill along with their constraints specification. The distinction between progressive and non-progressive skills is also reflected on the task planning-level domain encoding (see Section 4.4).

### 4.3.2 Runtime Composition of Skills

In this work, we are interested in the problem of concurrent skills composition, i.e., skills that must be executed simultaneously. Once a skill starts, it will be executed by the eTaSL controller by including the skill in the QP optimization problem. We define the *active skills set*, $\mathbb{S}_a$, as a list of skills currently under execution: $\mathbb{S}_a = \langle \mathsf{s}_1 \dots \mathsf{s}_{n_s} \rangle$ where each $\mathsf{s}_i$ for $1 \leq i \leq n_s$ is a running skill.

Runtime skill composition deals with the problem of introducing a new composed skill during execution. This is illustrated in a Gantt chart of the skill execution in Figure 4.4. Initially, the robot executes a skill sequence refined from the nominal plan (solid lines). At time $t_1$, the active skills set consists of only one skill $\mathbb{S}_a = \langle \mathsf{s}_1 : \mathsf{approach_to_grip} \rangle$. At time $t_2$, an obstacle is detected nearby, therefore a replanning is performed that introduces a new skill $\mathsf{s}_3$:avoid_collision (dashed lines) to be composed concurrently with $\mathsf{s}_1$:approach_to_grip. Execution of the

new plan is performed from time $t_2$ onward. Finally, when the composition is carried out, at time $t_3$, the active skills set consists of two running skills: $\mathbb{S}_a = \langle \mathsf{s}_1{:}\mathsf{approach_to_grip}, \mathsf{s}_3{:}\mathsf{avoid_collision} \rangle$.

Runtime skill composition poses a problem to the eTaSL controller since the null space of the Jacobian of the currently active skills is suddenly changed upon introducing a new composed skill. This creates discontinuity in the control input, i.e., the joint velocities command of the eTaSL controller. The control input discontinuity for constraint-based controllers is addressed in [56]. Some strategies have been proposed to address this, e.g., using linear interpolation of the optimization solutions [56], modifying the target values of the constraints [70], and adapting the constraints weights [50].

We adopt the proposed approach in [50] since it avoids creating multiple QP problems in [56] and the modification of the skill specification in [70]. This approach ensures a smooth transient in the control input by adapting the weights of skills in the active skills set. The eTaSL controller used in our work only supports two levels of priority: the higher level signifies hard constraints that should not be violated while the lower level signifies soft constraints whose importance can be set with weights. An extension to arbitrary levels such as one proposed in [34] is also possible.

Using the scenario depicted in Figure 4.4, at time $t_1$, the eTaSL controller only takes into account $\mathsf{s}_1{:}\mathsf{approach_to_grip}$ skill. Assuming that $\mathsf{s}_1$ is performed with a lower priority level, the QP problem to be solved is as follows [3]:

$$\underset{x}{\text{minimize}} \quad x^\top \mathbf{H}^{\mathsf{s}_1} x \quad \text{subject to} \quad c_1^{\mathsf{s}_1}, \ldots, c_n^{\mathsf{s}_1} \tag{4.3}$$

where $\mathbf{H}$ denotes the diagonal QP optimization Hessian, and each $c_i$, for $1 \leq i \leq n$, is a shorthand notation of a single row constraint. The superscript signifies the skill considered in the QP problem. Each $c_i$ denotes an inequality constraint obtained by expanding the total derivative of the task function error (4.2):

$$\mathbf{J}_i \begin{bmatrix} \dot{q} \\ \dot{\chi}_f \end{bmatrix} \leq -k e_i - \frac{\partial}{\partial t} e_i + \varepsilon_i \tag{4.4}$$

where $\mathbf{J}_i$ is a Jacobian row from the partial derivatives of the task function. The slack variables $\varepsilon$ are introduced to allow conflicting soft constraints. In addition to a feedback term $-ke_i$, the constraint equation also contains a term $\partial e_i / \partial t$ that can be interpreted as a feedforward term to avoid tracking errors for task functions that explicitly depend on time. The optimization variable $x$ consists of joint velocities, feature variable velocities and slack variables that belong to $\mathsf{s}_1 : x = [\dot{q}^\top \dot{\chi}_f^{\mathsf{s}_1\top} \varepsilon^{\mathsf{s}_1\top}]^\top$. For simplicity, we assume all constraints in a skill have a uniform priority level. The Hessian consists of the constraints

weights:

$$\mathbf{H}^{\mathsf{s}_1} = \begin{bmatrix} \mu\mathbf{W}_q & 0 & 0 \\ 0 & \mu\mathbf{W}_f^{\mathsf{s}_1} & 0 \\ 0 & \mathbf{0} & \mu\mathbf{I} + \mathbf{W}_\varepsilon^{\mathsf{s}_1} \end{bmatrix} \tag{4.5}$$

where $\mathbf{W}_q$, $\mathbf{W}_f^{\mathsf{s}_1}$ and $\mathbf{W}_\varepsilon^{\mathsf{s}_1}$ denote diagonal weight matrices corresponding to the joint, feature, and slack variables, respectively. A small numerical value $\mu \in \mathbb{R}$ is added for regularization and to ensure positive definiteness. More details about the construction of the optimization Hessian and the objective function used in eTaSL is described in [3].

Continuing the scenario, at time $t_2$, $\mathsf{s}_1$ and $\mathsf{s}_3$ are composed concurrently. Assuming that $\mathsf{s}_3$ has to be performed with higher priority level where $p^{\mathsf{s}_3} > p^{\mathsf{s}_1}$, i.e., composed of hard constraints, a smooth transient behavior is ensured as follows. As soon as the composition starts, $\mathsf{s}_3$'s priority level is assigned to the lower level such that $p^{\mathsf{s}_3} = p^{\mathsf{s}_1}$. Then the appropriate scaling factors are applied to the constraints weights, resulting in a modified QP problem:

$$\begin{aligned} \underset{x}{\text{minimize}} \quad & x^\top \begin{bmatrix} \tilde{\mathbf{H}}^{\mathsf{s}_1} & 0 \\ 0 & \tilde{\mathbf{H}}^{\mathsf{s}_3} \end{bmatrix} x \\ \text{subject to} \quad & c_1^{\mathsf{s}_1}, \ldots, c_n^{\mathsf{s}_1} \\ & c_1^{\mathsf{s}_3}, \ldots, c_m^{\mathsf{s}_3} \end{aligned} \tag{4.6}$$

where $n$ and $m$ are the number of constraints for $\mathsf{s}_1$ and $\mathsf{s}_3$, respectively. The weight matrices of the slack variables are premultiplied with the appropriate scaling factors:

$$\begin{aligned} \tilde{\mathbf{W}}_\varepsilon^{\mathsf{s}_3} &= \alpha \mathbf{W}_\varepsilon^{\mathsf{s}_3} \\ \tilde{\mathbf{W}}_\varepsilon^{\mathsf{s}_1} &= \beta \mathbf{W}_\varepsilon^{\mathsf{s}_1} \end{aligned} \tag{4.7}$$

The scaling factor $\alpha$ and $\beta$ follow the profiles shown in Figure 4.5. Slightly different to [50], we divide the transient time in two phases: $t_0 \leq t \leq t_i$ and $t_i < t \leq t_f$. At the first phase, $\mathsf{s}_3$ is smoothly activated while $\mathsf{s}_1$'s influence is diminished. At $t = t_i$, the priority switches and $\mathsf{s}_3$'s priority level is increased. Finally, in the second phase, $\mathsf{s}_1$'s weight is restored by applying scaling factor $\gamma$ to the slack variables weight matrix $\tilde{\mathbf{W}}_\varepsilon^{\mathsf{s}_1} = \gamma \mathbf{W}_\varepsilon^{\mathsf{s}_1}$. This method ensures a smooth transient upon the addition of a new skill to the active skills set. As a result, we can perform runtime skill compositions safely as soon as a new plan is generated.

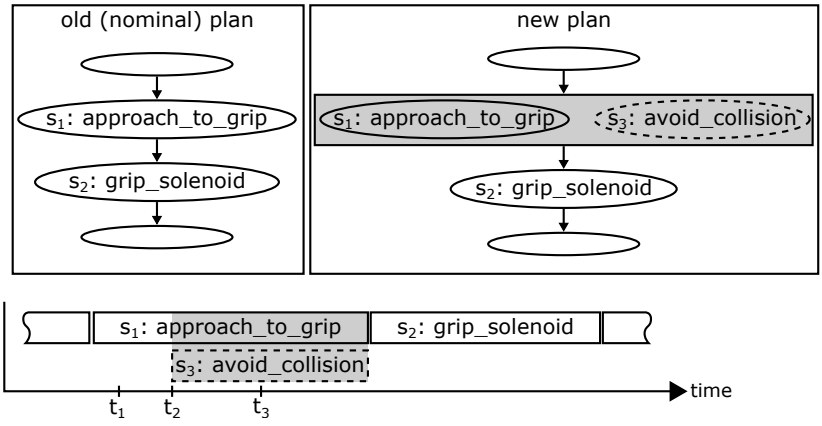



Figure 4.4: Plans containing skills sequences (top) and Gantt chart representing skills execution (bottom) to illustrate concurrent skill compositions at runtime.

## 4.4 Task Planning

We use task planning for two purposes. On the one hand task planning is conventionally used for finding a *plan*, i.e., a solution to a task planning problem consisting of a set of (totally/partially) ordered actions, which leads from an initial state to a goal state [41]. On the other hand task planning is used to compute composed skills at runtime when disturbances are detected, making the execution of current actions infeasible, that is, output plans must include concurrent actions as composed skills. When multiple actions have to be taken simultaneously, it is necessary to model the concurrency of actions. For this purpose, we use PDDL2.1, an extension to PDDL for expressing temporal planning domains and durative actions [37]. PDDL2.1 allows the specification of temporally annotated conditions and effects enabling actions to be described in which external factors can be involved in determining their temporal extent.

We propose to encode non-progressive skills as durative actions with temporal conditions and effects which can then enforce a planner to exploit concurrency, a requisite for autonomously composing skills. A durative action, similar to classical actions, is represented by a set of conditions specifying the action's precondition, and a set of effects to change a state where the action is applied. However, unlike classical actions, every condition and effect must be temporally annotated. The annotation of a condition specifies whether the associated proposition must hold at the *start* point at which the action is applied; the *end* point at which the final effects of the action are asserted; or *over* the duration of the action. (from the start to the end). Likewise, the annotation of an effect

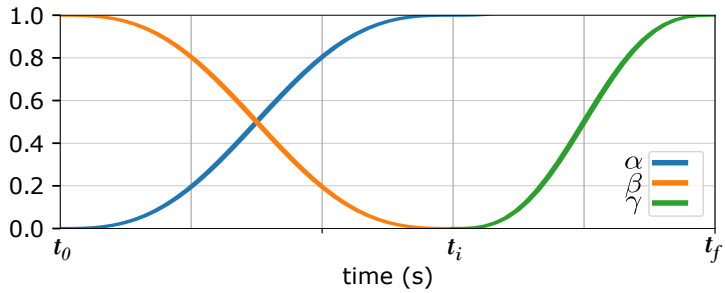



Figure 4.5: The weight scaling factor profiles to achieve smooth transient behavior at runtime composition of skills. The transient is divided into two phases: $(i)$ at $t_0 \leq t \leq t_i$, the weight factor $\alpha$ is applied to the newly introduced skill, and simultaneously the old active skill is scaled with weight factor $\beta$. At $t = t_i$, the priority is changed for the new skill; $(ii)$ at $t_i \leq t \leq t_f$, the old active skill weight is restored by applying a scaling factor $\gamma$.

specifies whether the effect is immediate (happens at the start of the action) or delayed (happens at the end of the action). Progressive skills, by contrast, are encoded as classical actions.

The given example in Figure 4.6 illustrates how we model concurrency in this work. Let us assume that the durative action avoid_collision corresponds to a non-progressive skill and the action approach_to_grip corresponds to a progressive skill. In order to start the action approach_to_grip, the condition collision_free must hold in a state. So, when a collision is detected in a state, the action avoid_collision starts and immediately produces the effect collision_free in the state, allowing the action approach_to_grip to start. However, the action avoid_collision only completes when its precondition arm_at holds at the end, produced by the action approach_to_grip. This model ensures a planner to compute a plan that the duration of the action avoid_collision involves the action approach_to_grip. Given the action descriptions and the problem in Figure 4.6, a plan including concurrent actions with their start points and duration is produced. The interpretation of this concurrent plan is also illustrated.

We identify the off-the-shelf temporal planner OPTIC [13] that could support reasoning over our domain encoding and computing concurrent plans. We develop an online replanning interface which interacts with OPTIC and the skill execution engine, i.e., the eTaSL controller. Given a temporal planning domain including durative action descriptions, as well as the initial state and the goal of a task, the task planner computes a (possibly concurrent) plan to achieve the task. The computed plan is then translated and refined into a set

**Actions**

```
(:durative-action avoid_collision
 :parameters (?a - arm ?d - location)
 :duration (>= ?duration 0)
 :condition (and (at start (collision_detected ?a ?d))
                 (at end (arm_at ?a ?d)))
 :effect (and (at start (not (collision_detected ?a ?d)))
              (at start (collision_free ?a ?d))))


(:action approach_to_grip
 :parameters (?a - arm ?s - location ?d - location)
 :precondition (and (arm_at ?a ?s)
                    (free ?d)
                    (collision_free ?a ?d))
 :effect (and (arm_at ?a ?d)
              (free ?s)
              (not (free ?d))
              (not (arm_at ?a ?s))))
```

**Problem**

```
Initial state: (arm_at arm location1)
               (free location2)
               (collision_detected arm location2)
Goal: (arm_at arm location2)
```

**Plan**

```
Plan: 0.0: (avoid_collision arm location2) [0.2]
      0.1: (approach_to_grip arm location1 location2) [0.1]
```

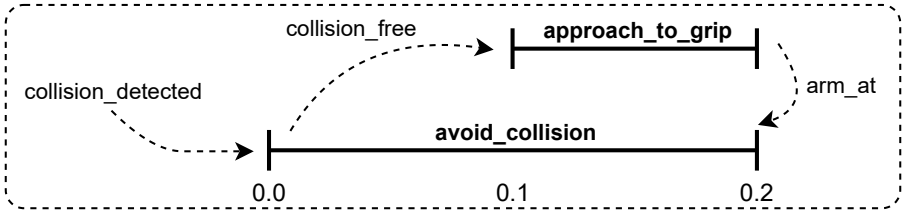



Figure 4.6: An example of a problem with a durative action useful for enforcing concurrency. Temporally annotated conditions in the durative action avoid_collision, specify when the action can start, continue or end, and when it can produce effects immediately or at the end of the action. The avoid_collision action produces the collision_free condition necessary to start the approach_to_grip action, however, it only ends when the arm_at condition holds by the approach_to_grip action. The model ensures that two actions run concurrently.

of skills to be executed by the robot (see Figure 4.2). During runtime if some actions' conditions are violated due to unpredicted disturbances, e.g., a collision or a human admittance, the planning state is updated with the description of

disturbances and a new plan is attempted, containing concurrent actions as composed skills that react to disturbances.

## 4.5 Experimental Validation

We evaluated our framework on an industrial pump solenoids assembly scenario performed by a dual-arm robot. The goal of this experiment is to show the effectiveness of the combined task planning and reactive skills in: *i)* finding nominal plans to assemble a desired number of solenoids with the two arms concurrently, and *ii)* replanning and composing concurrent reactive skills upon the detection of action failures due to unforeseen disturbances.

### 4.5.1 Experimental Setup and Task Scenario

Our experimental setup was shown in Figure 4.1. It consists of two 7-DOF KUKA LBR iiwa robots mounted on a torso. The robots have an overlapping workspace such that collision between them can occur. The obstacle is attached with a Vive tracker, and its pose is tracked by a pair of Vive SteamVR Base Station 2.0. The task is to assemble solenoids for an industrial pump. Both robots assemble the solenoids simultaneously in order to minimize idle time.

The following disturbances can occur at runtime: *i)* An obstacle may suddenly appear in the workspace where the robots are performing free-space motions. This disturbance is detected by a monitor that checks the distance between the obstacle to the robots links using the Vive tracker's measurements. As soon as an obstacle is detected, the monitor issues an event describing the obstacle and conflicting arms, that invokes the task planner to compute a new plan containing a concurrent skill composition with a collision avoidance skill. *ii)* A human operator may disturb any of the two robots by exerting force to the end-effector. This disturbance is detected by a monitor that triggers when the end-effector wrench measurements exceed a predefined threshold. Similarly, after detecting a human disturbance, the task planner recomputes a solution that composes an admittance behavior to follow the exerted force direction.

We note that collision avoidance between the two robots is set as a hard constraint which is always active during the task execution. It is also worth mentioning that our monitors implementation is edge triggered, i.e., when a disturbance is detected, an event is fired once. The monitor is reset after one action execution finishes. This way, we avoid frequent event triggers that would cause repeated replanning.

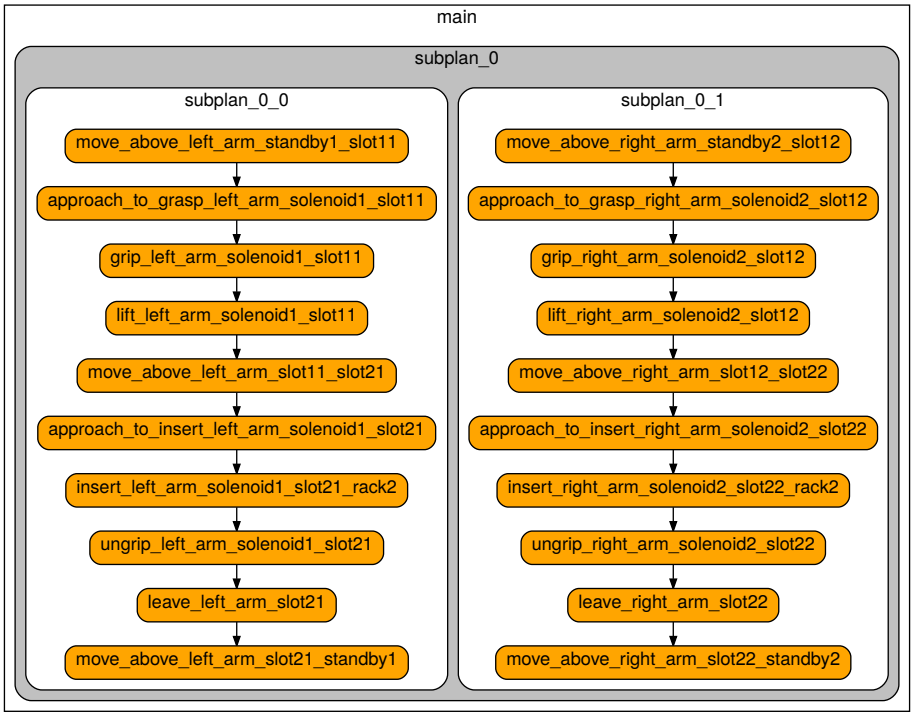



Figure 4.7: The initial concurrent plan for the task of two solenoids assembly. The plan includes subplans represented as rectangles: total order (white), partial order (light gray). Actions are shown in orange color. subplan_0 involves subplan_0_0 and subplan_0_1 for the left and the right arms respectively, that can be executed in a partial order manner. Each of subplan_0_0 and subplan_0_1 consists of a sequence of totally-ordered actions.

### 4.5.2 Experimental Results

We begin with an assembly task for two solenoids without any disturbances. The task problem involves the following goal propositions:

```
(:goal (and (inserted_in solenoid1 slot21)
            (inserted_in solenoid2 slot22))))
```

specifying the final assembly positions of solenoid1 and solenoid2, respectively. Given the initial state and the goal above, the OPTIC planner computes a plan to achieve the goal. It is worth mentioning that although OPTIC computes a partially-ordered plan for assembling two solenoids, the plan does not consider

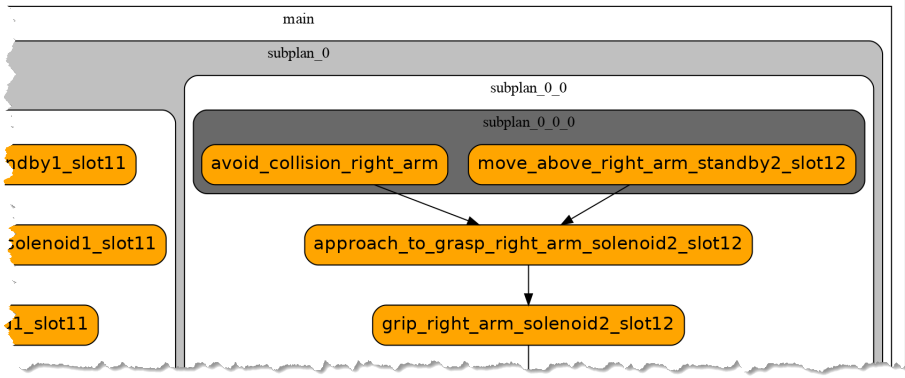



Figure 4.8: Part of a new plan computed after detecting an obstacle to right_arm. The plan contains a concurrently composed skill, move_above and avoid_collision, shown in dark gray.

minimizing the execution time when there are multiple agents. For example, the produced partially-ordered plan allows both arms to simultaneously pick up two objects, but, it does not transition to a next action if one arm fulfilled its action sooner than the other arm. So, in order to exploit multi-robot execution as fully as possible, we adopt a parallelization algorithm, similar to [137], to translate the produced plan into a concurrent plan as shown in Figure 4.7. The plan can then be executed by two arms concurrently.

Due to unpredictable appearing of disturbances to different arms and different solenoids, we are interested in computing and composing concurrent reactive skills at runtime. To this end, we test the robustness of our framework by introducing disturbances at different steps of the plan execution. As a result, the task planner replans multiple times during the execution to incorporate unpredicted disturbances into the plan. For example, Figure 4.8 shows part of a plan that composes (move_above right_arm standby2 slot12) with (avoid_collision right_arm) when an obstacle appears to right_arm at runtime. In this experiment, we showed the successful integration of an online replanner with the concurrent skill composition while reacting to disturbances at runtime. Figure 4.9 shows snapshots of the two solenoids assembly task execution in the real robots.

We further validated our framework in an assembly task of four solenoids. Table 4.2 summarizes the total number of replanning, the longest and shortest replanning time, and the task completion time in both experiments. The total

number of replanning correspond to the total action failures that are detected by the monitors. We introduced more disturbances to the four solenoids assembly task. In the table, it can be understood that the planning time increases as the problem size becomes larger. In the four solenoid assembly task, the longest replanning time is obtained when the planner has to compose multiple reactive skills, i.e., when the obstacle and the force disturbances are introduced to both robots at the same time. However, as the plan execution progresses, the replanning time becomes significantly shorter since the search space is reduced for the remaining solenoids to assemble.

Additionally, we performed the assembly task in a simulated environment where the locations of the solenoid rack and pump housing change dynamically. This experiment shows how robust the Cartesian motion skills are in dealing with dynamic target poses without requiring any modification in the skills. A video of our experiments is available online at: https://youtu.be/58BhGmGL3kI.

### 4.5.3 Conflict Resolution in Multiple Skills Execution

Since the task planner generates concurrent plans, there will be conflicting skills when two arms need to occupy the same workspace. We adopt a set of heuristics for resolving conflicts that might occur in multi-robot skills execution. These heuristics give a preference for choosing between two conflicting progressive skills, e.g., when two arms are performing Cartesian motion skills. The following heuristics are implemented in this work: *i) Manipulation heuristic* ($h^m$): a preference is given to a skill that is manipulating/holding an object; *ii) Action advancement heuristic* ($h^a$): a preference is given to a skill that has further advanced in the plan. For example, in Figure 4.7, the skill that executes approach_to_insert is given a higher priority over the skill that executes approach_to_grasp. *iii) Skill progress heuristic* ($h^s$): a preference is given to a skill with a higher skill progress value. A skill progress value is a skill-specific metric that measures how far the skill has completed, e.g., the distance traveled, as used in our previous work [92].

We then compute the overall preference, $\rho$, for each arm by the canonical combination of the above heuristics as follows, $\rho = w_1 * h^m + w_2 * h^a + w_3 * h^s$, where $h^m$, $h^a$, and $h^s$ are binary values corresponding to the above heuristics, and $w_i$ are non-negative real numbers s.t. $w_1 > w_2 > w_3$. The skill with a higher overall preference is assigned a higher relative weight in the QP optimization. These preferences are computed and the weights are assigned to the skills at runtime, thus avoiding the interruption of ongoing skills and getting stuck in a local optima.

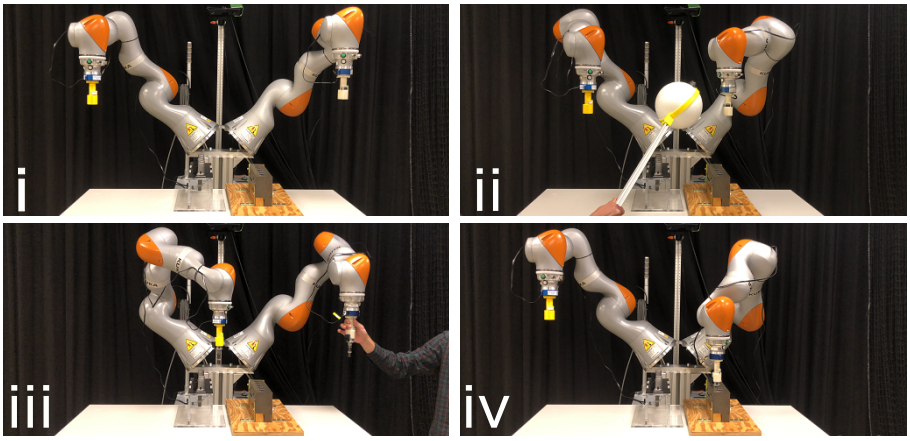



Figure 4.9: Snapshots of the two solenoids assembly task execution. The arms start from standby poses (i). In (ii), they execute the first actions when an obstacle suddenly enters and blocks the robots' path. Here, a new plan is computed which composes reactive skills to avoid the obstacle. In (iii), another disturbance occur in which a human operator pull the right arm from its nominal path. Again here, the task planner recomputes a new plan and composes a reactive admittance skill. In (iv), the left robot has already inserted one solenoid while the right arm is still performing the insertion.

### 4.5.4 Smooth Transient in Runtime Skill Composition

We present an example of a transient behavior at runtime skill composition using the method explained in Section 4.3.2. Figure 4.10 shows the elbow joint velocities of the robot when a new action avoid_collision is introduced while the robot is performing the move_above action, resulting in computing a concurrent skill composition. The plot shows that without the transient method, the joint velocities suffer from discontinuity, i.e., infinite acceleration. By adapting the weights of skills in the active skill set, smooth transient velocities are achieved. The smoothness of this transient is affected by the duration of transient phases, i.e., $[t_0, t_i]$ and $[t_i, t_f]$, that is, the shorter the duration, the less smooth transient joint velocities. We note that these values are tuned by considering a trade-off between reactivity and smoothness. In our experiment, we set the duration of both phases to 0.25 s.

Table 4.2: Relevant information for the two assembly task scenarios.

| Metric | Two solenoids | Four solenoids |
|---|---|---|
| Total number of replanning | 9 | 11 |
| Initial planning time (s) | 0.15 | 0.458 |
| Longest replanning time (s) | 0.178 | 0.756 |
| Shortest replanning time (s) | 0.025 | 0.031 |
| Task completion time (s) | 90 | 168 |

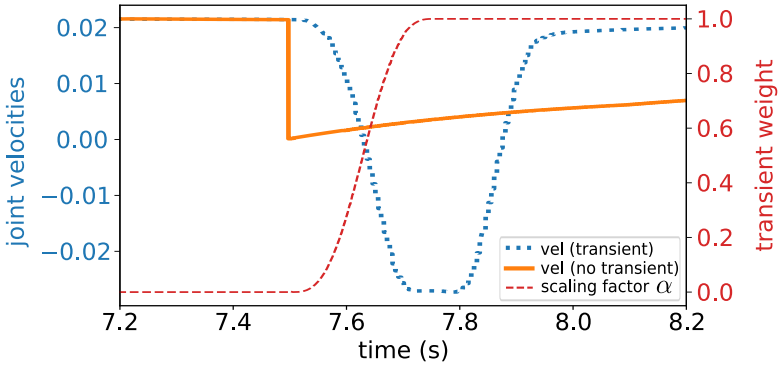



Figure 4.10: The elbow joint velocities at runtime composition with and without the transient method. Discontinuous velocity is obtained when the skill is introduced without scaling the weights in the optimization problem. In contrast, a smoother velocity is achieved when the skill's weights are scaled.

## 4.6 Discussion and Conclusion

We addressed the problem of autonomous concurrent reactive skill compositions at runtime. We proposed a model for planning actions, corresponding to robot skills, based on temporal planning in PDDL2.1 that ensures concurrency, such that a temporal task planner can compute concurrent actions as composed skills. Robot skills are specified using a model that separates constraints into task and progress, such that desired behaviors can be achieved by composing these constraints. In order to preserve the continuity of the joint velocities command upon composing skills at runtime, we implemented a method that smoothly scales the skills' constraints weights.

We validated this work in a real-world industrial solenoid assembly task using two robotic arms, demonstrating how our system reacts to disturbances by computing concurrent composed skills at runtime. Despite achieving fairly good results, engineering optimizations are possible on the proposed framework. For

example, in the current work, when a disturbance is detected, the replanning is performed to achieve the original goal of the problem. This might cause some inefficiency, in particular, in large problems and pose a risk of high computational time at runtime. The future work includes replanning to recover from the failed action, rather than planning for the whole problem, and preserve the original plan to continue after reacting to disturbances.

# Chapter 5

# A System Architecture for CAD-based Robotic Assembly with Sensor-based Skills.*

## Abstract

Specifying assembly tasks in computer-aided design (CAD) level is a promising approach to intuitively program complex robot skills. In this article, a three-layered system architecture is presented to generate sensor-based robot skills from an assembly task instance. The architecture consists of an application layer where the user instantiates assembly tasks by specifying CAD constraints between geometric primitives pairs. A process layer infers the most suitable robot skills and their appropriate parameters. This inference is made possible by reasoning on a knowledge database represented as an ontology. The ontology contains semantic models of relevant classes such as tasks, skills, and geometric

*This chapter is based on: Pane, Y., Arbo, M. H., Aertbeliën, E., and Decré, W. A System Architecture for CAD-Based Robotic Assembly With Sensor-Based Skills. In *IEEE Transactions On Automation Science And Engineering*, 17 (3), pp. 1237-1249 (2020). Yudha Pane was the primary writer of the article, with a significant contribution from Mathias Arbo. Mathias developed the FreeCAD workbench for annotation of assembly tasks and the RViz part frame publisher. Yudha implemented the Prolog and fuzzy inference software, and the extended OWL ontology. Yudha performed the compressor assembly experiment at KU Leuven while Mathias performed the motor assembly experiment at NTNU. Minor adaptations have been made in order to maintain consistent terminologies and notations throughout the thesis. An accompanying video is available online at: https://youtu.be/WkVxxnEgyTU

primitives as well as the relations between them. A control layer executes the sensor-based skills in real time using the eTaSL programming framework. A software implementation for the three layers is presented. The application layer is implemented in FreeCAD, whereas the process layer consists of a Web ontology language (OWL) ontology, a Prolog-based reasoner, and fuzzy inference to correctly select the skill and generate its parameters. In the control layer, the instantiated eTaSL skills execute the assembly tasks by sending an optimized control command to the robot. The system is validated on two challenging assembly cases with two distinct robot types, thus demonstrating the system's capability across different scenarios.

## 5.1 Introduction

Recent advancement in robotics technology provides the opportunity to perform complex tasks that were previously considered unsuitable for automation. Examples are tasks that require on-line sensor interaction as well as human-robot cooperation. The increasingly more complex tasks, however, also mean an increased programming effort. One of the main reasons is that task specification and parameterization require sufficient expertise and is often written in robot-specific languages. The traditional way of programming is excellent for specifying simple tasks such as point-to-point motion, but unsuitable for complex tasks such as force-based assembly. Not only does this make it difficult for non-robot experts to program challenging tasks, it also increases the associated cost. Therefore, there is a need for more intuitive methods to program robot tasks.

Computer-Aided Design (CAD) is a rich source of information that is widely used by manufacturing companies and can potentially help reduce the robot programming effort. The CAD information includes part dimensions, geometric features, mating descriptions, etc. In this work, we are interested in exploiting CAD information to solve a particular class of robotic tasks, namely assembly. Although this problem has been addressed in the literature, there are still challenges to be solved. Our system aims to improve upon existing works by introducing a more systematic grounding from assembly specification in CAD-level to robot motion control. Inspired by the work of Perzylo et al. [99], we implemented an extended ontology which, additionally, covers force-based task classes. The ontology stores knowledge about different CAD primitives, assembly constraints, robot motion constraints, as well as their relations. By reasoning with this knowledge, suitable skills to execute a given assembly task can be deduced and suggested to the user.

In designing the system, we advocate for: a workpiece perspective, composability,

and sensor integration. A workpiece perspective allows us to specify the task's constraints relative to the parts. This makes the system more intuitive for users with less robot expertise, and it gives a robot-agnostic formulation of what is to be achieved. Composability allows us to combine different models of the environment, workpiece, and robot. This property is important in order to improve reusability of the assembly program. Sensor integration, in particular for force control, is a key to support a larger class of assembly tasks.

The key contribution of this work is the design of a system architecture and software implementation that facilitates instantiation of constraint-based skills to execute position and force-based assembly tasks specified with CAD semantics. For the skill implementation, we opted for the eTaSL framework [3] due to it being one of the state-of-the-art frameworks for constraint-based robot programming and its relatively mature software development. The grounding from the CAD-level assembly specification to executable skills is broken down into two main problems: *skill selection* and *skill parameterization*. The former deals with finding the suitable skills from the provided library for a given assembly task. To this end, we extend our initial work from [10] by adding a reasoning module implemented in Prolog to search through the ontology for the appropriate skills. Once a skill is chosen, it needs context-dependent parameters in order to be executable. These skill-specific parameters are split into two groups. One group, namely application parameters, are geometry-related and can be inferred from the CAD information. Examples of this class of parameters are assembly direction and insertion depth. The second group, termed process parameters, are difficult to completely determine in a model-based way, hence they are typically tuned. Parameters such as insertion force and anti-jamming dither belong to this parameter class. In this work, we propose a data-driven approach to estimate the appropriate process parameters using a fuzzy inference method. The resulting parameters are then used to instantiate an eTaSL-based skill. Using this architecture, complex force-controlled assembly tasks can be executed even though the skills rely on empirical parameters.

In our previous work [10], the mapping from CAD-level specification to concrete skills was done statically without involving any reasoning capability. This limits the reusability of the skills and puts more programming load on the user. The differentiating contributions of this article compared to our previous work are as follows. First, we introduce a knowledge database which is represented as an ontology to encode semantic information of CAD-compliant geometric models as well as force-based skill constraints. Secondly, we implement an inference module using logic programming to reason on top of the ontology in order to assist in grounding from CAD-level specification to suitable skills. Finally, we present more extensive use cases by showing a different assembly problem performed by a different robot setup.

This article is organized as follows. Section 5.2 summarizes the previous works related to our research. Section 5.3 describes the overview of the system architecture and defines tasks and skills. The theory behind the ontology knowledge database and the inference module is covered in Section 5.4. Section 5.5 explains the implementation detail of each system's component. This includes the software implementation of the CAD workbench and the inference module. The experimental validation on two different assembly scenarios is presented in Section 5.6. Section 5.7 discusses the experimental results.

## 5.2 Related Works

One of the earliest works in exploiting CAD information to generate robotics assembly program was presented by Ambler and Popplestone [6], where they proposed a constraint-based description of pose relations between geometric features of assembly parts. Reference frames attached to the geometric features were used to define the end-product of assembly in terms of "fits" or "against" relationships. The relative pose between these frames was described by equality or inequality constraints.

AUTOPASS [72] is an early CAD-based assembly programming system that starts from a workpiece rather than a robot-centric perspective. AUTOPASS includes modules for precalculating trajectories that are executed by the robot with basic use of sensor data for ensuring the success of the assembly task. Archimedes 2 [54] describes an architecture where the CAD model and assembly description, sequence planning, and motion control execution are separated into modules. Archimedes 2 defines two forms of assembly tasks: mating and joining. Mating is establishing a relative positioning of the parts along a translation, rotation, or screw trajectory. Joining is attaching the part in its relative position by either gluing or welding, while using a predefined tool for the particular task. One of the most advanced systems is HighLAP [83] [134] where the CAD model is annotated with a small set of simple semantic assembly descriptions. The descriptions give constraints similar to Ambler and Popplestone that are used to find remaining degrees-of-freedom. HighLAP uses the task frame formalism to execute force-controlled motion [30]. Tasks are executed by a "net" of skill primitives, i.e., a finite-state machine of skills with sensor-based transition conditions. From the geometric constraints between the parts, the tasks are classified, e.g., as object placing or peg in hole, and the particular suitable net of skill primitives is employed [132].

The work by Perzylo et al. uses object-centric programming based on geometric constraints between parts to simplify the robot motion programming [98]. The

comparison of the object-centric programming and classical teach-pendant based programming shows that the former is faster. The underlying framework is presented in [122] where Somani et al. proposes a robot programming framework specified with CAD semantics. The framework translates the inherently underspecified CAD constraints into a position-resolved optimization problem. The framework makes use of the Boundary Representation (BREP)-based ontology explained in [99] for annotation of geometric constraint relationships between CAD parts. By defining an ontology for geometric constraints and CAD primitives, other secondary information such as the motion null-spaces formed by constraints can be exploited to find the assembly paths. The work of Perzylo et al. is a part of the EU funded SMErobotics consortium [97] which aims to simplify production systems for small and medium-sized enterprises. The consortium includes a product-process-resource technology model [65] that allows process-level information such as welding voltage to be incorporated through the technology model. The research presented by the consortium approaches a wide range of manufacturing domains from assembly to welding, and includes a product-centric robot instructions and uncertainty-aware robot skills [131].

The aforementioned literature deals with a common problem: to ground a higher-level abstraction of task specification in the CAD-level into the lower-level abstraction of robot motion. Although they bring significant progress in solving the problem, there are still some limitations to be resolved. In the AUTOPASS language, assembly specification is refined into robot motion using a dedicated compiler that composes functional modules. However, the language lacks semantics of CAD primitives and CAD-specific constraints as it was developed before the BREP-format becoming popular. This makes grounding to the corresponding robot motion more difficult due to the specification's ambiguity and reliance on outdated formats. Archimedes 2 presents results for position-based assembly tasks, but does not treat force-based assembly problems which are often found in real-life assemblies. Furthermore, the system generates robot-specific function calls which implies robot vendor dependency, thus reducing its reusability. The framework proposed by Perzylo and Somani et al. covers more comprehensive classes of CAD primitives, assembly constraints, and their mapping to the corresponding robot motion constraints. However, the framework is limited to position-based assembly problems. The uncertainty-aware robot skills of Thomas et al., are generalizable skills that include force-controlled motions but are yet to be unified with the CAD assembly constraints of Perzylo et al. The HighLAP system aims to fully automate an assembly task using contact formation graphs to eventually generate manipulation primitive nets. The disadvantage of such an approach is that it requires state estimation which is prone to errors due to measurement uncertainties and disturbances. Furthermore, constraints between two parts is specified with relations such as

fits, against, and aligned. The semantics of these relations are typically not supported in modern CAD software.

Motions for assembly are easily defined w.r.t. the workpiece, environment, or robot using constraints, and the constraints are naturally composable. Mason's task frame formalism [78] presents an early work for describing fine motion in the task space by introducing the notion of *natural* and *artificial* constraints. It shows how different control modes can be applied independently along instantaneous task frame's directions, both in translation and rotation. In [113], De Schutter et al. describe a procedure to design a robot controller that can deal with sensor interactions and motion in contact, using a set of auxiliary frames to systematically describe the constraints. This approach is implemented in the software framework iTaSC [120]. Subsequently, expression graphs are used to simplify constraint-based programming leading to the specification language eTaSL [3]. eTaSL facilitates the specification of controller behavior using constraints that relate to geometry or sensor-input. Furthermore, it also allows specifying monitors that trigger events upon certain conditions such as success or failure. These events can be captured by discrete-time coordinators such as finite state machine (FSM) or behavior tree to facilitate transition between skills.

## 5.3 Overview

The system architecture is designed by following separation of concern principle, such that different level of abstraction is treated separately. We describe the three layers that constitute the proposed architecture and distinguish between the notions of *tasks* and *skills*.

### 5.3.1 System Architecture

The system architecture is outlined in Figure 5.1 and is split up into three layers: the application layer, the process layer, and the control layer. In the *application layer* we have a workpiece-centric view, where the user instantiates a task by selecting a desired template from the task library and annotating the CAD model in accordance to the task specification. The task library consists of predefined assembly tasks, each with its own task specification. After a task instance is created, it is passed on to the *process layer*. In the process layer, assembly is considered from the point of view of the workcell. The planning and inference module ensures appropriate assembly sequence selection and composition of correct skills, as well as generating both the skills' application

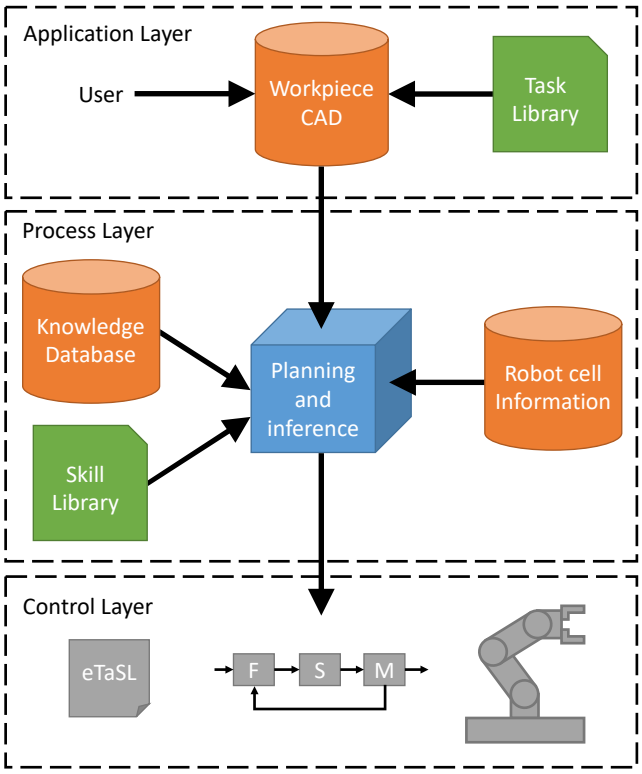



Figure 5.1: The overall system architecture of the proposed system

and process parameters. The application parameters can be extracted from the task instance's CAD model, e.g., the assembly direction or the insertion depth. Process parameters may relate to the CAD data but without a clear underlying model. They may be empirical or have a range of values that produce acceptable results, e.g., insertion force or amplitude of the anti-jamming dither. Furthermore, robot cell information such as safety boundary and workspace limit can be taken into account so as to ensure task execution while respecting the workcell condition. For example, collision with the assembly workbench can be avoided by composing an inequality constraint to the end-effector pose such that it always operates above the workbench's surface. Finally, the skill instances that ground the assembly tasks are executed in the *control layer*. Transition between the skills are handled with a finite state machine.

### 5.3.2 Skill and Task

The terms skill and task are sometimes used interchangeably in the robotics literature. In this work, the terms refer to two different concepts. This follows the nomenclature presented in the previous work [10] with a slight alteration from the increased CAD integration presented in this chapter.

A *task* is defined as a piece of work to be undertaken, while a *skill* is the robot's particular ability. An assembly task is described with a task specification which is composed of CAD constraints between two different parts. These constraints describe the desired end condition of the assembly. They conform to the mating constraints typically found in most well known CAD software. Examples of the supported assembly constraints are: *concentric*, *coincident* and *parallel*. An example of a task specification for a cylindrical object insertion is provided in Listing 5.1. It specifies that the two cylindrical surfaces of the object and the hole should be concentric and the bottom planes of the two should be coincident.

```
Insert (Part1, Part2)
{
  CylinderCylinderConcentric(Part1.CylSurface, Part2.CylSurface)
  PlanePlaneCoincident(Part1.BottomPlane, Part2.BottomPlane)
}
```

Listing 5.1: Cylindrical object insert specification

A skill is defined as an eTaSL specification that has: configuration, inputs, outputs, and events description. The configuration is a set of application and process parameters that are constant during the skill execution. The inputs bring information from continuous sources such as sensors. The outputs are mainly used for logging or analysis purposes. Monitors can trigger events that denote success or failure of the skill.

Since eTaSL is a constraint-based programming paradigm, a skill is composed of eTaSL constraints that are imposed on robot expressions such as joint positions or the end-effector pose. The eTaSL constraints that are used in our skills can be categorized as either kinematic-based or wrench-based. A kinematic-based constraint is used to control a given expression in position or velocity-level. A wrench-based constraint imposes certain force and torque values.

As Figure 5.1 shows, in our proposed system, the possible tasks and skills are predefined in the task and skill libraries, respectively. The task library can be extended by the user by simply defining a new composition of CAD constraints. Meanwhile, extending the skill library is delegated to the *skill developers* who are typically robot experts, thus promoting a separation of roles.

## 5.4 Grounding From Task to Skill

A CAD-level assembly specification must be refined to executable skills while respecting context-dependent constraints. Generic knowledge and rules to perform the refinement is stored in an ontology while the planning and inference module is responsible for selecting the most suitable skill and parameterizing it such that the appropriate robot motion is produced. The following subsections describe in more details about the theory behind these two aspects.

### 5.4.1 CAD-Based Ontology: From Task to Skill

In order to ground an underspecified assembly task to executable skills, context-dependent information needs to be taken into account. One approach to this problem is to have the robot learn the necessary skills [71] [51]. In this work, we favor a knowledge-based approach where a general model of different entities involved in the task is stored in a knowledge database. An advantage of such an approach is the elimination of data-intensive and time-consuming learning processes, thanks to the partial model of the real world encoded in the knowledge database. A widely used knowledge representation in the state-of-the-art literature [129] is the relational knowledge model, namely *ontology*.

Our ontology is based on the CAD semantics ontology proposed by Perzylo et al. in [99], extended with support for force-based assembly tasks. One of the main roles of the ontology is to store a relational model of workpieces. In this work, it is not necessary to provide a complete CAD model of the workpiece. Instead, it is sufficient to provide the CAD model of the workpiece's relevant assembly *features*. Such a paradigm allows more flexibility in modelling a workpiece because only features that are relevant to the task need to have the corresponding CAD model. This approach of workpiece modeling is illustrated in Figure 5.2. A class is denoted with italic letters with the corresponding instance written below it. The arrows between classes denote the relations as they are defined in the ontology. The figure shows that a workpiece instance named `rotor` has two relevant features: `shaft` and `body`. Both features are approximated by BREP *TopologicalPrimitive* instances. A topological primitive is composed of geometric primitives, each with a link to the underlying CAD model. Classes such as *Plane* and *Cylinder* are subclasses of geometric primitive. The inference module can later query various numerical data from the geometric primitives such as the radius of a *Cylinder* instance. Note that although the workpiece `rotor` may have other features, they can be omitted if they are not relevant to the task specification.

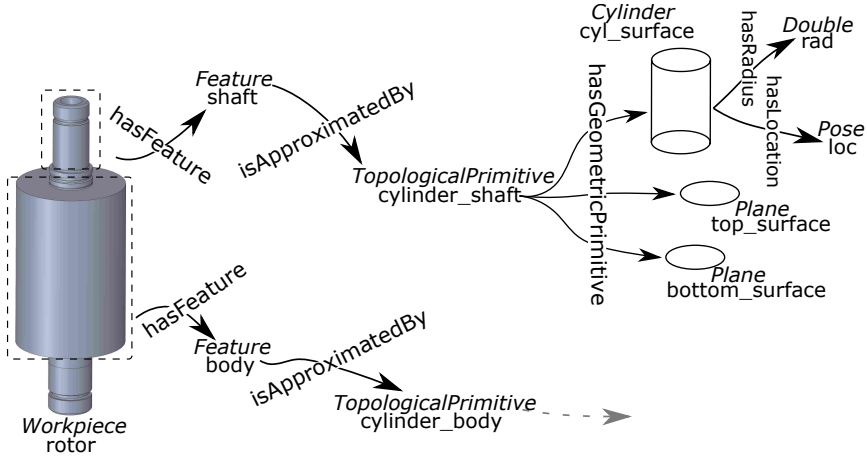



Figure 5.2: The relational model describing a workpiece

Apart from modelling the workpiece, the ontology also models the relation between tasks and skills, as illustrated in Figure 5.3. An assembly task is instantiated from the class *Task*. A task instance is composed of one or more *CADConstraint* instances. CADConstraint is a class that represents an assembly constraint between two geometric primitives. It can be further specialized into subclasses such as *ConcentricConstraint* and *CoincidentConstraint*. On the other hand, a skill is instantiated from a *Skill* class and is composed of a number of *SkillConstraint* instances. These constraints are implemented in eTaSL and are imposed on robot controlled variables such as joint position or end-effector frame. For example the *VectorVectorCollinearConstraint* can be used to align the $z$-axis of the robot's end-effector frame to a target vector such as the axis of a cylindrical feature on a part.

A CADConstraint instance is related to SkillConstraint through the *groundedIn* property. This means that for a given assembly constraint defined in the workpiece perspective, there is at least one skill constraint defined in the robot perspective to execute it. By matching the CADConstraints comprising a task to the required SkillConstraints, the most suitable skill candidates can be inferred.

As indicated before, knowing which SkillConstraints to be used for achieving a given CADConstraint is crucial in the skill selection problem. For this purpose, the ontology covers an extensive knowledge of different mappings between these two classes. For tasks that only require position control, such as insertion with large enough clearance, *KinematicConstraints* should be used. Meanwhile,

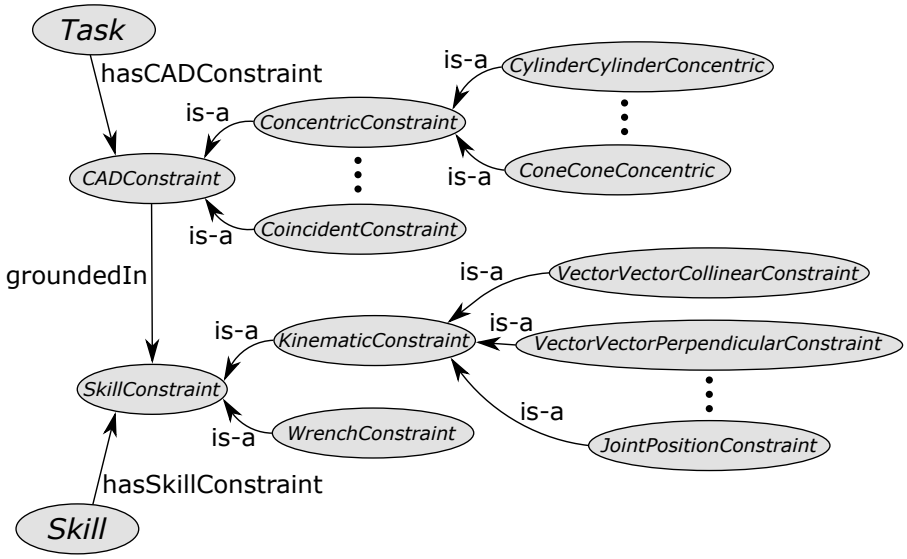


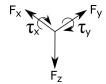
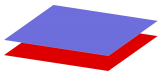
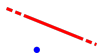

Figure 5.3: A sub-ontology modeling the relation between task and skill

fine manipulation tasks require force-controlled skills, i.e., skills composed of *WrenchConstraints*. We list some examples of these mappings in Table 5.1. The complete mappings can be found in the ontology which is publicly available [94]. The table shows that, depending on the clearance between the two cylinders, the CylinderCylinderConcentric constraint can be realized either with a KinematicConstraint or a set of WrenchConstraints. For some subclasses of CADConstraint, however, it only makes sense to execute one class of skill constraint regardless of the context. The PlanePlaneCoincident, for example, can only be executed with wrench control since the robot's position control accuracy is not sufficient to ensure contact between two planes can be robustly made. Another example is the LinePointDistance which inherently does not involve any contact, hence only KinematicConstraint is suitable. In addition, the table also shows which WrenchConstraint directions are relevant. For instance, the CylinderCylinderConcentric requires force constraints along the $x$, $y$ and $z$ directions as well as torque constraints about the $x$ and $y$ directions.

### 5.4.2 Planning and Inference Module

The planning and inference module has three main purposes: plan the assembly sequence, select the skills based on the task, workpiece, and workcell model,

Table 5.1: Examples of grounding maps between CAD constraints to the required skill constraints

| CAD Constraint | Skill Constraints | |
|---|---|---|
| | KinematicConstraint | WrenchConstraint |
| CylinderCylinderConcentric  | 1 VectorVectorCollinearConstraint  | 3 ForceConstraint and 2 TorqueConstraint   |
| PlanePlaneCoincident  | n/a | 1 ForceConstraint and 2 TorqueConstraint   |
| PlanePlaneParallel  | 1 ForceConstraint and 2 TorqueConstraint   | 1 VectorVectorParallelConstraint  |
| LinePointDistance  | 1 PointVectorDistanceConstraint  | n/a |

and generate the appropriate skill parameters. Optimal planning of assembly involves optimal task sequencing, optimal trajectory planning, and selecting the optimal parameter values. These domains are interconnected, but they are assumed separable in the scope of this article.

As the tasks instantiated in the application layer describing the assembly from a workpiece perspective, they do not include information such as where the parts are in the workcell, or where the tools are. The planning and inference module includes sequential composition of motion skills, e.g., move and grip, move to assembly location, which allows to integrate such workcell information. Generating this skill sequence automatically is particularly challenging since it is tightly related to the assembly planning problem. Some of the difficulties are: ensuring that the skill sequence is geometrically valid, i.e., no interpenetration of parts, and respecting context-dependent constraints such as the gripper types or the reachability of the robot. Methods to localize parts in a workcell is not considered in this article, but the architecture aims for easy integration of localization systems.

### Planning Module

Tasks and parts form a *liaison graph*. In liaison graphs, a node represents a part, and an edge represents an assembly situation [25]. A *precedence graph* specifies which edges in the liaison graph should be completed before others. This precedence graph represents all feasible assembly sequences. In [47], Homem De Mello and Sanderson show the relation between precedence graphs and other assembly representations such as And/Or graphs. The precedence graph is created in the application layer. Assembly of geometric features gives a defined assembly direction and virtual disassembly in the CAD software along these lines provides a suggested precedence graph. The precedence graph is passed from the application layer to the planner for generation of the assembly sequence. Moving to grip, changing tool, reorientation, and other workcell-related skills are added to the assembly sequence by the planner. Workcell information such as which parts and tools are available, or whether collaborative workspace skills are required, is an essential part of the planning module. This information must be stored in a robot cell database describing the different setups available. As the underlying control layer is robot-agnostic and the skills are transferable, we view this as a problem to be addressed in the planning module.

### Inference Module

Given the assembly task instance, the inference module facilitates in grounding to the executable eTaSL skill. The role of the module is to infer the following:

- suitable skill candidates from the provided skill library
- the skill's application parameters
- the skill's process parameters

As robotized assembly is a complex problem with uncertainties and disturbances that are difficult to model, not all skills and their parameters can be inferred correctly. For example, non-geometric data such as the material properties, workpiece's weight and inertia are not taken into account in the task specification. Therefore, influence of these factors on the process parameters is not compensated. In this situation, knowledge from the user is still required. With this limitation in mind, we designed the inference module such that the operator's inputs can still be integrated. The interaction between the user and the inference module is illustrated by a flowchart shown in Figure 5.4.

Once the task is instantiated and the workpieces to be assembled are defined, the inference module selects the most suitable skills. This is done by querying the ontology for skills whose eTaSL constraints match the task's CAD constraints according to the mapping described in Section 5.4.1. The implication of this strategy is that there may be more than one skill that can execute a given task. If such a situation occurs, the inference module informs the user to make the final skill selection.

The second purpose of the inference module is to generate application parameters that mainly depend on the geometric data contained in the CAD model. As an example case, consider the force-based assembly insertion shown in Figure 5.5. Here, the relevant application parameters are the assembly direction $dir$ and the insertion depth $h$. The assembly direction is inferred by checking the normal vectors of the bottom plane of the peg and the hole. Since the normal vectors are pointing outward of both the rotor and hole's solid, successful assembly can only be achieved when the two vectors have opposing directions. The assembly direction can be easily inferred by respecting this condition. The insertion depth is necessary for checking the skill status because the skill succeeds not only when the force along the $z$-axis reaches the target value, but also when the peg has been inserted with the correct depth. This way, a false positive error due to jamming can be detected.

The third role of the module is to infer the selected skill's process parameters. This is a difficult problem due to it being influenced by many different variables

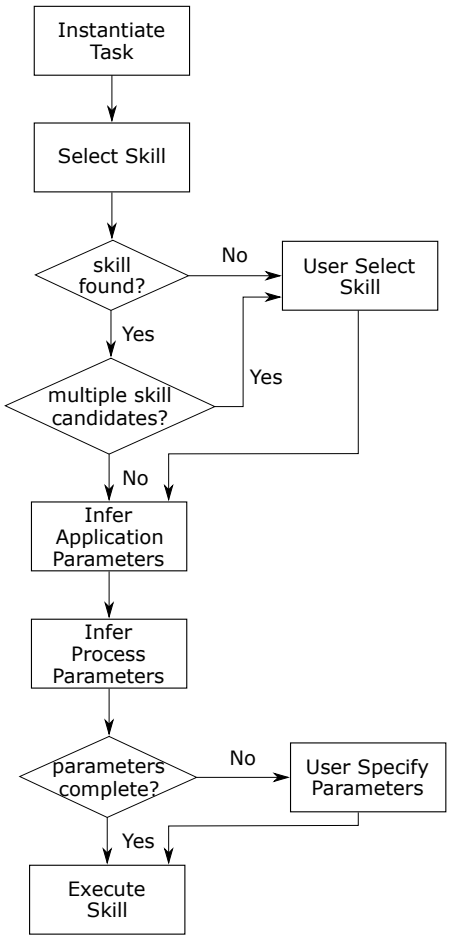



Figure 5.4: Flowchart illustrating the steps in grounding from task to executable skill

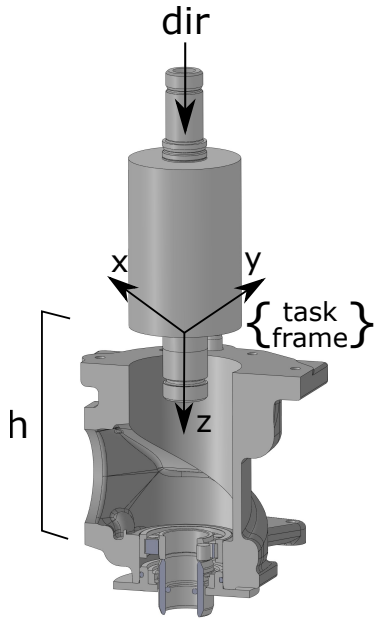



Figure 5.5: Relevant application parameters in a peg-in-hole problem applied to a rotor assembly

where good interaction models are usually not available. At its current state, our inference module can infer the process parameters required for insertion of a cylindrical object in a tight clearance situation. This task would be extremely difficult to perform with only position control, hence the necessity of a force-controlled skill. We generalize the task as a peg-in-hole problem.

A number of process parameters need to be tuned for an optimal peg-in-hole skill. From existing literature [100], it is known that the insertion behavior is determined by a number of factors such as the peg's dimension and the peg-hole clearance. Even for a peg-in-hole assembly with relatively simple geometry, accurate process modeling is difficult to achieve for narrow clearances. Due to hyperstatic contact situations, not all contact forces are always externally observable. We therefore use a data-driven approach to determine the process parameters. To infer the parameters in an uncertain or stochastic situation, a fuzzy inference approach is chosen [43]. Our fuzzy inference module uses the peg's length, peg's diameter and clearance to estimate the appropriate insertion force and dither amplitude.

## 5.5 Implementation

### 5.5.1 Application Layer

The application layer is implemented as a workbench in the open-source parametric CAD program FreeCAD 0.18 [105]. We implemented three task classes in the task library: *Insert*, *Place*, and *Screw*. Each task class is defined by a set of CAD constraints. Once the task is selected, the user annotates the CAD model with the relevant CAD constraints between features of two different parts. These CAD constraint instances together form the task instance. As task instances are defined between only two parts, we can automatically form a liaison diagram for the full assembly, see Figure5.6. To transfer the information of the annotated tasks into the inference module, we implemented a tool for extracting the information as an OWL file. This automatically extracts the part-relative placement of the features, their geometric primitive class, and their geometric parameters such as normal of a plane, or the axis of a cylinder, as well as the constraints and tasks they are related with. The FreeCAD workbench also has tools for annotating relevant part-specific frames such as grasp frame as automatic generation of these is considered outside the scope of this article. These part-specific frames belong to a class called *FeatureFrame* and can be instantiated on a selected vertex, edge, or face. The parts are exported as STL meshes for visualization, and a JSON file describing the feature frames and the attributes of the geometric primitives relative to the mesh origin. The workbench is publicly available [8]. Since assembly sequence planning is not the focus of this work, the assembly sequence was determined as the order in which tasks were annotated.

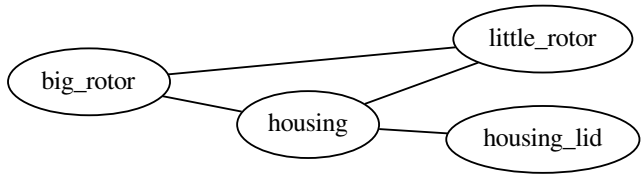



Figure 5.6: Liaison diagram for the first assembly use case. Although the rotors are in contact with the lid, the chamfers allow us to use a place task defined between the housing and the lid.

### 5.5.2 Process Layer

#### Ontology

The ontology is implemented in the web ontology language (OWL) [7], a standard commonly used in robotics research [129] [126] [140]. We created the ontology using the Protégé software tool [87] which enables graph visualization and has a user-friendly interface for managing the ontology. One of the reasons for choosing OWL is because it allows for property restrictions which is important for mapping a CAD constraint to the corresponding skill constraint. We implemented the general knowledge described in Section 5.4.1 in an OWL file.

#### Inference Module

For the skill selection and application parameter inference, a Prolog-based reasoner is implemented. Prolog is a language designed for logic programming and is particularly suitable for the first-order logic required in our application. Our inference module is written in SWI-Prolog [142] which features packages to load OWL files and parse them into Prolog's facts. We implemented a number of inference functions that reason using the stored facts.

As an example of how the inference module resolves the skill selection problem, we queried for the suitable skill candidates for the rotor assembly task illustrated in Figure 5.5. This task can be specified with a cylindrical insertion task given in Listing 5.1. The function call and the returned skills is provided in Listing 5.2 with Prolog-like syntax. The function `get_suitable_skill` takes the task instance `RotorAssemblyTask` as an argument and returns the `Skills` variable. The function infers that there are three suitable skill candidates to execute the task. Apart from the obvious `CylindricalInsert` skill, the `ApplyWrench` skill is suitable because the skill controls forces and torques on all six axes of a given task frame. By applying the correct target values on each force and torque axis, similar behavior to `CylindricalInsert` is obtained. Furthermore, the `NonCylindricalInsert` is also a valid candidate since the skill has the same constraints as `CylindricalInsert` with additional torque control along the $z$-axis. Due to the symmetry of a cylinder, the presence of this extra constraint does not affect the end-goal of the task.

```
get_suitable_skill('RotorAssemblyTask', Skills).
  Skills = 'ApplyWrench';
  Skills = 'CylindricalInsert';
  Skills = 'NonCylindricalInsert'
```

Listing 5.2: Inferring skill candidates for rotor assembly task

In order to infer the appropriate process parameter values for the cylindrical peg-in-hole task, a Mamdani-type fuzzy inference is implemented in MATLAB R2017a's fuzzy logic designer toolbox. We defined in total 21 rules describing the mapping from the peg's dimension and the clearance to the appropriate insertion force and dither amplitude. The fuzzy rules are derived from rough modeling of the contact situation and the empirical trend observed after executing the cylinder insertion skill. Each fuzzy rule is approximated with Gaussian membership functions that categorize a range of parameter values into fuzzy sets. For example, the peg's diameter can be categorized into "*very small*", "*small*", "*medium*", "*large*" and "*very large*". The fuzzy sets for each geometric and process parameters are shown in Figure 5.7 while five sample rules are provided in Listing 5.3.

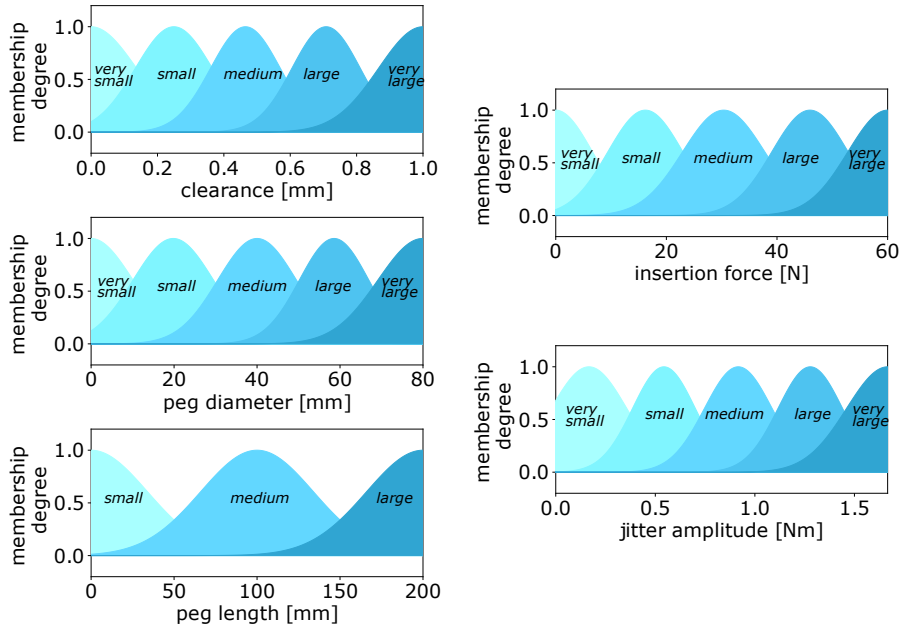



Figure 5.7: The fuzzy sets used in the inference module. The left column shows the geometric parameters while the right one shows the process parameters.

```
if diameter is ''small'' then insertion force is ''large''
if clearance is ''medium'' then insertion force is ''medium''
if length is ''long'' then insertion force is ''small''
if clearance is ''very small'' then dither amplitude is 'very large''
if diameter is ''small'' and length is ''long'' then dither amplitude is ''medium''
```

Listing 5.3: Selection of the fuzzy rules used in the inference module

In this data-driven approach, we first collected the training data by running a total of 40 peg-in-hole trials with different process parameter values. For each run, the insertion force and dither amplitude were carefully selected to ensure successful execution, while minimizing contact force. Based on these experiments, the number of membership functions and their shapes were then tuned. Finally, the fuzzy inference module was validated with a new set of peg-in-hole tasks.

### Implemented Skills

Our skill library consists of around 20 skills, and the skill semantics are stored in the ontology. The ontology organizes the skills according to different taxonomies, e.g., whether a skill belongs to *PositionControlSkill* or *ForceControlSkill* class. Figure 5.8 shows a graph visualization of the skill taxonomies generated by the Protégé tool.

For executing a given assembly task, not only skills to perform the assembly during interaction are needed, but also skills to perform the free-space motion of the pre- and post-assembly action. This includes skills to grip the workpiece and skills to retract the robot once the assembly is finished. These skills are workcell dependent, but common for all workpieces. As assembly planning is not the focus of this article, the inference module presented in this article only deals with selecting and generating the parameters for the actual assembly skill, i.e., the robot motion that achieves the task-specific CAD constraints. However, as pointed out in Section 5.4.2, the intended behavior of the proposed system is to easily integrate existing planning modules such that the pre- and post-assembly skills are composed automatically while taking into account the workcell context. For this reason, in our current implementation, the pre- and post-assembly skills are preselected manually and configured with the parameters obtained from the CAD model and querying the relative transformations between relevant feature frames. For example, to move the gripper to the workpiece location, *MoveCartesianAbsolute* is preselected, while the target pose of the gripper is obtained from the part's feature frame.

To coordinate the skills from gripping, inserting to finally retracting, we used the restricted finite state machine (rFSM) library [60]. Whenever a skill finishes or fails, its monitor will trigger an event message to the rFSM coordinator and the transition to the subsequent skill will be made.

A common approach in robotics is to have states in the FSM denote the motion goals, e.g., *move_to_grip_pose*. This puts the skill complexity in the states. An approach more often found in computer games is to have states denote operational modes such as *walk* and *run*, and then redefine the parameters

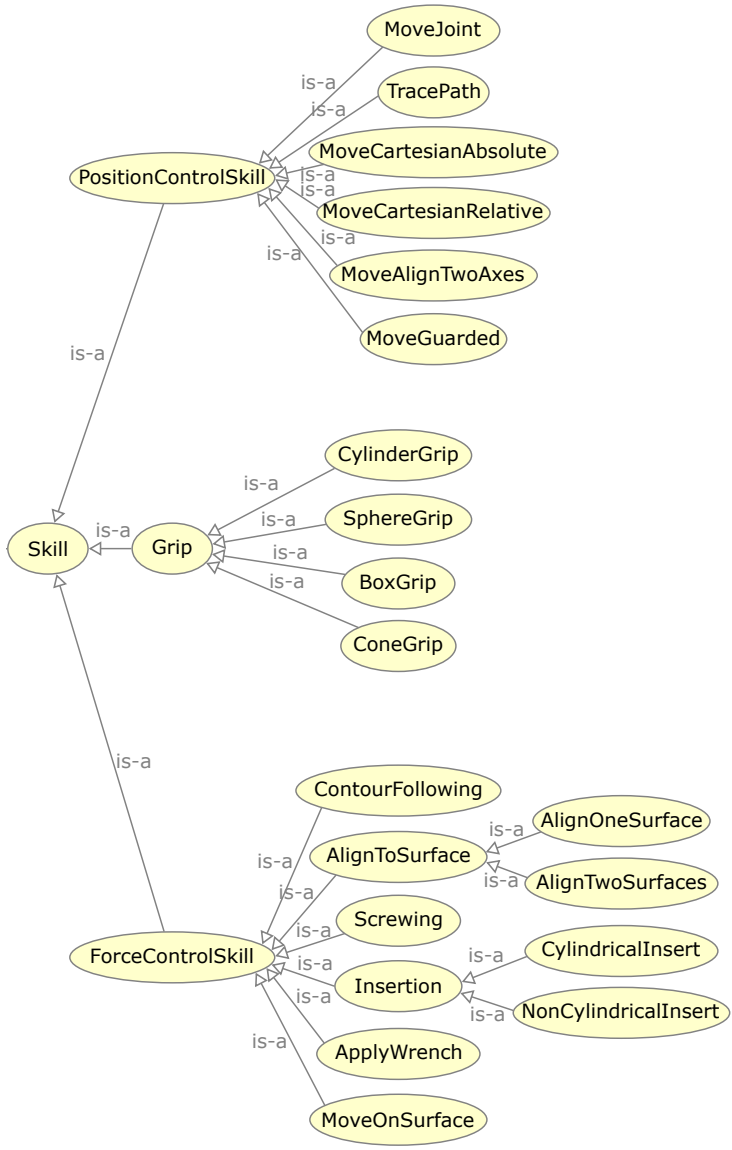



Figure 5.8: The skill taxonomy as defined in the OWL ontology.

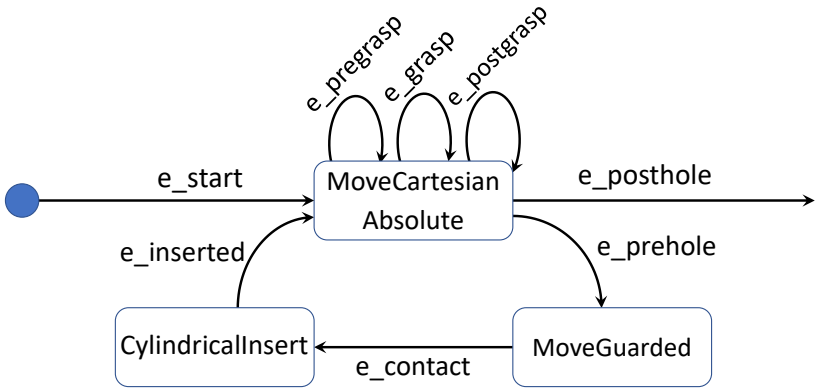



Figure 5.9: Example of the finite state machine for gripping and inserting a workpiece. The transition between two states is triggered by incoming event "e_event_name"

of these modes during runtime. This reduces the states used and puts the complexity in the transitions. We have used the operational modes approach as it is easy to reuse an FSM with new skill parameters, and as transition events denote the momentary situation that the robot is in. Therefore, each rFSM's state corresponds to a skill to be executed, and the state transition reconfigures the skill with the updated application and process parameters. An example of the rFSM to grip and assemble a workpiece is shown in Figure 5.9. When *e_start* occurs, we set the target pose of *MoveCartesianAbsolute* to an offset distance above the workpiece's pick frame. Success in reaching this target pose is associated with the *e_pregrip* event. When *e_pregrip* occurs the target pose of *MoveCartesianAbsolute* is updated to the part's pick frame and *e_grip* event is emitted upon success. When *e_grip* occurs the robot has reached the workpiece's pick frame, engaged the gripper and set the next target pose to a $z$-axis offset from the pick frame, and success gives the *e_postgrip* event. The procedure is repeated for the $z$-axis pre-hole locations. When *e_prehole* occurs we transition into *MoveGuarded* which is a Cartesian motion towards the hole with monitors that trigger when the end-effector force exceeds a threshold. Once contact between the workpiece and hole is made, signaled by the *e_contact* event, the *CylindricalInsert* skill is executed. This procedure of reconfiguring the skill parameters is adhered to for all of the transitions.

### 5.5.3 Control Layer

In this article, we present a more extensive use cases by testing our system on two different robot setups with different communication interfaces. The first setup consists of a seven DOF KUKA LBR iiwa robot equipped with a pneumatic SCHUNK RH9010 parallel gripper. The LBR iiwa is controlled over Fast Robot Interface (FRI) [111] with a control loop frequency of 200 Hz. This robot provides joint torques measurements and estimates of the external end effector wrench (force/moment), which can be used in the execution of force-based skills.

The second setup consists of a KUKA KR16-2, a larger industrial robot with lower control frequency of 83 Hz controlled using the Robot Sensor Interface (RSI). Since the KR16 does not have any torque sensor, we attached a Schunk 6D Force/Torque sensor on the end-effector. Both robots are controlled with eTaSL/eTC [3] which is deployed as an Orocos [121] component. eTaSL is used for constraint specification in Lua language while eTC is the controller implementation that ensures exponential convergence of the constraints in an instantaneously optimal way. Additionally, a *feature_frame_publisher* node [9] was created in ROS [82] for publishing all relevant frames with TF2.

## 5.6 Test Cases

This section describes two test cases that were performed to investigate the prototype architecture. The goal of the test cases is to validate that the implemented prototype works well in specifying the CAD-level assembly task as well as generating the executable skills.

Each test case is carried out according to the following steps. For every part to be assembled, the programmer annotates the relevant features of the part and selects the desired task class though the FreeCAD interface. Once the task instance is created, the most suitable skill class is deduced by the Prolog-based reasoner and the appropriate process parameters are generated by the inference module. The geometry of the parts such as the dimension and the clearance are different so as to verify the effectiveness of the process parameters inference. Once an executable skill is instantiated, an eTaSL controller is set to execute it.

### 5.6.1 Case 1: Assembly of Air Compressor

We consider an assembly of a rotary-screw compressor (see Figure5.10). The compressor is composed of over 30 parts. To limit the scope of this demonstration, we focus on the assembly of the rotors and the lid. The housing is attached to a fixture and all insertions are in the same direction. The compressor has a small and large meshing helical screw rotors. The upper shaft of the rotors go through the lid through two chamfered holes. All parts to be assembled are placed in known poses.

The assembly of the two rotors is done with the *Insert* task while the lid is performed with the *Place* task. The *Insert* task is specified with a set of CAD constraints reported in Listing 5.1. The *Place* task is specified with two *CylinderCylinderConcentric* constraints so as to align the two rotors' shafts with the lid's holes and an *AxisVelocityConstraint* to bring the lid down to the housing with a constant velocity.

Since the two rotors and holes have tight clearance, the task should be executed with force-controlled skills. The inference module correctly deduces the suitable skill candidate to be *CylindricalInsert* skill, infers the assembly direction and insertion length, and generates the process parameters presented in Table 5.2. As for the lid, there is sufficient clearance for the robot to accurately perform the assembly without using force control. Therefore, the task is solved by executing a position control skill *MoveAlignTwoAxes* (see Figure 5.8) with force threshold monitor to detect contact.

The experiment shows that the robot successfully assembles the parts using the skill and parameters inferred by the inference module, see Figure 5.11 for a snapshot of the assembly sequence. The recorded insertion forces and moments during a portion of the assembly sequence are shown in Fig 5.12. The graph gives a recording of the transition from *MoveGuarded* to *CylindricalInsert*.

Table 5.2: Geometric and Inferred Process Parameters for the Compressor Rotor Assemblies

| Parameter Category | Parameter Name | Part Name | |
|---|---|---|---|
| | | *small_rotor* | *big_rotor* |
| Geometric | Length (mm) | 192 | 192 |
| | Diameter (mm) | 61 | 75.5 |
| | Clearance (mm) | 0.1 | 0.08 |
| Process | Insertion Force (N) | -31.1 | -33.5 |
| | Dither Amplitude (Nm) | 1.13 | 1.33 |

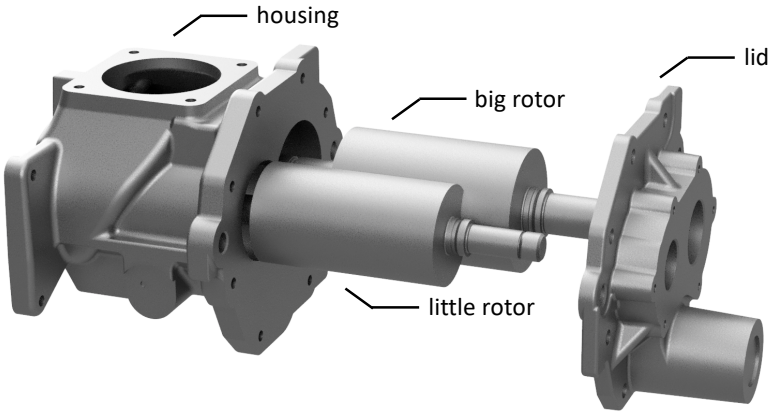



Figure 5.10: CAD model of the air compressor. Note that the helical screw profile is not modeled.

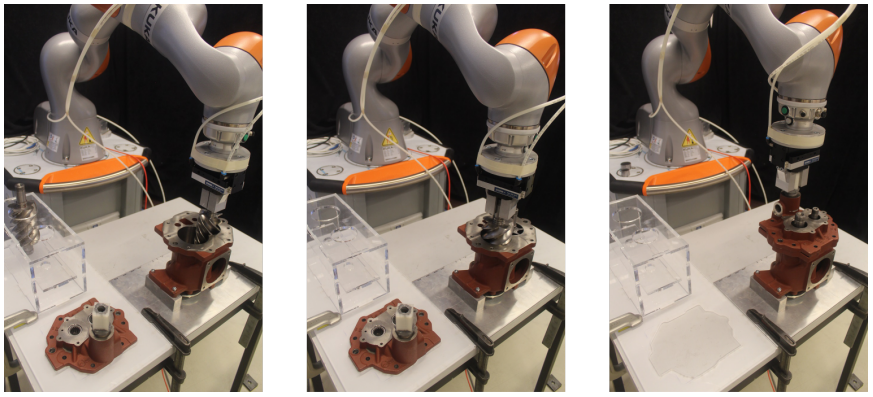


Figure 5.11: The robot performing the assembly sequence of the compressor parts. From left to right: assembly of the small rotor, the big rotor, and the housing lid.

### 5.6.2 Case 2: Assembly of Electrical Motor

In this example we consider the assembly of an electrical motor (see Figure 5.13) by peg-in-hole insertion of the rotor. This is similar to the previous example but showcases the robot-agnostic nature of the control-layer as it is controlling a KUKA KR16-2, illustrated in Figure 5.14, with a high mechanical stiffness and a slower communication interface. The employed skills are the same as for

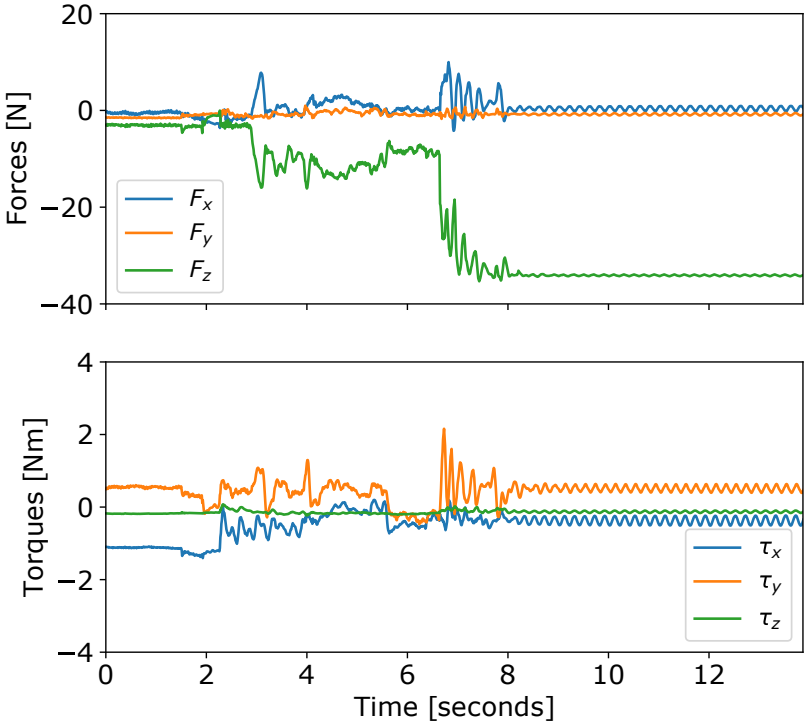



Figure 5.12: The measured forces and torques during insertion of the big rotor. Note that from 8 s, the rotor is in contact with the bottom. This was added to verify that the desired dithering is present.

the rotary screw compressor example, with process parameters generated by the inference module as shown in Table 5.3.

In Figure 5.15 the measured forces at the end-effector are given for the transition from *MoveGuarded* to *CylindricalInsert*, and a moment while it remains in *CylindricalInsert* after the skill is achieved to show the behavior once contact occurs. The gain of the force constraint in assembly direction has been lowered to reduce the overshoot and movement speed of the robot. Monitors on insertion distance, forces, and torques have been removed to observe the continuous behavior of *CylindricalInsert* under the higher mechanical stiffness and slower controller rate. A video showing these two experiments is available online at: https://youtu.be/WkVxxnEgyTU.

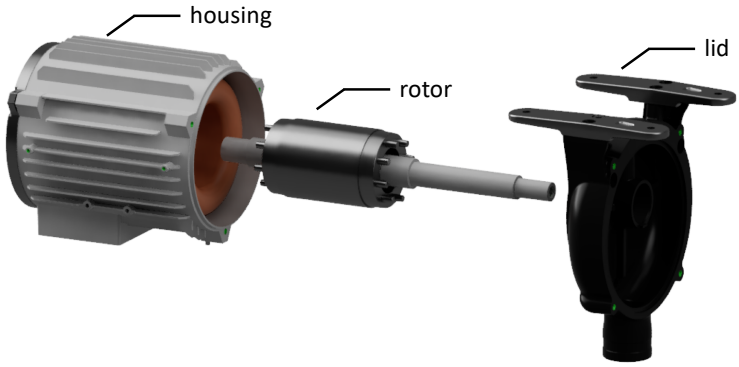



Figure 5.13: CAD model of the electrical motor.

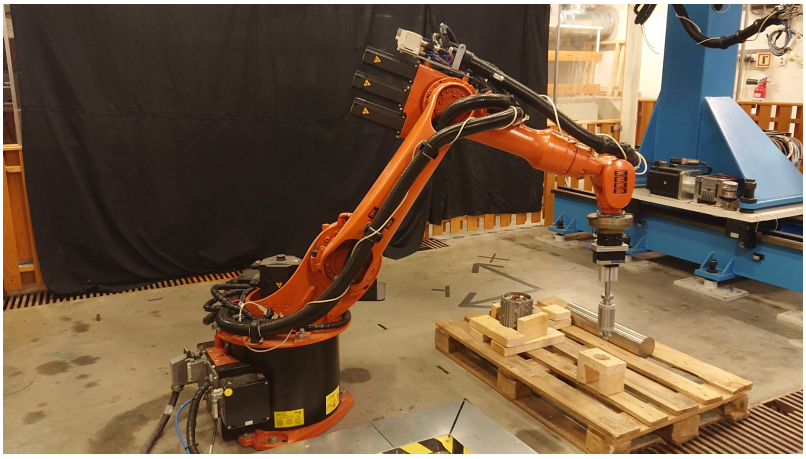


Figure 5.14: KR16-2 performing the assembly sequence of the electrical motor.

## 5.7 Discussion

The force controller in eTaSL has a damping behavior [3]. This resulted in a velocity proportional to the difference between the measured and desired force. For the first use case, Figure 5.12 shows that when the robot approached the rotor's hole between 0 s and 2.8 s the forces were approximately zero. When a force magnitude of -10 N in $F_z$ was experienced by *MoveGuarded*, a transition into *CylindricalInsert* occurred. The insertion process lasted from 2.8 s to 6.7

Table 5.3: Geometric and Inferred Process Parameters for the Motor Assemblies

| Parameter Category | Parameter Name | Part Name *rotor* |
|---|---|---|
| Geometric | Length (mm) | 74 |
| | Diameter (mm) | 64 |
| | Clearance (mm) | 0.5 |
| Process | Insertion Force (N) | -37.4 |
| | Dither Amplitude (Nm) | 0.74 |

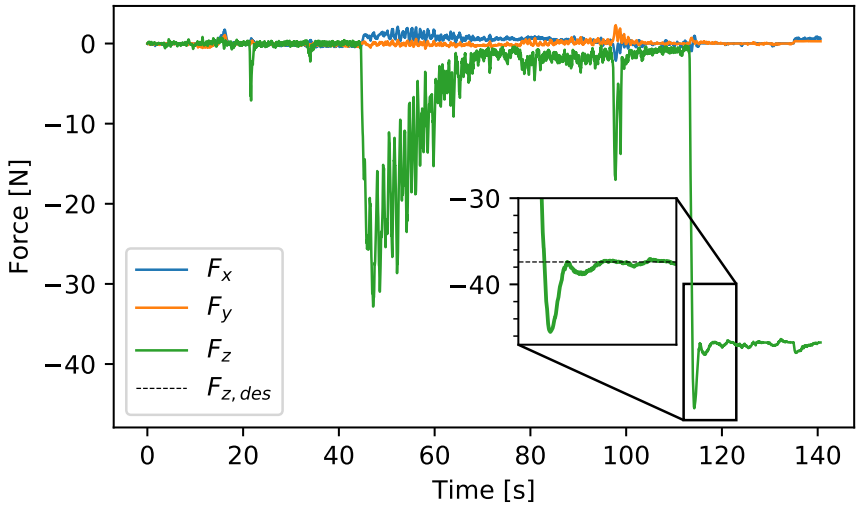



Figure 5.15: The measured forces when inserting the rotor in the motor housing. Note that this example moves slower as it uses a lower gain to show *CylindricalInsert* behavior. The inset axes show the overshoot but eventual convergence to the desired insertion force.

s and the robot was moving down while trying to achieve the target insertion force. The effects of friction (approximately -10 N in $z$-direction) are visible during the insertion. From 6.7 s to 8 s, the rotor touched the bottom of the hole, the velocity dropped to zero, and the set-point for the insertion force reached its desired value. The measured force along the $z$-axis was consistent with the inferred process parameters listed in Table 5.2. Furthermore, the oscillations due to the applied dither-force remained bounded. This shows that the skill

executed the assembly task as desired.

The monitors in eTaSL trigger switching of skills for safer execution. The KR16-2 has a motion buffer that introduces a tracking delay [73]. When a force controlled skill such as *CylindricalInsert* is run, the tracking delay can cause an overshoot from the desired force as seen in Figure 5.15, but the system converges to the desired force in the end. In the test presented in the figure, the gain of the compliance constraint has been lowered so as to lower the movement speed of the robot, and monitors to indicate successful execution of the skill have been removed. When the monitors are present, the skill will switch at first available sensor signal indicating failure or success condition. This ensures a safer execution of force-controlled skills.

In this chapter, we have emphasized the geometric information available in the CAD data, but there is another aspect that is of relevance, the material properties of the parts involved. For example, insertion of a teflon peg in a steel hole has different process parameters to inserting a steel peg in a soft-plastic hole. Material properties are available in STEP AP214 file format, and many CAD programs support it. For multi-material assembly scenarios, the inference module should be trained on the material properties as well as the geometric properties. The newer STEP AP242 format includes methods for annotating physical surface characteristics of features as well which can further improve the inference engine [141].

Although this chapter does not quantify the benefit of the proposed architecture, such as how much the programming time is reduced, the preliminary evaluation shows promising benefits for potential users. The current industry practice for programming assembly tasks is a robot-centric approach. Not only does this make the program often not reusable, the level of robotics expertise needed is considerably high. Programmers have to write low-level code such as implementing the impedance controller for force-sensitive tasks. The architecture proposed in this thesis alleviates this problem by facilitating task-centric specifications that liberate users from low-level coding. Therefore, users such as application deployers who are experts in the assembly process can specify the task in the abstraction level that they are familiar with. Furthermore, the use cases across different robot models with the same skill library suggests better reusability than the current industry practice.

## 5.8 Conclusion

In this work, we extend the system architecture proposed in our prior research [10] by presenting an improved methodology to automatically select a suitable

skill and its corresponding parameters to execute a given assembly task. This improvement is made possible by modeling the semantics of relevant classes involved in the assembly context such as geometric primitives, tasks and skills, as well as building an ontology to model their relations. The architecture is split into three layers. The application layer allows the user to annotate tasks according to a task specification using CAD constraints. The process layer infers appropriate skills and generates both application and process parameters using a Prolog-based reasoner and fuzzy-inference. Finally, the control layer executes the task in the eTaSL framework with an rFSM-based state machine to coordinate the skill transitions. The system architecture is designed such that the task is executed in a robot-agnostic manner.

The proposed system is validated on challenging assembly tasks that require force-control. The inference module is shown to correctly select the suitable force-based skills to successfully execute tight-tolerance tasks. Apart from the geometry-related application parameters, the selected skills require process parameters such as insertion force and dither amplitude. These empirical parameters are not readily available in the CAD model. Therefore, a fuzzy inference method is employed to estimate these parameters based on a set of key geometric parameters from the CAD model.

The experimental validation is conducted on two different assembly problems performed by two different robot setups: assembly of an air compressor and an electric motor. In both cases, the rotor assemblies can be generalized as a peg-in-hole task, for which the conventional solutions have been studied well [74]. The key geometric parameters used in the fuzzy inference were the peg's clearance, peg's diameter, and insertion length, all of which were extracted from the CAD model. For more complex assembly tasks, such as snap-fit, the key geometric parameters are yet to be defined.

A limitation of the fuzzy inference method is that it fails to extrapolate outside the predefined fuzzy set range, hence the dataset should vary enough to cover the given workpiece geometry. Furthermore, finding the relevant geometric parameters for a given assembly task can be challenging. However, with a sufficiently large dataset and more advanced regression methods, such as automatic relevance detection [14], we argue that it is possible to automatically deduce which of the parameters are really relevant for the process parameters. Such an approach will be beneficial for including material properties when inferring process parameters. The fuzzy set database can grow over time to accommodate a larger variety of assembly cases and this database can be shared with similar robot setups.

The software implementation serves as a proof-of-concept that validates the intended goal of the proposed system architecture. The assembly of the rotors

and housing lid was successfully executed by the system, and the inference module provided reasonable parameters. The application layer implementation is used for instantiating assembly tasks through annotation of the CAD model's geometric features and the publisher node gives the relevant feature frames transformation in ROS.

# Chapter 6

# Discussion

This chapter discusses the connections between the contributions and their distinct roles in robot skill programming and deployment. Furthermore, this chapter also highlights some of the insights acquired during the development of the proposed solutions. At the end, we explain the main limitations of this thesis contributions.

## 6.1 Connections between the contributions

As explained in the introduction chapter, each contribution gives a solution to achieve the thesis' objectives. These contributions can be seen as complementary components that form a more comprehensive solution. We illustrate their connections in Figure 6.1. The diagram shows different components that are involved in the programming and deployment of constraint-based skills. Note that the diagram is mainly intended for highlighting the connection between the components, therefore omitting some details such as the feedback from the robot to the controller. Here, we reintroduce the two stakeholders mentioned in Chapter 3, namely *skill developers* and *application developers*. The former are experts in robotics whose job is to implement robot skills, meanwhile the latter are the primary users of the skills. The application developers have only limited knowledge on robotics and control, but they are the domain experts in the specific tasks to be automated.

The skill modelling from Contribution 1 provides a methodology for skill developers to create new skills using the constraint-based task specification

approach. This thesis envisions a collection of the created skills in a library that allows users to inspect and reuse them. This is inspired from the "apps store" concept that is common in today's smartphones. The skills are encapsulated in templates that are easy to comprehend by application developers with adequate robotics background, therefore allowing them to compose, instantiate, and parameterize the skills by hand. Furthermore, for tasks that require concurrent behaviors, application developers may apply the composition patterns provided by Contribution 1.

Contribution 2 and Contribution 3 allow application developers who lack robotics background to specify tasks in the higher abstraction layers. Contribution 2 facilitates task specification in the symbolic level through PDDL. This way of programming is suitable for application developers who come from the symbolic planning background such that they can focus on the domain and problem specifications while abstracting out the lower level details, e.g., skill and monitor implementations. Here, the application developers do not manually select and instantiate the skills by themselves. Instead, the skill library is accessed by a refinement module as it maps the symbolic actions to the corresponding skills. As of the current implementation presented in Chapter 4, this action-to-skill mapping is assumed to be already available and is not part of this thesis contribution.

Contribution 3 is intended to support application developers whose background is from product design where their main tool is CAD/CAM software. It facilitates skill deployments through CAD-level task specifications. Similar to Contrbution 2, the skill library is not directly accessed by the application developers, but instead by an inference module presented in Chapter 5. This inference module leverages the CAD models in order to select the appropriate skills and generate the suitable parameters.

A potential connection that is not yet explored in this thesis is to leverage the inference module from Contribution 3 to support the refinement process from symbolic actions to the corresponding skills when the CAD models are available. This connection will replace the static mapping which is currently used with a dynamic refinement that takes into account the model of the workpieces being manipulated.

## 6.2 Conflict resolution of robotic tasks

As the applications become increasingly complex, conflicting tasks becomes a recurring phenomenon in robotics. This phenomenon appears accross different levels of abstraction. At the low-level control, as seen in Chapter 3 and 4, robots

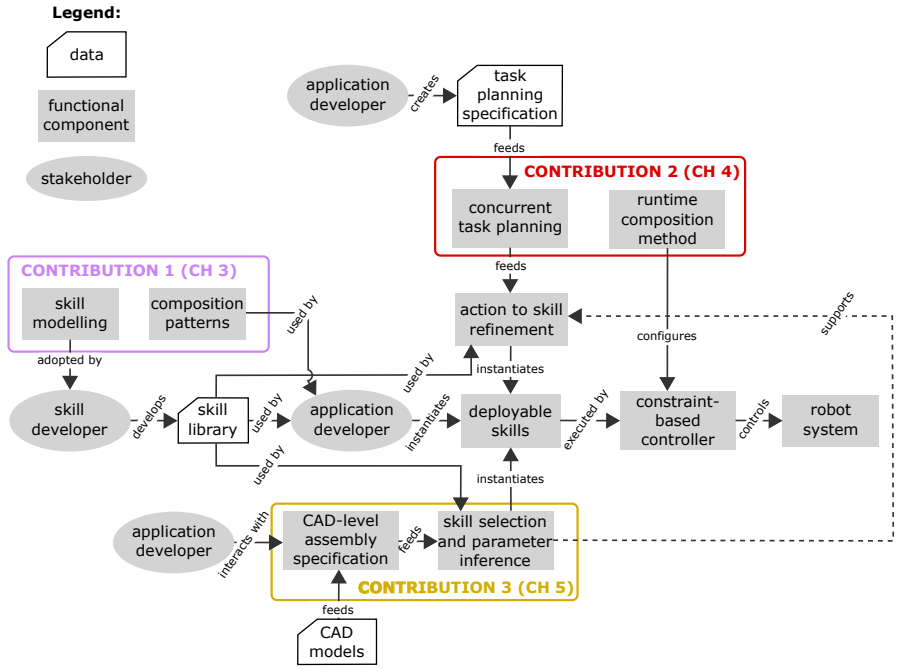



Figure 6.1: Connections between this thesis contributions and other relevant parts shown in a broader view of robot skill programming and deployment. The contributions and their corresponding chapters are shown in rounded rectangles. The solid arrows signify how two entities are related. The dashed arrow signifies a potential connection that was not explored in this thesis.

may execute incompatible tasks simultaneously and try to minimize the task errors. In the high-level deliberation, a mobile robot may have to decide on either continuing towards its destination or diverting to the nearest charging station given its limited energy. In any case, such a conflict may sacrifice the execution qualities of one or more tasks.

Naturally, the main reason for conflicting tasks is the limited resources. These resources correspond to different aspects of a robot system from actuation and sensing to computation. While this thesis does not deal with the problem of allocating sensing and computational resources, the adopted constraint-based task specification approach nicely fits the actuation allocation problem, in particular the low-level control of robot joints.

To a lesser degree, a capability to handle conflicting tasks can be achieved

with an offline motion planning approach. The classical graph-based and sampling-based planning can already take into account conflicting tasks such as reaching a desired pose and avoiding collision [88, 55, 69]. A more recent development on motion planning that uses an optimal control formulation can generate both a collision-free path and the corresponding control input [112, 80]. The optimization-based motion planning offers a higher flexibility to define a desired objective function such as minimizing time or energy as well as including relevant constraints such as the robot dynamics. Nevertheless, the limitation of motion planning to incorporate sensor signals and its slow computation time make it unsuitable for resolving conflicting tasks in real-time. Furthermore, motion planning also lacks the capability to accomodate tasks other than target-reaching.

The constraint-based task specification and control adopted in this thesis provides an elegant way to address conflicting tasks in real-time at the continuous level. Since tasks can be formulated as a constrained optimization problem, handling conflicts is reflected in the constraints specification. As presented in Chapter 2, constraints can be assigned with weights and priorities. It is then left to the solver to compute the optimal control input that respects the configured constraints. While predicting how exactly the robots will behave during conflicting tasks can be difficult, the skill model presented in Chapter 3 alleviates this problem by capturing the deviation behaviors of skills.

Different perspectives exist regarding the handling of conflicting tasks using a constraint-based task specification and control approach. One perspective favors the use of weights, as in the case of iTaSC [113]. The use of weights can be handy in achieving a smooth blending of behaviors such as a composed skill that combines Cartesian motion and impedance behaviors, as illustrated in Section 3.5. However, as motivated in Chapter 2, obtaining proper weight values may be difficult in some scenarios. Another perspective favors the use of priority levels, as in the case of SoT [75, 77]. While the use of priority levels avoids parameter tuning in relative importances between tasks, it can be impractical as the number of tasks increases. This is mainly due to the recursive nature of priority levels that requires more computation cost.

This thesis' standpoint is to make use of a good balance of both weights and priority levels, arguing that, ideally, the number of priority levels should be kept to a minimum in order to reduce computation cost. For simple tasks, two levels of priority is usually sufficient: a higher priority level (hard constraints) for the safety aspect and a lower priority level (soft constraints) for the other aspects. However, in some cases, using priority levels can cut down the parameter tuning effort, as illustrated by the example in Section 2.3. For eTaSL's HQP solver, we find that, empirically, the maximum number of priority levels can be kept to around four to five (including the top priority level for safety).

## 6.3 Coordinated execution of concurrent robot tasks

In some applications it is required to perform concurrent tasks in a coordinated manner. At the continuous level, [81] shows a human-robots handover task where two robots have to synchronously reach an object handed by an operator. In [16], a robot has to coordinate a reaching and a gripper opening skill in order to open a drawer successfully . These examples illustrate cases where concurrent tasks are not executed independently, but according to specified coordination rules.

The need for coordinated tasks may also arise at the discrete level. For instance, consider two robots that cooperate to assemble screws into a motor. The first robot is given the task of "*grasp the stator and position the various screw holes facing towards the second robot*", while the second robot is given the task "*pick the screws and fasten them in their corresponding holes*". Here, the two sequences should be executed while respecting several conditions. For example, the screw insertion can only start once the stator is well-positioned. Such a problem requires a coordination at the discrete level that takes into account the interdependencies of the tasks and, possibly, aslo temporal conditions.

Automatic computation of discrete-level task coordination is typically formulated as a scheduling problem. When the duration of the tasks are known, several established approaches such as temporal planning and pure scheduling are suitable [84]. The temporal planning approach is used to compute a plan that consists of concurrent actions in [23]. As for the pure scheduling approach, methods such as *simple temporal networks* (STN) [27] and *mixed integer programming* (MIP) [101] are well-studied. In [89], the authors use STN to compute feasible task sequences in a cooperative multi-robot scenario. Meanwhile, in [62], MIP is applied to solve scheduling problems of a multi-robot system involving precedence and synchronization constraints.

Although concurrent task coordination is not the main subject of this thesis, the presented methods touch upon this subject at both continuous and discrete level in a limited scope. In Chapter 3, we show how, by formulating a skill progress as function of feature variables and applying the composition patterns, we can implement two modes of continuous-level skill synchronization. The proposed composition patterns allow to automatically adapt the skills' progress in the presence of disturbances.

In Chapter 4, we demonstrate a discrete-level task coordination using task planning for dual-arm assembly cases. Specifically, the task planner is based on a temporal planning approach that supports specification of temporal conditions.

This not only allows us to generate a partial order plan for the overall task, but also to enforce concurrent actions that react to disturbances.

Reflecting on the contributions of this thesis and the existing work in the literature, we obtain the following two insights:

### Improving the reactivity of a discrete-level plan execution

The existing discrete-level schedulers mostly assume task executions in nominal conditions and known task durations, with the exception of a few contributions [29, 62]. However, this assumption can be too conservative for real-world applications. The constraint-based task specification and control approach can potentially relax this assumption. It enables reactivity to disturbances at the continuous level while preserving the plan execution. Furthermore, task synchronizations that are previously only handled at the discrete-level can be more seamlessly enforced at the continuous level too. For example, consider two tasks that have to end at the same time, but are subject to varying durations. The existing schedulers typically synchronize both tasks through discrete-level commands such as *start*, *stop* or *resume*. While this may be acceptable for a task in which disturbances rarely occur, it can be unsuitable for a task that is subject to disturbances that are more frequent and dynamic. In such a case, the synchronization behavior should also be reactive. The constraint-based task specification and control approach facilitates this reactive synchronization within a common framework, i.e., without the need for a separate controller. It is worth mentioning that this idea was first explored by Scioni et al., in [115]. The authors propose a scheduler that bridges symbolic plans to lower-level constraint-based skill execution. However, their work does not yet formulate task synchronization as a reactive behavior.

### Compatibility of task planning with constraint-based skills

Among existing discrete-level scheduling methods, this thesis views task planning as an approach with a closer affinity to the constraint-based task specification and control. This in turn helps ease their integration. There are several similarities between the two approaches. First, similar to the constraint-based task specification, task planning facilitates a *declarative* specification, through, for example, PDDL and STRIPS. This allows the solver to compute the optimal policy without describing the necessary steps to achieve it. Second, task planning allows to separate the domain and problem specifications. This is analogous to the constraint-based task specification approach in which the environment and robot specifications can be separated from the actual task

specifications. This enables the reuse of specifications e.g., using the same domain/environment/robot for different problems/tasks, and vice versa. Finally, task planning specifications consist of entities that are intuitive to refine into the skill-related entities presented in Chapter 3 and 4. For example, actions are intuitively refined into skills and the pre- and post-conditions can be refined into monitors.

## 6.4 Limitations of the proposed methods

### 6.4.1 Limitation of reactive planning

The constraint-based task specification and control approach is typically formulated as an instantaneous problem (the horizon corresponding to one time step), therefore allowing fast computation, enabling real time reactive behaviors. One of the main limitations of the constraint-based task specification and control approach is that the solution is only locally optimal. This means for tasks that require navigating around obstacles the robot may get stuck in an undesirable configuration. One approach to alleviate this is to lengthen the horizon in which the optimization is performed. This requires a formulation in the form of MPC. One example is a nonlinear MPC-based formulation that is applied for free-space motions with obstacle avoidance [108].

Naturally, a better suited approach to overcome the above problem is a path/motion planning approach. However, motion planning alone may not be sufficient if the task is subject to runtime disturbances. In such a case, it makes sense to combine the two approaches such that the path computed from the motion planner is used as a nominal trajectory for the contraint-based controller. This way, the robot can still react by e.g., deviating from the paths when local changes in the environment are present. While the idea of combining motion planning and reactive control was already endorsed decades ago [102, 18], the integration with a more modern constraint-based task specification and control framework such as eTaSL is still relatively unexplored.

### 6.4.2 Limitation of the runtime replanning approach

Contribution 2 presents a framework to deal with sudden disturbances by replanning and composing skills during runtime. This approach allows robot behaviors to be adapted flexibly online without pre-programming in advance. However, in general, task planners cannot guarantee finding a solution reliably within a certain timing requirement. Therefore, the thesis' approach is more

suited for applications whose safety margin can still tolerate the computation time of the task planner. For tasks where the safety margin is narrow and planning time could pose a risk to the robot and human, the approach may no longer be a viable solution. In such a case, the task planner should not be the only entity that has the privilege to compose the skills. Rather, a safety supervisor should be able to immediately intervene and execute the appropriate safety skills.

### 6.4.3 Limitation of CAD-level assembly specifications

The application layer of the system architecture from Contribution 3 allows programmers to specify assembly tasks by composing CAD-level constraints between two CAD features such as imposing concentric constraints between the cylindrical surfaces of two parts. One drawback of such a specification is that it only informs the desired end state of the assembly. With this, we can only derive the skill that performs the insertion phase, i.e., bringing the part from the pre-insertion state to the desired end state. However, it does not inform how the robot should perform the pre-insertion phase.

This thesis assumes that the pre-insertion phase can be performed with simple policies, such as approaching the workpiece with a downward motion. This strategy works relatively well for peg-in-hole tasks such as the compressor and motor assemblies as presented in Chapter 5. The uncertainties of the hole location can be resolved by performing searching motions such as a Lissajous pattern. Unfortunately, for more intricate assembly tasks, this strategy is no longer suitable. To illustrate this problem, consider a home switch assembly task shown in Figure 6.2. In practice, a human assembles the switch by first "hooking" the top (button) part to the bottom (base) part, forming a "hinge" joint, then, constrained by this joint, applying a downward torque to snap fit the button to the base. If we translate this problem to a CAD-level assembly specification, the desired end state is to have two pairs of concentric cylindrical surfaces, i.e., $\{f_{1a}, f_{1b}\}$ and $\{f_{2a}, f_{2b}\}$ and two coincident planar surfaces, i.e., $\{f_{3a}, f_{3b}\}$. Clearly, such specification is not sufficient to compute an appropriate pre-insertion policy. The "hooked" state is not achieveable by a simple linear/downward motion and the snap fit can only be performed reliably once the cylindrical surfaces are concentric. Nevertheless, the solution provided by Contribution 3 is still useful in generating the appropriate skills for the actual insertion phase.

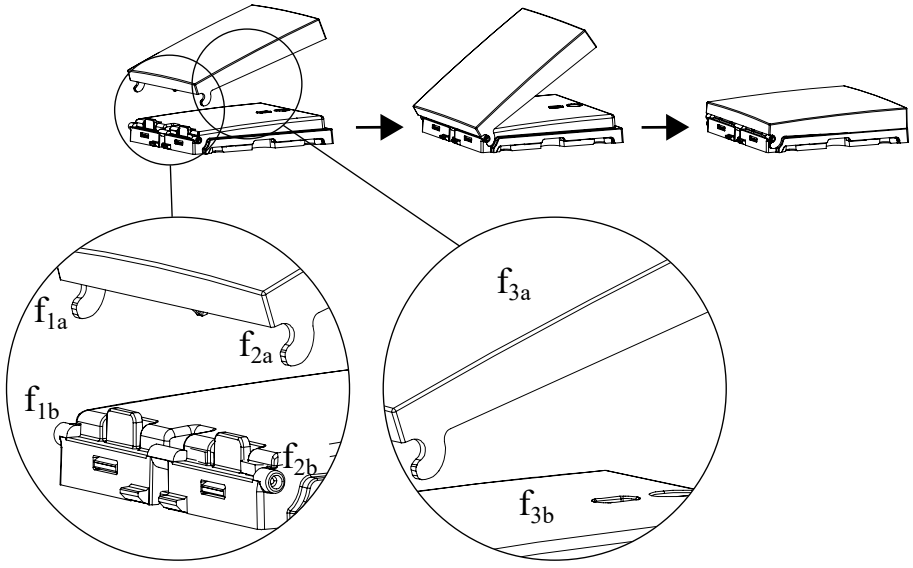



Figure 6.2: Illustrating the limitation of the presented CAD-level assembly specification with a home switch assembly task. Top, left to right: the assembly steps as performed by a human. Bottom: magnification of the switch to show the feature pairs that are subject to CAD-level constraints.

# Chapter 7

# Conclusion

This chapter revisits the thesis objectives, summarizes the contributions and provides suggestions for future work. The main goal of the thesis was to reduce the commissioning effort and improve the robustness of robotic applications in less structured, dynamic environments where geometric uncertainties and unpredictable disturbances are present. The research objectives that stem from this goal are recapitulated as follows:

1. to improve the robustness of skill execution and extend skill composability. This objective is motivated by the increasingly complex tasks that today and future robots are expected to perform. To cope with these tasks, robots would need reactive behaviors with respect to sensor input as well as the ability to combine multiple behaviors to handle a larger task variety. For these purposes, the classical robot programming paradigms, which are largely based on specifying desired trajectories or configurations/poses, are no longer viable.

2. to endow robots with discrete deliberation behaviors to deal with a high variability in the environment and disturbances that cannot be handled at the continuous level. This objective is motivated by task execution in the presence of sudden disturbances which require adaptation of the ongoing robot behaviors. In such a scenario, robots can no longer rely solely on reactive behaviors at the continuous level, but need a discrete-level deliberative capability for composing new behaviors during runtime.

3. to reduce the level of expertise required in programming robot skills. This objective is motivated by the high level of expertise needed in specifying

robotic tasks and the untapped potential of CAD information and CAD-level task representation. By reducing the required level of expertise, robot programming becomes more accessible to non experts, therefore lowering the barrier of robot adoption in industry.

## 7.1 Main contributions

### 7.1.1 A framework to program composition of reactive and synchronized robot skillls - Chapter 3

The first contribution presents a framework that includes a constraint-based skill model and methodologies to compose skills. The skill model captures deviation behaviors that can react to continuous-level disturbances. This deviation can be specified to favor keeping the robot on the constrained path or keeping the robot on track with the desired progress. The proposed model leverages the constraint-based task specification framework eTaSL that supports defining feature variables to specify a skill progress. Composition of skills in a concurrent manner is made possible using a set of functions called composition patterns. These patterns allow to achieve specific composed behaviors such as progress synchronization and superposition of reactive behaviors. An experimental evaluation in a dual-arm assembly task of a screw compressor shows the effectiveness of the proposed framework.

### 7.1.2 A framework for integrating task planning and reactive skills to enable autonomous runtime skill composition - Chapter 4

The second contribution presents a framework that integrates symbolic task planning as a deliberative layer to automatically compose constraint-based skills. The composed skill exhibits concurrent behaviors that consist of a nominal skill and reactive skills, allowing the robots to adapt to unexpected disturbances. Most existing planners are unsuitable to generate this composition since they generally compute partial-order plans that do not guarantee concurrency. Therefore, as the first highlight of this contribution, we propose a temporal planning model for encoding symbolic durative actions. This model enforces a temporal planner to build fully concurrent actions as composed skills. We adopt PDDL2.1 that extends the standard PDDL for expressing the planning domains and durative actions. The second highlight is a methodology for runtime composition of constraint-based skills. Runtime composition of constraint-based

skills can introduce discontinuities in the control input due to sudden changes of the robot task nullspace. This contribution adopts a method that varies the skill weights in the QP optimization. It ensures that the control input sent to the robot remains smooth, therefore avoiding unwanted dangerous motions. This contribution is validated with a dual-arm assembly task of a solenoid pump in the presence of dynamic disturbances.

### 7.1.3 CAD-based task specification and skill programming for assembly applications - Chapter 5

The third contribution presents a system architecture and software implementation to semi-automatically generate sensor-based skills from CAD-level task specification for assembly applications. The architecture consists of three layers. The first layer is an application layer that facilitates specification of assembly tasks in a higher-level CAD domain. Here, domain experts can specify the end condition of the assembly by adding constraints between part features. The second layer is a process layer that infers suitable skill candidates and the appropriate parameters. The skill candidates selection is performed by a Prolog-based reasoner that takes into account a relational knowledge represented as ontology. The skill parameterization is broken down into two types. Application parameters such as the assembly direction are generated by reasoning on the geometric data of the CAD model; process parameters such as the insertion force are generated by a fuzzy inference module. Finally, the third layer is a control layer that executes the resulting parameterized skill with the underlying constraint-based controller. Two different assembly scenarios were implemented with two different robot types to validate the contribution.

## 7.2 Suggestions for future work

Apart from the future works suggested at the end of the contribution chapters, this thesis provides additional suggestions below.

### 7.2.1 Integration with motion planning approach

As explained in the discussion chapter, one of the main shortcomings of this work is the limitation of reactive planning in navigating around obstacles. A promising future work would be to alleviate this drawback by integrating a motion planner. In an environment that is filled with static obstacles such as

fixtures and tables, a motion planner becomes necesary to avoid getting stuck in local minima. In order to retain the advantage of reactivity and composability, the path computed by the motion planner can be used as a nominal path for the constraint-based skills. This would allow both a reactive execution of the planned path as well as combination with other behaviors, e.g., avoiding dynamic obstacles or admittance behaviors. Apart from addressing the above limitation, this integration can also further reduce the programming effort. By integrating a motion planner, collision-free paths can be computed automatically, therefore eliminating the need of manual programming of paths/waypoints.

### 7.2.2 An extended CAD-level assembly task specification

As stated in the discussion chapter, the main drawback of the proposed CAD-level assembly task specification is that it only captures the end conditions of the assembly. For many complex tasks, this approach is no longer effective since the assembly can only be performed by successive contact-based motions, as illustrated with the home switch assembly problem in Figure 6.2. As a future work, this thesis suggests to explore the subject of contact formation [134, 45] or also referred to as contact sequence [26]. Thomas et al. show that a graph that describes possible contact formations can be generated automatically based on the geometric information stored in CAD models [134]. However, in practice, a stable contact formation does not only depend on the workpiece geometries, but also other aspects such as manufacturing tolerances, inaccurate robot/tool calibration, etc. Therefore, an alternative approach would be to delegate the contact sequence synthesis to the programmer. Assuming that the programmer is a domain expert in the assembly tasks at hand, it is reasonable to expect that he/she can synthesize an appropriate contact sequence while taking into account various aspects that are difficult to incorporate into the programming framework. The scientific question is then to define a formal representation of the assembly task that describes both the desired end conditions of the assembly and the corresponding contact formation.

### 7.2.3 Exploration of tactile sensing for improved manipulation skills

The third suggestion for future work is motivated by the limitation observed during the experimental evaluation of the thesis contributions. For all the three contributions, this thesis demonstrates assembly tasks that utilize force/torque (F/T) sensors. While F/T sensors are easy to integrate and useful for assembly tasks such as peg-in-hole, more advanced manipulation skills can be achieved

by integrating tactile sensors as an additional sensing modality. Some of the limitations that can be alleviated and advantages that can be obtained with this integration include:

- a better failure detection. During the compressor assembly experiments in Chapter 3 and 5, it was observed that, after numerous trials, the robot fingers became worn out due to friction. In some of the trials, this caused the workpieces to slip. During the picking task the workpiece may fall from the gripper while during the insertion task the workpiece may shift. While the first failure can be detected with a F/T sensor, the second failure is more difficult since the F/T signal cannot reliably distinguish forces due to slip and insertion.
- possibility of in-hand workpiece state estimation. Recent work such as [46, 68, 139] shows that today's tactile sensing technology offers sufficient resolution and sampling rate to estimate the state of objects in hand in real-time. This could potentially improve the robustness of manipulation skills when there are uncertainties in the object state in the gripper (in-hand state). For example, by compensating task frame dislocations when slip occurs.
- reducing the need of special fingers design. As skills that use F/T sensors only work reliably when in-hand object state does not change, we often had to purposely design the fingers specificically to match the object shapes in order to constrain the object motions. For example, the finger design for gripping a cylindrical shape such as the compressor rotor would be different compared to planar surfaces such as the switch button in Figure 6.2. The use of tactile sensing may alleviate the need to design specific fingers if the object state can be estimated and compensated.

# Bibliography


[1] ABB Robotics. *Technical reference manual: RAPID instructions, functions and data types*. ABB Robotics, 2014.

[2] ABB Robotics. *Application manual: Wizard*. ABB Robotics, 2021.

[3] Aertbeliën, E., and De Schutter, J. ETaSL/eTC: A constraint-based task specification language and robot controller using expression graphs. In *IEEE International Conference on Intelligent Robots and Systems* (2014), pp. 1540–1546.

[4] Alami, R., Simeon, T., and Laumond, J.-P. A geometrical approach to planning manipulation tasks. the case of discrete placements and grasps. In *The fifth international symposium on Robotics research* (1990), MIT Press, pp. 453–463.

[5] Alt, B., Katic, D., Jäkel, R., Bozcuoglu, A. K., and Beetz, M. Robot program parameter inference via differentiable shadow program inversion. *arXiv preprint arXiv:2103.14452* (2021).

[6] Ambler, A., and Popplestone, R. Inferring the positions of bodies from specified spatial relationships. *Artificial Intelligence 6*, 2 (June 1975), 157–174.

[7] Antoniou, G., and Van Harmelen, F. Web ontology language: Owl. In *Handbook on Ontologies in Information Systems* (2003), Springer, pp. 67–92.

[8] Arbo, M. H., and Pane, Y. ARBench. `https://github.com/mahaarbo/ARBench`. [Online; accessed 06 April 2021].

[9] Arbo, M. H., and Pane, Y. ARBench part publisher. `https://github.com/mahaarbo/arbench_part_publisher`. [Online; accessed 06 April 2021].

[10] Arbo, M. H., Pane, Y., Aertbeliën, E., and Decré, W. A System Architecture for Constraint-Based Robotic Assembly with CAD Information. In *IEEE 14th International Conference on Automation Science and Engineering (CASE'18)* (Munich, August 2018), pp. 690–696.

[11] Bartels, G., Kresse, I., and Beetz, M. Constraint-based movement representation grounded in geometric features. In *2013 13th IEEE-RAS International Conference on Humanoid Robots (Humanoids)* (2013), pp. 547–554.

[12] Behringer, F. A. Lexicographic quasiconcave multiobjective programming. *Zeitschrift für Operations Research 21*, 3 (1977), 103–116.

[13] Benton, J., Coles, A., and Coles, A. Temporal planning with preferences and time-dependent continuous costs. In *22nd International Conference on Automated Planning and Scheduling (ICAPS)* (2012), pp. 1–10.

[14] Bishop, C. M. *Pattern Recognition and Machine Learning*. Springer, New York, 2006.

[15] Bohren, J., and Cousins, S. The smach high-level executive. *IEEE Robotics and Automation Magazine 17*, 4 (2010), 18–20.

[16] Borghesan, G., Scioni, E., Kheddar, A., and Bruyninckx, H. Introducing geometric constraint expressions into robot constrained motion specification and control. *IEEE Robotics and Automation Letters 1*, 2 (2016), 1140–1147.

[17] Brafman, R. I., and Domshlak, C. From one to many: Planning for loosely coupled multi-agent systems. In *18th International Conference on Automated Planning and Scheduling (ICAPS)* (2008), pp. 28–35.

[18] Brock, O., and Khatib, O. Elastic Strips: A Framework for Motion Generation in Human Environments. *The International Journal of Robotics Research 21*, 12 (2004), 1031–1052.

[19] Bruyninckx, H., and De Schutter, J. Specification of force-controlled actions in the "Task frame formalism" - A synthesis. *IEEE Transactions on Robotics and Automation 12*, 4 (1996), 581–589.

[20] Chernoff, A. W., and Warman, C. Covid-19 and implications for automation. Working Paper 27249, National Bureau of Economic Research, July 2020.

[21] Coleman, D., Sucan, I., Chitta, S., and Correll, N. Reducing the barrier to entry of complex robotic software: a moveit! case study. *Journal of Software Engineering for Robotics 5*, 1 (2014), 3–16.

[22] Coombs, C. Will covid-19 be the tipping point for the intelligent automation of work? a review of the debate and implications for research. *International Journal of Information Management 55* (2020), 102182. Impact of COVID-19 Pandemic on Information Management Research and Practice: Editorial Perspectives.

[23] Crosby, M., and Petrick, R. P. A. Temporal Multiagent Planning with Concurrent Action Constraints. *Proceedings of the 2nd ICAPS Distributed and Multi-Agent Planning workshop (ICAPS DMAP-2014)*, Brenner (2014), 16–24.

[24] Dantam, N. T., Kingston, Z. K., Chaudhuri, S., and Kavraki, L. E. An incremental constraint-based framework for task and motion planning. *The International Journal of Robotics Research 37*, 10 (2018), 1134–1151.

[25] De Fazio, T., and Whitney, D. Simplified generation of all mechanical assembly sequences. *IEEE Journal on Robotics and Automation 3*, 6 (December 1987), 640–658.

[26] De Schutter, J., Bruyninckx, H., Dutré, S., De Geeter, J., Katupitiya, J., Demey, S., and Lefebvre, T. Estimating first-order geometric parameters and monitoring contact transitions during force-controlled compliant motion. *The International Journal of Robotics Research 18*, 12 (1999), 1161–1184.

[27] Dechter, R., Meiri, I., and Pearl, J. Temporal constraint networks. *Artificial Intelligence 49*, 1 (1991), 61–95.

[28] Department, I. S. World robotics report 2020. In *VDMA Robotics + Automation* (2020), International Federation of Robotics.

[29] Di Rocco, M., Pecora, F., and Saffiotti, A. When robots are late: Configuration planning for multiple robots with dynamic goals. In *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems* (2013), pp. 9515–5922.

[30] Dietrich, F., Maass, J., Raatz, A., and Hesselbach, J. RCA562: Control Architecture for Parallel Kinematic Robots. In *Springer Tracts in Advanced Robotics*, vol. 67. Springer, 2010, pp. 315–331.

[31] Ding, D., Shen, C., Pan, Z., Cuiuri, D., Li, H., Larkin, N., and van Duin, S. Towards an automated robotic arc-welding-based additive manufacturing system from cad to finished part. *Computer-Aided Design 73* (2016), 66–75.

[32] drag and bot GmbH. drag&bot os: graphical robot operating system. `https://www.dragandbot.com/software/`, 2021. [Online; accessed 06 April 2021].

[33] Escande, A., Mansard, N., and Wieber, P.-B. Fast resolution of hierarchized inverse kinematics with inequality constraints. In *2010 IEEE International Conference on Robotics and Automation* (2010), pp. 3733–3738.

[34] Escande, A., Mansard, N., and Wieber, P.-B. Hierarchical quadratic programming: Fast online humanoid-robot motion generation. *The International Journal of Robotics Research 33*, 7 (2014), 1006–1028.

[35] Ferreau, H. J., Kirches, C., Potschka, A., Bock, H. G., and Diehl, M. qpoases: A parametric active-set algorithm for quadratic programming. *Mathematical Programming Computation 6*, 4 (2014), 327–363.

[36] Fikes, R. E., and Nilsson, N. J. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial intelligence 2*, 3 (1972), 189–208.

[37] Fox, M., and Long, D. PDDL2.1: An extension to PDDL for expressing temporal planning domains. *Journal of artificial intelligence research 20* (2003), 61–124.

[38] Franka Emika GmbH. *Panda's Instruction Handbook*. Franka Emika GmbH, 2020.

[39] Garrett, C. R., Paxton, C., Lozano-Pérez, T., Kaelbling, L. P., and Fox, D. Online replanning in belief space for partially observable task and motion problems. In *International Conference on Robotics and Automation (ICRA)* (2020), IEEE, pp. 5678–5684.

[40] Gepetto Research Group, LAAS-CNRS. Stack of tasks: Software for controlling redundant robots. `http://stack-of-tasks.github.io/`, 2021. [Online; accessed 24 September 2021].

[41] Ghallab, M., Nau, D., and Traverso, P. *Automated planning: theory & practice*. Elsevier, 2004.

[42] Ghallab, M., Nau, D., and Traverso, P. The actor's view of automated planning and acting: A position paper. *Artificial Intelligence 208* (2014), 1–17.

[43] Guillaume, S. Designing fuzzy inference systems from data: An interpretability-oriented review. *IEEE Transactions on Fuzzy Systems 9*, 3 (jun 2001), 426–443.

[44] Halt, L., Nagele, F., Tenbrock, P., and Pott, A. Intuitive constraint-based robot programming for robotic assembly tasks. In *IEEE International Conference on Robotics and Automation (ICRA)* (2018).

[45] Hirai, S. *Analysis and planning of manipulation using the theory of polyhedral convex cones*. PhD thesis, Kyoto University, 1991.

[46] Hogan, F. R., Ballester, J., Dong, S., and Rodriguez, A. Tactile dexterity: Manipulation primitives with tactile feedback. In *2020 IEEE international conference on robotics and automation (ICRA)* (2020), IEEE, pp. 8863–8869.

[47] Homem de Mello, L., and Sanderson, A. Representations of mechanical assembly sequences. *IEEE Transactions on Robotics and Automation 7*, 2 (apr 1991), 211–227.

[48] Ierusalimschy, R. *Programming in Lua, 4th edition*. Lua.Org, 2016.

[49] Ingrand, F., and Ghallab, M. Deliberation for autonomous robots: a survey. *Artificial Intelligence 247* (2015), 10–44.

[50] Jarquín, G., Escande, A., Arechavaleta, G., Moulard, T., Yoshida, E., and Parra-Vega, V. Real-time smooth task transitions for hierarchical inverse kinematics. In *IEEE-RAS International Conference on Humanoid Robots* (2013), pp. 528–533.

[51] Kalakrishnan, M., Righetti, L., Pastor, P., and Schaal, S. Learning force control policies for compliant manipulation. In *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems* (September 2011), pp. 4639–4644.

[52] Kang, Y., Kim, D., and Kim, K. Urdf generator for manipulator robot. In *2019 Third IEEE International Conference on Robotic Computing (IRC)* (2019), IEEE, pp. 483–487.

[53] Kanoun, O., Lamiraux, F., and Wieber, P.-B. Kinematic control of redundant manipulators: Generalizing the task-priority framework to inequality task. *IEEE Transactions on Robotics 27*, 4 (2011), 785–792.

[54] Kaufman, S., Wilson, R., Jones, R., Calton, T., and Ames, A. The archimedes 2 mechanical assembly planning system. In *Proceedings of IEEE International Conference on Robotics and Automation (ICRA)* (1996), vol. 4, pp. 3361–3368 vol.4.

[55] Kavraki, L. E., Svestka, P., Latombe, J.-C., and Overmars, M. H. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE transactions on Robotics and Automation 12*, 4 (1996), 566–580.

[56] Keith, F., Wieber, P.-B., Mansard, N., and Kheddar, A. Analysis of the discontinuities in prioritized tasks-space control under discreet task scheduling operations. In *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems* (2011), pp. 3887–3892.

[57] Khatib, O. Real-time obstacle avoidance for manipulators and mobile robots. In *Proceedings. 1985 IEEE International Conference on Robotics and Automation* (1985), vol. 2, pp. 500–505.

[58] Khatib, O., Sentis, L., Park, J., and Warren, J. Whole-body dynamic behavior and control of human-like robots. *International Journal of Humanoid Robotics 1*, 01 (2004), 29–43.

[59] Kingston, D., Ren, W., and Beard, R. Consensus algorithms are input-to-state stable. In *Proceedings of the 2005 American Control Conference* (2005), vol. 3, pp. 1686–1690.

[60] Klotzbücher, M., and Bruyninckx, H. Coordinating robotic tasks and systems with rfsm statecharts. *Journal of Software Engineering for Robotics 3*, 1 (2012), 28–56.

[61] Knoblock, C. A. Generating parallel execution plans with a partial order planner. In *AIPS* (1994), vol. 94, pp. 98–103.

[62] Korsah, G. A., Kannan, B., Browning, B., Stentz, A., and Dias, M. B. xbots: An approach to generating and executing optimal multi-robot plans with cross-schedule dependencies. In *2012 IEEE International Conference on Robotics and Automation* (2012), pp. 115–122.

[63] Korsoveczki, G., Husi, G., and Erdei, T. I. The maintenance, commissioning of fanuc m-1ia 0.5 a type selecting robot and implementation of irvision picture-recognizing procedure. In *MATEC Web of Conferences* (2018), vol. 184, EDP Sciences, p. 02022.

[64] Kortenkamp, D., Simmons, R., and Brugali, D. Robotic systems architectures and programming. In *Springer Handbook of Robotics*. Springer, 2016, pp. 283–306.

[65] Kuss, A., Diaz P., J. R., Hollmann, R., Dietz, T., and Hägele, M. Manufacturing knowledge for industrial robot systems: Review and synthesis of model architecture. In *2016 IEEE International Conference on Automation Science and Engineering (CASE)* (August 2016), pp. 348–354.

[66] Kuss, A., Dietz, T., Ksensow, K., and Verl, A. Manufacturing task description for robotic welding and automatic feature recognition on product cad models. *Procedia Cirp 60* (2017), 122–127.

[67] Lagriffoul, F., Dantam, N. T., Garrett, C., Akbari, A., Srivastava, S., and Kavraki, L. E. Platform-independent benchmarks for task and motion planning. *IEEE Robotics and Automation Letters 3*, 4 (2018), 3765–3772.

[68] Lambeta, M., Chou, P.-W., Tian, S., Yang, B., Maloon, B., Most, V. R., Stroud, D., Santos, R., Byagowi, A., Kammerer, G., et al. Digit: A novel design for a low-cost compact high-resolution tactile sensor with application to in-hand manipulation. *IEEE Robotics and Automation Letters 5*, 3 (2020), 3838–3845.

[69] LaValle, S. M., and Kuffner Jr, J. J. Randomized kinodynamic planning. *The international journal of robotics research 20*, 5 (2001), 378–400.

[70] Lee, J., Mansard, N., and Park, J. Intermediate desired value approach for task transition of robots in kinematic control. *IEEE Transactions on Robotics 28*, 6 (2012), 1260–1277.

[71] Levine, S., Wagener, N., and Abbeel, P. Learning contact-rich manipulation skills with guided policy search. In *2015 IEEE International Conference on Robotics and Automation (ICRA)* (May 2015), pp. 156–163.

[72] Lieberman, L. I., and Wesley, M. A. Autopass: An automatic programming system for computer controlled mechanical assembly. *IBM Journal of Research and Development 21*, 4 (1977), 321–333.

[73] Lind, M., Schrimpf, J., and Ulleberg, T. Open Real-Time Robot Controller Framework. In *3rd CIRP Conference on Assembly Technology and Systems (CATS'10)* (Trondheim, 2010), pp. 13–18.

[74] Lozano-Perez, T., Mason, M. T., and Taylor, R. H. Automatic synthesis of fine-motion strategies for robots. *The International Journal of Robotics Research 3*, 1 (1984), 3–24.

[75] Mansard, N., and Chaumette, F. Task sequencing for high-level sensor-based control. *IEEE Transactions on Robotics 23*, 1 (2007), 60–72.

[76] Mansard, N., Khatib, O., and Kheddar, A. A unified approach to integrate unilateral constraints in the stack of tasks. *IEEE Transactions on Robotics 25*, 3 (2009), 670–685.

[77] Mansard, N., Stasse, O., Evrard, P., and Kheddar, A. A versatile generalized inverted kinematics implementation for collaborative working humanoid robots: The stack of tasks. In *2009 International Conference on Advanced Robotics* (2009), pp. 1–6.

[78] Mason, M. T. Compliance and Force Control for Computer Controlled Manipulators. *IEEE Transactions on Systems, Man, and Cybernetics 11*, 6 (1981), 418–432.

[79] Mcdermott, D., Ghallab, M., Howe, A., Knoblock, C., Ram, A., Veloso, M., Weld, D., and Wilkins, D. PDDL – the planning domain definition language. Tech. rep., Technical Report CVC TR-98-003/DCS TR-1165, Yale Center for Computational Vision and Control, 1998.

[80] Mercy, T., Van Parys, R., and Pipeleers, G. Spline-based motion planning for autonomous guided vehicles in a dynamic environment. *IEEE Transactions on Control Systems Technology 26*, 6 (2017), 2182–2189.

[81] Mirrazavi Salehian, S. S., Figueroa, N., and Billard, A. A unified framework for coordinated multi-arm motion planning. *The International Journal of Robotics Research 37*, 10 (2018), 1205–1232.

[82] Morgan Quigley, M., Gerkey, B., Conley, K., Faust, J., Foote, T., Leibs, J., Berger, E., Wheeler, R., and Ng, A. ROS: an open-source Robot Operating System. In *IEEE International Conference on Robotics and Automation (ICRA) Workshop on Open Source Robotics* (2009).

[83] Mosemann, H., and Wahl, F. M. Automatic Decomposition of Planned Assembly Sequences Into Skill Primitives. *IEEE Transactions on Robotics and Automation 17*, 5 (2001), 709–718.

[84] Mudrova, L., and Hawes, N. Task scheduling for mobile robots using interval algebra. In *2015 IEEE International Conference on Robotics and Automation (ICRA)* (2015), pp. 383–388.

[85] Mühe, H., Angerer, A., Hoffmann, A., and Reif, W. On reverse-engineering the kuka robot language. *arXiv preprint arXiv:1009.5004* (2010).

[86] Murphy, K., Norcross, R. J., and Proctor, F. M. CAD directed robotic deburring. In *Proceedings of the second international symposium on robotics and manufacturing research, education, and applications* (1988).

[87] Musen, M. A. The protégé project: a look back and a look forward. *AI matters 1*, 4 (2015), 4–12.

[88] Nilsson, N. J. *Problem-Solving Methods in Artificial Intelligence.* McGraw-Hill Pub. Co., 1971.

[89] Nunes, E., and Gini, M. Multi-robot auctions for allocation of tasks with temporal constraints. In *Proceedings of the AAAI Conference on Artificial Intelligence* (2015), vol. 29.

[90] Nägele, F., Halt, L., Tenbrock, P., and Pott, A. A prototype-based skill model for specifying robotic assembly tasks. In *2018 IEEE International Conference on Robotics and Automation (ICRA)* (2018), pp. 558–565.

[91] Pacheco, P. *An introduction to parallel programming.* Elsevier, 2011.

[92] Pane, Y., Aertbeliën, E., De Schutter, J., and Decré, W. Skill-based programming framework for composable reactive robot behaviors. In *IROS* (2020), IEEE, pp. 7087–7094.

[93] Pane, Y., Aertbeliën, E., Schutter, J. D., and Decré, W. Skill-based programming framework for composable reactive robot behaviors. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (2020), pp. 7087–7094.

[94] Pane, Y., and Arbo, M. H. CAD Based Assembly Ontology. `https://github.com/yudhapane/cad_based_assembly_ontology`. [Online; accessed 06 April 2021].

[95] Pedersen, M. R., Nalpantidis, L., Andersen, R. S., Schou, C., Bøgh, S., Krüger, V., and Madsen, O. Robot skills for manufacturing: From concept to industrial deployment. *Robotics and Computer-Integrated Manufacturing 37* (2016), 282–291.

[96] Pedersen, M. R., Nalpantidis, L., Bobick, A., and Krüger, V. On the Integration of Hardware-Abstracted Robot Skills for use in Industrial Scenarios. *2nd International IROS Workshop on Cognitive Robotics Systems: Replicating Human Actions and Activities* (2013).

[97] Perzylo, A., Rickert, M., Kahl, B., Somani, N., Lehmann, C., Kuss, A., Profanter, S., Beck, A. B., Haage, M., Rath Hansen,

M., Nibe, M. T., Roa, M. A., Sörnmo, O., Gestegård Robertz, S., Thomas, U., Veiga, G., Topp, E. A., Kessler, I., and Danzer, M. SMErobotics: Smart Robots for Flexible Manufacturing. *IEEE Robotics Automation Magazine 26*, 1 (2019), 78–90.

[98] Perzylo, A., Somani, N., Profanter, S., Kessler, I., Rickert, M., and Knoll, A. Intuitive instruction of industrial robots: Semantic process descriptions for small lot production. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (2016), pp. 2293–2300.

[99] Perzylo, A., Somani, N., Rickert, M., and Knoll, A. An ontology for CAD data and geometric constraints as a link between product models and semantic robot task descriptions. In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (2015), pp. 4197–4203.

[100] Pitchandi, N., Subramanian, S. P., and Irulappan, M. Insertion force analysis of compliantly supported peg-in-hole assembly. *Assembly Automation 37*, 3 (2017), 285–295.

[101] Pochet, Y., and Wolsey, L. A. *Production planning by mixed integer programming*. Springer Science & Business Media, 2006.

[102] Quinlan, S., and Khatib, O. Elastic bands: connecting path planning and control. In *1993 Proceedings IEEE International Conference on Robotics and Automation* (1993), vol. 2, pp. 802–807.

[103] Rall, L. B., and LB, R. *Automatic differentiation: Techniques and applications*. No. 120 in Lecture Notes in Computer Science. Springer, 1981.

[104] Ready Robotics. *Forge/station: general application guide*. Ready Robotics, 2020.

[105] Riegel, J. Freecad: a free extensible cax system, 2002.

[106] Rovida, F., Crosby, M., Holz, D., Polydoros, A. S., Grossmann, B., Petrick, R. P., and Krüger, V. SkiROS – a skill-based robot control platform on top of ROS. In *Robot Operating System (ROS)*, vol. 2. Springer, 2017, pp. 121–160.

[107] Samson, C., Espiau, B., and Borgne, M. L. *Robot control: the task function approach*. Oxford University Press, Inc., 1991.

[108] Sathya, A. S., Gillis, J., Pipeleers, G., and Swevers, J. Real-time robot arm motion planning and control with nonlinear model predictive control using augmented lagrangian on a first-order solver. In *2020 European Control Conference (ECC)* (2020), pp. 507–512.

[109] Sathya, A. S., Pipeleers, G., Decré, W., and Swevers, J. A weighted method for fast resolution of strictly hierarchical robot task specifications using exact penalty functions. *IEEE Robotics and Automation Letters 6*, 2 (2021), 3057–3064.

[110] Schmitt, P. S., Wirnshofer, F., Wurm, K. M., Wichert, G. v., and Burgard, W. Modeling and planning manipulation in dynamic environments. In *2019 International Conference on Robotics and Automation (ICRA)* (2019), pp. 176–182.

[111] Schreiber, G., Stemmer, A., and Bischoff, R. The fast research interface for the KUKA lightweight robot. In *IEEE ICRA 2010 Workshop on Innovative Robot Control Architectures for Demanding (Research) Applications* (2010).

[112] Schulman, J., Ho, J., Lee, A. X., Awwal, I., Bradlow, H., and Abbeel, P. Finding locally optimal, collision-free trajectories with sequential convex optimization. In *Robotics: science and systems* (2013), vol. 9, Citeseer, pp. 1–10.

[113] Schutter, J. D., Laet, T. D., Rutgeerts, J., Decre, W., Smits, R., Aertbelien, E., Claes, K., and Bruyninckx, H. Constraint-based Task Specification and Estimation for Sensor-Based Robot Systems in the Presence of Geometric Uncertainty. *The International Journal of Robotics Research 23*, 3 (2007), 433–455.

[114] Scioni, E. *Online Coordination and Composition of Robotic Skills: Formal Models for Context-aware Task Scheduling.* PhD thesis, KU Leuven, 2016.

[115] Scioni, E., Borghesan, G., Bruyninckx, H., and Bonfè, M. Bridging the gap between discrete symbolic planning and optimization-based robot control. In *2015 IEEE International Conference on Robotics and Automation (ICRA)* (2015), pp. 5075–5081.

[116] Sentis, L., and Khatib, O. Synthesis of whole-body behaviors through hierarchical control of behavioral primitives. *International Journal of Humanoid Robotics 2*, 04 (2005), 505–518.

[117] Sentis, L., and Khatib, O. A whole-body control framework for humanoids operating in human environments. In *Proceedings 2006 IEEE*

*International Conference on Robotics and Automation (ICRA)* (2006), pp. 2641–2648.

[118] Siciliano, B., and Slotine, J.-J. A general framework for managing multiple tasks in highly redundant robotic systems. In *Fifth International Conference on Advanced Robotics 'Robots in Unstructured Environments* (1991), pp. 1211–1216 vol.2.

[119] Siemens. Kineoworks: Quickly and easily compute collision-free motion with the leading path planning solution, 2021.

[120] Smits, R., De Laet, T., Claes, K., Bruyninckx, H., and De Schutter, J. itasc: a tool for multi-sensor integration in robot manipulation. In *2008 IEEE International Conference on Multisensor Fusion and Integration for Intelligent Systems* (2008), pp. 426–433.

[121] Soetens, P. *A Software Framework for Real-Time and Distributed Robot and Machine Control*. PhD thesis, Katholieke Universiteit Leuven, 2006.

[122] Somani, N., Gaschler, A., Rickert, M., Perzylo, A., and Knoll, A. Constraint-based task programming with CAD semantics: From intuitive specification to real-time control. In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (2015), pp. 2854–2859.

[123] Srivastava, S., Fang, E., Riano, L., Chitnis, R., Russell, S., and Abbeel, P. Combined task and motion planning through an extensible planner-independent interface layer. In *2014 IEEE International Conference on Robotics and Automation (ICRA)* (2014), pp. 639–646.

[124] Stellato, B., Banjac, G., Goulart, P., Bemporad, A., and Boyd, S. OSQP: an operator splitting solver for quadratic programs. *Mathematical Programming Computation 12*, 4 (2020), 637–672.

[125] Stenmark, M., Haage, M., and Topp, E. A. Simplified Programming of Re-usable Skills on a Safe Industrial Robot: Prototype and Evaluation. *ACM/IEEE International Conference on Human-Robot Interaction Part F1271* (2017), 463–472.

[126] Stenmark, M., and Malec, J. Knowledge-based instruction of manipulation tasks for industrial robotics. *Robotics and Computer-Integrated Manufacturing 33* (2015), 56–67.

[127] Stenmark, M., and Topp, E. A. From demonstrations to skills for high-level programming of industrial robots. *AAAI Fall Symposium - Technical Report FS-16-01* - (2016), 75–78.

[128] Tenorth, M., Bartels, G., and Beetz, M. Knowledge-based specification of robot motions. In *European Conference on Artificial Intelligence* (2014), IOS Press, pp. 873–878.

[129] Tenorth, M., and Beetz, M. Representations for robot knowledge in the KnowRob framework. *Artificial Intelligence 247* (2017), 151–169.

[130] Teulieres, M., Tilley, J., Bolz, L., Ludwig-Dehm, P. M., and Wägner, S. Industrial robotics: insights into the sector's future growth dynamics. In *Growth dynamics in industrial robotics* (July 2019), McKinsey & Company.

[131] Thomas, U., Hirzinger, G., Rumpe, B., Schulze, C., and Wortmann, A. A New Skill Based Robot Programming Language Using UML / P Statecharts. *IEEE International Conference on Robotics and Automation (ICRA)* (2013), 461–466.

[132] Thomas, U., Wahl, F., Maass, J., and Hesselbach, J. Towards a new concept of robot programming in high speed assembly applications. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS'05)* (2005), pp. 3827–3833.

[133] Thomas, U., and Wahl, F. M. A system for automatic planning, evaluation and execution of assembly sequences for industrial robots. In *Proceedings 2001 IEEE/RSJ International Conference on Intelligent Robots and Systems* (2001), vol. 3, IEEE, pp. 1458–1464.

[134] Thomas, U., and Wahl, F. M. Assembly Planning and Task Planning — Two Prerequisites for Automated Robot Programming. In *Robotic Systems for Handling and Assembly*. Springer, 2010, pp. 333–354.

[135] Vanthienen, D., De Laet, T., Decré, W., and De Schutter, J. Acceleration- vs. velocity-resolved constraint-based instantaneous task specification and estimation (iTaSC). In *Benelux meeting on systems and control 2013.* (2013).

[136] Vasilopoulos, V., Topping, T. T., Vega-Brown, W., Roy, N., and Koditschek, D. E. Sensor-based reactive execution of symbolic rearrangement plans by a legged mobile manipulator. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (2018), pp. 3298–3305.

[137] Veloso, M. M., Pérez, A., and Carbonell, J. G. Nonlinear planning with parallel resource allocation. In *Innovative Approaches to Planning, Scheduling and Control: Proceedings of the DARPA Workshop* (1990), MorganKaufmann, pp. 207–212.

[138] Vergara, C., Iregui, S., De Schutter, J., and Aertbeliën, E. Generating reactive approach motions towards allowable manifolds using generalized trajectories from demonstrations. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (2020), pp. 9697–9704.

[139] Villalonga, M. B., Rodriguez, A., Lim, B., Valls, E., and Sechopoulos, T. Tactile object pose estimation from the first touch with geometric contact rendering. In *Conference on Robot Learning* (2021), PMLR, pp. 1015–1029.

[140] Waibel, M., Beetz, M., Civera, J., D'Andrea, R., Elfring, J., Gálvez-López, D., Häussermann, K., Janssen, R., Montiel, J. M. M., Perzylo, A., Schiessle, B., Tenorth, M., Zweigle, O., and De Molengraft, R. V. Roboearth. *IEEE Robotics Automation Magazine 18*, 2 (June 2011), 69–82.

[141] Wardhani, R., and Xu, X. Model-based manufacturing based on step ap242. In *2016 12th IEEE/ASME International Conference on Mechatronic and Embedded Systems and Applications (MESA)* (2016), pp. 1–5.

[142] Wielemaker, J., Schrijvers, T., Triska, M., and Lager, T. SWI-Prolog. *Theory and Practice of Logic Programming 12*, 1-2 (2012), 67–96.

[143] Wilson, M. *Implementation of Robot Systems: An Introduction to Robotics, Automation, and Successful Systems Integration in Manufacturing*. Butterworth-Heinemann, Oxford, 2015.

[144] Yudha Pane. Expressiongraph solver hqp. `https://gitlab.kuleuven.be/u0108404/expressiongraph_solver_hqp`, 2021. [Online; accessed 16 August 2021].

[145] Yudha Pane. Expressiongraph solver hqp - factory component. `https://gitlab.kuleuven.be/u0108404/etasl_solver_hqp`, 2021. [Online; accessed 16 August 2021].

# List of publications

## Articles Published in Internationally Reviewed Academic Journals

- Pane, Y., Mokhtari, V., Aertbeliën, E., De Schutter, J., and Decré, W. Autonomous Runtime Composition of Sensor-based Skills Using Concurrent Task Planning. In *IEEE Robotics and Automation Letters* (2021).
- Pane, Y., Arbo, M. H., Aertbeliën, E., and Decré, W. A System Architecture for CAD-Based Robotic Assembly With Sensor-Based Skills. In *IEEE Transactions On Automation Science And Engineering*, 17 (3), pp. 1237-1249 (2020).

## Papers at international conferences, published in full in proceedings

- Pane, Y., Aertbeliën, E., De Schutter, J., and Decré, W. Skill-based Programming Framework for Composable Reactive Robot Behaviors. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 7087-7094, Las Vegas, USA (2020).
- Tirmizi, A., Cat, B. D., Janssen, K., Pane, Y., Leconte, P., and Witters, M. User-Friendly Programming of Flexible Assembly Applications with Collaborative Robots. In *20th International Conference on Research and Education in Mechatronics (REM)*, pp. 1-7, Wels, Austria (2019).
- Hosseini, M., Sengül, A., Pane, Y., De Schutter, J., and Bruyninckx, H. ExoTen-Glove: A Force-Feedback Haptic Glove Based on

Twisted String Actuation System. In *27th IEEE International Symposium on Robot and Human Interactive Communication (RO-MAN)*, pp. 320-327, Nanjing, China (2018).

- Hosseini, M., Sengül, A., Pane, Y., De Schutter, J., and Bruyninckx, H. Haptic Perception of Virtual Spring Stiffness Using ExoTen-Glove. In *11th International Conference on Human System Interaction (HSI)*, pp. 526-531, Gdansk, Poland (2018).
- Arbo, M. H., Pane, Y., Aertbeliën, E. and Decré, W. A System Architecture for Constraint-Based Robotic Assembly with CAD Information. In *IEEE 14th International Conference on Automation Science and Engineering (CASE)*, pp. 690-696, Munich, Germany (2018). **(Best student paper award)**

## Book chapters

- Hosseini, M., Pane, Y., Sengül, A., De Schutter, J., and Bruyninckx, H. A Novel Haptic Glove (ExoTen-Glove) Based on Twisted String Actuation (TSA) System for Virtual Reality. In *Haptics: Science, Technology, and Applications*, pp. 612-622 (2018).

## Abstracts/Presentations/Posters

- Vergara, C., Cuellar, J., Ulloa, F., Iregui Rincon, S., Vanroye, L., Viljoen, R., Pane, Y., De Schutter, J., Aertbeliën, E., and Decré, W. Learning Robust Dual-Arm Coordination in the Manipulation of Elastic Elements from Intuitive Human Demonstrations. In *ICRA2021 Workshop on Bimanual Manipulation: Learning, Planing, and Control*, Xi'an, China (2021).
- Pane, Y., Aertbeliën, E., De Schutter, J., and Decré, W. A Composable Robot Skill Programming Framework. In *Flanders Make Scientific Conference*, Leuven, Belgium (2021).
- De Winter, J., Vergara, C., Bagheri, E., Pane, Y., Hostens, E., Aertbeliën, E., Vanderborght, B., and Decré, W. PROUD: A Hybrid Model-Based/Learning Approach for Symbolic-Level Robot Programming by User Demonstration. In *ICRA2020 Workshop: Integrating Multidisciplinary Approaches to Advance Physical Human-Robot Interaction*, Paris, France (2020).

- Pane, Y., Decré, W., and De Schutter, J. Knowledge-based Skill Programming Framework. In *The 38th Benelux Meeting on Systems and Control*, Lommel, Belgium (2019).
- Pane, Y., Decré, W., and De Schutter, J. Composable Skill Programming Framework for Complex Sensor-Based Robot Tasks. In *The 37th Benelux Meeting on Systems and Control*, Soesterberg, The Netherlands (2018).
- Pane, Y., Decré, W., and De Schutter, J. (2018). A composable skill programming framework for sensor-based robot tasks. In *IROS2018 Workshop: Semantic Policy and Action Representations for Autonomous Robots (SPAR)*, Madrid, Spain (2018).



FACULTY OF ENGINEERING SCIENCE
DEPARTMENT OF MECHANICAL ENGINEERING
ROBOTICS RESEARCH GROUP
Celestijnenlaan 300
B-3001 Leuven
yudha.pane@kuleuven.be
https://mech.kuleuven.be/robotics