Learning with Skill-based Robot Systems

Combining Planning & Knowledge Representation with Reinforcement Learning

Mayr, Matthias

2024

Document Version:
Publisher's PDF, also known as Version of record

Learning with Skill-based Robot Systems

Learning with Skill-based Robot Systems

Combining Planning & Knowledge Representation with Reinforcement Learning

by Matthias Mayr

LUND
UNIVERSITY

Thesis for the degree of Doctor of Philosophy
Thesis advisors: Prof. Dr. Volker Krueger, Prof. Dr. Jacek Malec
Faculty opponent: Prof. Dr. hc. Michael Beetz

To be presented, with the permission of the Faculty of Engineering (LTH) of Lund University, for public criticism in the E:1406 at the Department of Computer Science on Friday, the 2nd of February 2024 at 10:00.

Organization
LUND UNIVERSITY

Department of Computer Science
Box 118
SE–221 00 LUND
Sweden

Document name
DOCTORAL DISSERTATION

Date of disputation
2024-02-02

Sponsoring organization
Wallenberg AI, Autonomous Systems and Software Program (WASP)

Author(s)
Matthias Mayr

Title and subtitle
Learning with Skill-based Robot Systems: Combining Planning & Knowledge Representation with Reinforcement Learning

Abstract
The usage of robots in industry is transforming. Traditionally, robots have been deployed to automate monotonous tasks through manual programming, excelling in speed and precision yet lacking flexibility. Now, as part of *Industry 4.0*, the paradigm is shifting towards collaborative robotics, where robots are expected to interact dynamically with their environment and handle non-repetitive tasks. This evolution demands a leap towards flexibility and adaptability at both control and task levels. To address these challenges, the concept of "robot skills" — reusable, parameterizable procedures — emerges as a potentially pivotal building block. The skill-based robot control system *SkiROS2* is designed to be robot-agnostic and to represent such skills and the necessary knowledge. This knowledge in the *world model* describes the robot and the environment, facilitating sophisticated reasoning and task planning capabilities.

Despite these advancements, contact-rich tasks remain a complex endeavor, often challenging to fully encapsulate in predefined models. To overcome this, it is possible to allow robot to learn from experience and improve. This thesis presents an approach for robot control and learning based on behavior trees and reinforcement learning (RL). Our integration of robot skills, knowledge and planning with RL does not only enable robots to proficiently learn and execute contact-rich tasks but also allows for the seamless transfer of learned policies to real-world applications. In a comparison with state-of-the-art RL algorithms we show that this combination of planning and learning demonstrates markedly accelerated learning curves. Furthermore, we can demonstrate that the operators can formulate priors for the optimum to guide and speed up the learning process. An extension of this framework further enables robots to adapt to task variations without the need for relearning from scratch, showcasing the system's robust adaptability and potential for diverse industrial applications.

Keywords
Robot Skills, Reinforcement Learning, Task Planning,Knowledge Representation, Reasoning

Classification system and/or index terms (if any)

Supplementary bibliographical information

Language
English

ISSN and key title
1404 − 1219

ISBN
978-91-8039-884-8 (Print)
978-91-8039-885-5 (PDF)

Recipient's notes

Number of pages
286

Price

Security classification

Signature _____ Date 09.01.2024

Learning with Skill-based Robot Systems

Combining Planning & Knowledge Representation with Reinforcement Learning

by Matthias Mayr

LUND
UNIVERSITY

Funding information: This work was partially supported by the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by Knut and Alice Wallenberg Foundation.

Faculty of Engineering (LTH), Department of Computer Science

ISBN: 978-91-8039-884-8 (print)
ISBN: 978-91-8039-885-5 (pdf)
ISSN: 1404 − 1219
Dissertation 74, 2024
LU-CS-DISS: 2024-01

Printed in Sweden by Tryckeriet i E-Huset, Lund 2024

Dedicated to my family

Contents

List of Publications

This thesis is based on the following publications:

I **SkiROS2: A skill-based Robot Control Platform for ROS**

M. Mayr, F. Rovida, V. Krueger
IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Detroit, United States of America, 2023, pp. 6273-6280, DOI: 10.1109/IROS55552.2023.10342216.

II **Using Knowledge Representation and Task Planning for Robot-agnostic Skills on the Example of Contact-Rich Wiping Tasks**

M. Mayr, F. Ahmad, A. Durr, V. Krueger
IEEE 19th International Conference on Automation Science and Engineering (CASE), Auckland, New Zealand, 2023, pp. 1-7, DOI: 10.1109/CASE56687.2023.10260413.

III **Learning of Parameters in Behavior Trees for Movement Skills**

M. Mayr, K. Chatzilygeroudis, F. Ahmad, L. Nardi, V. Krueger
IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Prague, Czech Republic, 2021, pp. 7572-7579, DOI: 10.1109/IROS51168.2021.9636292.

IV **Skill-based Multi-objective Reinforcement Learning of Industrial Robot Tasks with Planning and Knowledge Integration**

M. Mayr, F. Ahmad, K. Chatzilygeroudis, L. Nardi, V. Krueger
IEEE International Conference on Robotics and Biomimetics (ROBIO), Jinghong, China, 2022, pp. 1995-2002, DOI: 10.1109/ROBIO55434.2022.10011996.

V **Learning Skill-based Industrial Robot Tasks with User Priors**

M. Mayr, C. Hvarfner, K. Chatzilygeroudis, L. Nardi, V. Krueger
IEEE 18th International Conference on Automation Science and Engineering (CASE), Mexico City, Mexico, 2022, pp. 1485-1492, DOI: 10.1109/CASE49997.2022.9926713.

VI **Learning to Adapt the Parameters of Behavior Trees and Motion Generators (BTMGs) to Task Variations**

F. Ahmad, **M. Mayr**, V. Krueger
IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Detroit, United States of America, 2023, pp. 10133-10140, DOI: 10.1109/IROS55552.2023.10341636.

VII **BeBOP – Combining Reactive Planning and Bayesian Optimization to Solve Robotic Manipulation Tasks**

J. Styrud, **M. Mayr**, E. Hellsten, V. Krueger, C. Smith
Submitted to the International Conference on Robotics and Automation (ICRA), Yokohama, Japan, 2024

All papers are reproduced with permission of their respective publishers.

Publications not included in this thesis:

VIII **EzSkiROS: A Case Study on Embedded Robotics DSLs to Catch Bugs Early**

M. Rizwan, R. Caldas, C. Reichenbach and **M. Mayr**
IEEE/ACM 5th International Workshop on Robotics Software Engineering (RoSE), Melbourne, Australia, 2023, pp. 61-68, DOI: 10.1109/RoSE59155.2023.00014.

IX **Flexible and Adaptive Manufacturing by Complementing Knowledge Representation, Reasoning and Planning with Reinforcement Learning**

M. Mayr, F. Ahmad, V. Krueger
IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Detroit, United States of America, 2023, Workshop on Robotics & AI in Future Factory

X **A C++ Implementation of a Cartesian Impedance Controller for Robotic Manipulators**

M. Mayr and J. M. Salt-Ducaju
Submitted to the Journal of Open-Source Software (JOSS)

XI **Combining Planning, Reasoning and Reinforcement Learning to solve Industrial Robot Tasks**

M. Mayr, F. Ahmad, K. Chatzilygeroudis, L. Nardi, V. Krueger
IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Kyoto, Japan, 2022, Workshop on Trends and Advances in Machine Learning and Automated Reasoning for Intelligent Robots and Systems

XII **Research Challenges in Skill-based Reinforcement Learning of Industrial Manufacturing Tasks**

M. Mayr, V. Krueger and J. Malec
Swedish AI Society (SAIS) 34th annual Workshop, Stockholm, Sweden, 2022, track for ongoing Ph.D. projects

XIII **Generalizing Behavior Trees and Motion-Generator (BTMG) Policy Representation for Robotic Tasks Over Scenario Parameters**

F. Ahmad, **M. Mayr**, V. Krueger
International Joint Conferences on Artificial Intelligence (IJCAI), Vienna, Austria, 2022, Workshop on Bridging the Gap Between AI Planning and Reinforcement Learning

Acknowledgements

It's done. After five years at Lund University and 4.5 months at Bosch Research these acknowledgements are the last thing for the thesis to do. I hope I don't forget anyone and if I did, bear with me — it was not on purpose and there were a lot of people to think of.

Like pretty much all PhDs, it has been a journey with its twists and some unexpected turns. When I started my PhD I had no knowledge about using manipulators, practically implementing controllers and using RL algorithms. I learned a lot and hopefully also gave a lot to other people. However, the journey that brought me to this point started much earlier, so I want to start where the first credits are due.

My interest in technology and engineering was nurtured mainly by my father. Maybe even more passively than actively, because I saw that this field always brought new problems to solve and wouldn't just be monotonous work. Then after having opted for studies in electrical engineering at Karlsruhe Institute of Technology, being about three terms into my bachelor's degree, I got my hands on a student job posting in robotics. It was from a group with professor Dillmann which worked on decision making and mobile manipulation. Thanks again to Sven Schmidt-Rohr for this opportunity and Gerhard Dirschl, Rainer Jäkel and Pascal Meissner for the great environment, discussions and opportunities to learn. This was really where my robotics career started.

Through Sven I also had the opportunity to do my bachelor's thesis in robotics. He was the one who set up the contact with Roland Philippsen in Halmstad where I worked with mobile robots in ambient-assisted living. Moreover, Sven also supported my internship during the master studies, where I went to Siemens Corporate Technology in Berkeley. It was there where I learned about the knowledge representation techniques that were also used in this thesis. Thanks to Jack Hodges and Simon Mayer for the valuable lessons.
Thanks also to my master thesis supervisor Sascha Wirges - without you asking if I want to start a PhD there, I am not sure if I would have aimed for one.

Here we are in 2018 in Lund. The main role goes, of course, to my supervisor Volker Krueger. Thank you not only for choosing me from a large pool of candidates, but especially for your time, your ideas and your interest in the work. I really enjoyed that so many things, such as trips or the sabbatical, that could be easy, were also easy and I did not have to worry. Thank you for many interesting discussions, for challenging my plans and ideas, but at the same time being able to be convinced. Thanks to Jacek Malec for always having

a sympathetic ear and for the many questions you raised. In the end, we did not work together too much scientifically, but only because there was not enough time (and I did too much RL). Thank you for handling administrative matters, so that we don't have to worry about them.

Within the group it's of course Faseeh Ahmad I shared the most working hours and papers with. Thank you for your input, your questions and your commitment. Luigi Nardi for joining LU and not only bringing your own expertise, but soon also your PhD students Carl Hvarfner and Leonard Papenmaier and postdocs Kenan Šehić and Erik Hellsten. Without all of you, this thesis would not have been the same. Not to forget Konstantinos Chatzilygeroudis: the postdoc the group never had. Your expertise in robot control, robot learning, simulation and pretty much everything else really helped to bootstrap this research.
Thanks to my fellow robotics PhD students in the group, Alexander Duerr, Momina Rizwan, Johan Oxenstierna, Ayesha Jena, Simon Kristoffersson Lind and Hampus Åström. Thanks to Elin Anna Topp for many interesting discussions, valuable tips and finally also for allowing me to use your sailboat. That definitely added a lot to my time in Lund.

Fortunately enough, the robotics research is hosted in the *Robotlab* which is shared between the departments of automatic control and computer science. The first person who comes to my mind is, of course, Anders Robertsson. He was the good heart in the robotlab. Exceptional expertise together with strong commitment and always a sympathetic ear is hard to find. He left us much too early and also left a big hole behind.
I want to thank Björn Olofsson for his advice and background knowledge. Julian Salt Ducaju for the robotics exchange, the student supervision and controller advice. Francesco Rovida, Bjarne Grossmann and David Wuthier for sharing your knowledge and providing a basis of skills and experiences.

Thanks to all our administrative staff in the department for providing a smooth working environment and being incredibly helpful. Thanks to WASP for providing such a great research environment and networking opportunities. WASP was one reason for taking this PhD position and it proved to be incredibly valuable. Niels van Duijkeren and Robert Krug for the organization of my sabbatical at Bosch Corporate Research. And thanks to everyone there for the exchange and the great time I had there.

Five years are a long time and I want to thank my partner Marianne for the incredible support. Especially during the pandemic it has not been easy.
It would also have been challenging to go through that time without some life

in addition to work. Many colleagues not only contributed to great lunch and fika conversations, but also enriched my life besides work. Among the ones not mentioned yet, there are at least Idriss Riouak, Gareth Callanan, Ilayda Yaman and Haorui Peng, but also many many more and it's impossible to name everyone.

Last but not least, I want to thank the opponent Michael Beetz and the committee, Karinne Ramirez-Amaro, Petter Ögren and Todor Stoyanov, for accepting to be a part of the jury.

Matthias Mayr
Lund, January 2024

Popular Summary in English

For years, robots in factories have been like skilled but inflexible workers, highly capable at their specific tasks but unable to adapt or learn new ones. They have been programmed in complex, manufacturer-specific languages, making them highly specialized but not very versatile. However, as we step into the era of the fourth industrial revolution, the demands on these robotic workers are changing. Now, they need to be more like Swiss Army knives: versatile, adaptable, and ready for new challenges.

Here the concept of "robot skills" comes in - think of these as apps for robots. Just like you download apps on your smartphone to give it new capabilities, robot skills are programs that can be mixed and matched to teach robots new tasks. These skills are not just one-trick ponies; they're reusable and can be tweaked to suit different needs, making robot programming much more flexible.

But how do these robots know what to do and where to do it? This is where the "world model" comes in. It is like a map and a guidebook rolled into one, containing information about the robot's environment and capabilities. For instance, if a robot is asked to fetch a cup of coffee, the world model helps it understand where the kitchen is, where the coffee machine is, and how to operate it. It is like giving the robot a basic understanding of its world and how to navigate it.

Now, teaching a robot new skills is one thing, but how about teaching it to learn on its own? This is where our research takes an exciting turn. We are using reinforcement learning - a way for robots to learn through trial and error, much like how humans learn. The robot tries different approaches, learns from feedback, and gradually improves its strategy. It is like a baby learning to walk, stumbling and getting better over time.

But here is the catch: teaching a robot to learn is not straightforward. Our research is focused on making this learning process easier and more efficient, even for those who are not robot experts. We are finding ways to speed up this learning, like giving the robot 'hints' or using simulations to practice before trying things out in the real world. Plus, we are teaching robots to handle variations of tasks without starting from scratch every time.

Our approach, which combines planning, robot skills, and learning, can be much faster than traditional methods. We are testing it in simulated environments and on real robots, focusing on tasks that involve a lot of contact, like manipulating objects or wiping surfaces.

In summary, we are not just teaching robots new tricks; we enable them to learn and adapt, making them ready for the ever-changing demands of the modern industrial world.

Populärvetenskaplig Sammanfattning på Svenska

I åratal har fabriksrobotar varit som skickliga men oflexibla arbetare - kapabla att utföra sina specifika uppgifter men oförmögna att anpassa sig eller lära sig något nytt. Ofta har de programmerats i komplexa tillverkarspecifika språk vilket är bra specialisering men brister när det kommer till mångsidighet. Men nu när vi tar tar steget in i den fjärde industriella revolutionenförändras kraven. Nu behöver robotarbetarna vara mer som schweiziska arméknivar: mångsidiga, anpassningsbara och redo för nya utmaningar.

Här kan vi införa begreppet "robotfärdigheter". Tänk på dem som appar för robotar. Precis som de appar du laddar till din smartphone, är robotfärdigheter program som kan kombineras för att lära robotar att lösa nya uppgifter. Dessa färdigheter enkla, återanvändbara och kan anpassas för olika behov, vilket gör robotprogrammeringen enklare och mer flexibel.

Men hur vet robotarna vad de ska göra och var de ska göra det? Det är här "världsmodellen" kommer in i bilden. Den är som en karta och en guidebok i ett, och innehåller information om robotens miljö och förmågor. Om en robot till exempel ombeds att hämta en kopp kaffe hjälper världsmodellen den att förstå var köket är, var kaffemaskinen finns och hur den ska användas. Det är som att ge roboten en grundläggande förståelse för sin omvärld och hur den ska navigera den.

Att lära en robot nya färdigheter är en sak, men vad sägs om att lära den att lära sig på egen hand? Här tar vår forskning en spännande vändning. Vi använder Reinforcement Learning - ett sätt för robotar att lära sig genom trial-and-error, ungefär som människor lär sig. Roboten prövar olika tillvägagångssätt, lär sig av feedback och förbättrar gradvis sin strategi. Det är som när en bebis lär sig gå, snubblar och blir bättre med tiden.

Men här är haken: att lära en robot att lära sig är inte okomplicerat. Vår forskning är inriktad på att göra inlärningsprocessen enklare och mer effektiv - även för dem som inte är robotexperter. Vi hittar sätt att påskynda inlärningen, som att ge roboten "tips" eller använda simuleringar för att öva innan man testar saker i den verkliga världen. Dessutom lär vi robotar att hantera variationer av uppgifter utan att behöva börja om från början varje gång.

Vår metod, som kombinerar planering, robotfärdigheter och inlärning, är ofta betydligt snabbare än traditionella metoder. Vi visar detta i simulerade miljöer och på riktiga robotar, med fokus på uppgifter som innebär mycket kontakt, som att manipulera föremål eller torka av ytor. Sammanfattningsvis lär vi inte

bara robotarna nya trick utan vi gör det möjligt för dem att lära sig och anpassa sig, vilket gör dem redo för de ständigt föränderliga kraven i den moderna industriella världen.

List of Figures

Learning with Skill-based Robot Systems

Combining Planning & Knowledge Representation with Reinforcement Learning

AI is the new electricity.
— Andrew Ng

1 Introduction

1.1 Motivation

In the past there have been three industrial revolutions that transformered how we work and live. The first industrial revolution was the introduction of mechanical production facilities, powered by water and steam [Sch17, p.11]. The second industrial revolution was the introduction of mass production with the help of electrical energy and assembly lines. The third industrial revolution, the digital revolution, was the introduction of electronics and information technology to further automate production [Sch17, p.11]. Now, we stand at the brink of the fourth industrial revolution, or *Industry 4.0*, characterized by the fusion of the physical and digital realms through cyber-physical systems [Sch17, p.12]. This era is not just about technological advancements; it represents a paradigm shift towards agile robotics and flexible manufacturing systems, promising unprecedented levels of customization and efficiency in production [Sch17, p.12].

While the previous industrial revolutions democratized access to goods and services, Industry 4.0 aims to revolutionize how these goods are produced, emphasizing customization, smaller batch sizes, and innovative products previously deemed unfeasible [Sch17, p.12]. A key barrier in the past has been the rigidity and high cost of reconfiguring production lines. However, the emerging flexible and agile systems in Industry 4.0 promise to overcome these challenges, potentially revitalizing production in advanced economies by reducing the reliance on labor costs. With these advancements, the fourth industrial revolution also has the potential to hold or even bring back production to advanced economies, as labor costs play a less crucial role.

A cornerstone of this revolution can be the development of more sophisticated, autonomous, and intelligent robotic systems with stronger vertical integration [Kru+19]. These systems must not only be capable of understanding and interacting with their environment but also be capable of learning from experiences and adapting their behavior accordingly. The concept of 'robot skills' — parametric procedures that alter the world's state — can be seen as central to this evolution [Bøg+12]. Yet, how these skills are optimally formulated and utilized to achieve the necessary flexibility remains an open and critical question [Alb+23].

Moreover, the challenge extends to the realm of robot learning, particularly in the context of contact-rich tasks. Traditional approaches often require extensive manual tuning and suffer from inefficiencies and opaque results, which hinders their industrial applicability [SKK22]. In contrast, the wealth of environmental, component, and process knowledge available in industrial settings presents a unique opportunity. This knowledge, if effectively integrated into the learning process, can significantly enhance efficiency and lead to more desirable outcomes. However, achieving this integration is far from trivial and remains largely unexplored.

This thesis aims to address these gaps by exploring approaches to formulating and implementing robot skills and learning processes. By leveraging the rich knowledge inherent in industrial environments, we seek to develop more adaptable, efficient, and transparent robotic systems, paving the way for the full realization of Industry 4.0's potential.

1.2 Problem Statement

The advent of Industry 4.0 demands robot systems that are not only flexible and adaptive but also easily reconfigurable to accommodate frequently changing

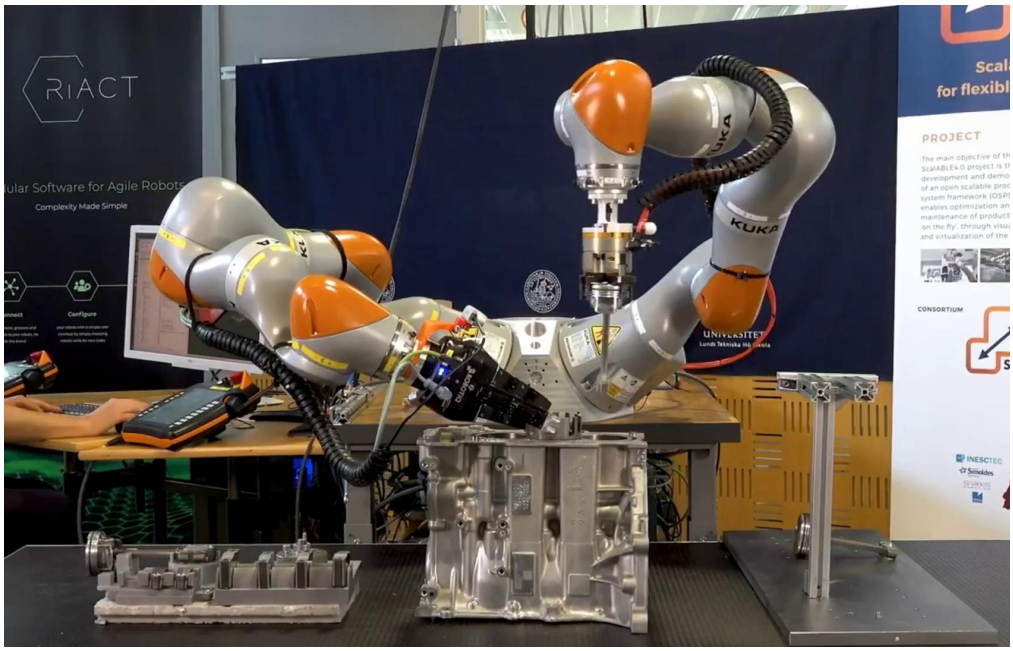


Figure 1: A dual-arm piston insertion task. The task is to pick up a piston and to insert it into the engine block. The execution is partially adaptivelly parameterized. However, for example the parameters for the insertion are manually tuned.

tasks and diverse environments [Ped+16]. However, the prevalent reliance on robot manufacturer-specific solutions presents a significant barrier. These systems often lack the necessary flexibility and adaptability, impeding the realization of truly reconfigurable robotic systems.

To overcome these limitations, scalable architectures for robot decision-making and control are essential [Alb+23]. A deep integration with the proprietary solutions of individual manufacturers hinders the development of such reconfigurable systems. Instead, a shift towards robot-agnostic approaches, utilizing robot skills in conjunction with knowledge representation, is an attractive option. This approach, encapsulating knowledge in a *world model*, enables systems to self-configure and self-parameterize within the model's framework, facilitating the reuse and sharing of skills among operators.

However, this approach is not without its challenges. Semantic modeling of the world is complex and initially resource intensive. Incomplete domain knowledge can lead to errors and reduced flexibility, particularly in contact-rich tasks where predicting interactions and resulting forces is difficult. This makes it challenging

to design model-based reasoners that can be used to completely parameterize such tasks. Previously, this problem was often solved by manually tuning the parameters of skills, such as the wiggle motion of the piston insertion in Figure 1. However, this is a time-consuming and expensive process that is not flexible and requires expert knowledge.

Machine learning (ML), and specifically reinforcement learning (RL), offers a potential solution to these challenges. RL allows for parameterization through direct environmental interaction, bypassing the need for manual tuning or complete model-based reasoning. However, formulating RL problems is inherently complex [SB18, p.469]. Many RL algorithms are intricate and demand considerable expertise. Designing effective reward functions and formulating problems to meet operational and safety expectations while ensuring the interpretability of the resulting policies is a formidable task. This lack of interpretability poses risks in terms of safety, error mitigation, and production process assurance, making many algorithms impractical for real-world, high-stakes industrial environments [Elg+23].

RL algorithms are often too inefficient to be used on real systems. This is especially true for contact-rich tasks that have complex interactions with the environment. While simulation-based learning offers a partial solution, accurately modeling contact-rich tasks for effective transfer to real systems remains a significant hurdle [Elg+23].
Additionally, the use of RL in industrial settings often overlooks the integration of existing knowledge, such as symbolic data, historical task traces, or expert insights, which could potentially accelerate learning and improve policy outcomes.

The development of flexible, adaptive, and easily reconfigurable robotic systems for Industry 4.0 faces multifaceted challenges. These include the complexity of semantic world modeling, the limitations of current robot control architectures, and the intricacies of implementing effective and interpretable RL strategies. Addressing these issues is crucial to advance robotic capabilities in the rapidly evolving landscape of industrial automation.

1.3 Research Objectives

The primary goal is to develop methodologies that enable robot systems to self-configure and autonomously learn behaviors. Starting from model-driven approaches, the focus is on exploring RL to learn how to perform tasks. The focus is on tasks that involve contacts and interaction with the environment. The aim is to apply these learned behaviors effectively on real robot systems.

In this context we identify the following concrete research objectives:

Accessible Learning: A key objective is to simplify the formulation of RL-based learning problems, making it accessible to individuals who are not experts in RL. This involves creating an approach where operators can easily input their experiences and insights into the learning process. The goal is to make the formulation of learning problems straightforward and user-friendly.

Ensuring Transparency and Interpretability: While the learning process might inherently involve complex computations (akin to a 'black box'), it is crucial that the resulting policies are interpretable. This objective focuses on developing methods that ensure that the outcomes of the learning process are transparent, allowing for easy mitigation and adjustment. This interpretability is vital for practical applications, especially in scenarios where safety and reliability are paramount.

Integrating Knowledge into Learning Processes: This research aims to integrate existing knowledge into the RL process. This includes leveraging symbolic knowledge, historical data, and expert experience to improve the efficiency of the learning process and improve the quality of the resulting policies. The integration of this knowledge is expected to accelerate learning and lead to more effective robotic behaviors.

Modular Learning: Learning of subtasks within their specific environments should be possible. The goal is to provide mechanisms to encode typical constraints and consider safety rules, ensuring that the learning process and the resulting behaviors adhere to operational constraints.

Thesis Structure

The first part of this thesis is the *Kappa*, the introduction to the research field and the scope of the work presented in this thesis.

This introduction starts by providing the background on robot programming, motion representations for manipulators and contact-rich tasks in Section 2. Section 3 delves into knowledge representation and reasoning in robotics, setting the stage for understanding its application in this research. In Section 4, the

concept of robot skills is explored, along with the integration of knowledge into these skills and the use of behavior trees (BTs) as a representation method. The thesis then transitions to task-level planning for longer-horizon goals in Section 5. Learning methods for robot skills, particularly reinforcement learning (RL), are introduced in Section 6. The concrete approach taken in this research is presented in Section 7. This section is followed by the final thoughts and conclusions of the Kappa.

The second part of the thesis comprises the seven individual papers that form the core of the research contributions. These papers are presented in full, providing detailed insights into the methodologies, experiments, results and discussions that are central to the research objectives established in the first part of the thesis.

2 Robot Programming and Motion Representations

Robot capabilities can be developed through three well-established approaches [KCC13]:

1. Direct programming
2. Imitation learning (IL) or learning from demonstration (LfD)
3. Reinforcement learning (RL).

RL is introduced in Section 6 and LfD in Section 6.1.

This section provides the background on the state of robot programming in industry. The second part introduces such motion representations for robot arms, ranging from easier representations to dynamical systems. Finally, the specific challenges posed by contact-rich tasks are highlighted.

2.1 Robot Programming

Traditionally, the most common way to program a robot is to use a programming language that is specific to the robot manufacturer and that is commonly proprietary [BM03; Bru+07]. Such a language is often on a quite low level, has its own unique syntax and is not very intuitive [SK16, *RAPID* example on p.1412]. To support existing customer solutions, the backward compatibility of such a language is important and therefore the language is often not very modern [BM03; Bru+07]. This makes these systems difficult to use and the competences are often taught in week-long professional training programs. This situation not only binds customers to specific hardware, but also locks them into the programming ecosystem of that hardware, due to the significant investment in developing programming competencies for a particular system [BM03].

Especially with the availability of colaborative robots, so-called *cobots*, manufacturers offer different methods to program their robot. For example, Universal Robots (UR) provides five different ways to create robot motions [Ryt23]: 1) visual programming with the teaching pendant. 2) The specific language *URScript*. 3) Capability blocks known as *URCaps*. 4) External control from another machine via an Ethernet connection and finally 5) an official robot operating system (ROS)[1] driver.

[1] ROS is a middleware and a set of libraries for robotics. It is especially popular in academia, but also receives more adoption by industry.

Figure 2: The teaching pendants of three different robot manufacturers. From left to right: *ABB*, *KUKA* and *Universal Robots (UR)*.

Other vendors offer similar, yet distinct, solutions for robot programming. The manufacturer *ABB*, for instance, facilitates visual programming using a teaching pendant and has developed its own language, *RAPID*, complemented by a visual programming environment called *RobotStudio*. Additionally, *ABB* robots can receive low-level commands through external guided motion (EGM).
KUKA, on the other hand, does not provide programming via a teaching pendant. For their *iiwa* robots shown in Figure 1, programming is done in Java using a computer and the *Sunrise Workbench* environment. They also support external motion input through the fast robot interface (FRI). Notably, ROS drivers for the *iiwa* robots are available, but they are developed and maintained by the academic community.

Among these methods, the most accessible for beginners is typically visual programming with a *teaching pendant*. Teaching pendants, as shown in Figure 2, are devices attached to the robot that feature a display that allows users to manually move the robot. The robot's positions can often be recorded and the movements can be replayed later. This method is intuitive to program basic robot movements, but lacks flexibility. It requires the physical presence of an operator and is limited to programming the robot in free space, without contact with the environment. Consequently, tasks such as part insertion or surface wiping, crucial in many industrial applications, cannot be easily programmed using this method.

Figure 3 presents the programming environments of two different robot manufacturers, highlighting the differences in their approaches. For example *KUKA* provides a simulation only as an additional product, while *ABB* includes it. Each of these environments necessitates specific training, and transitioning between different programming systems is not straightforward.

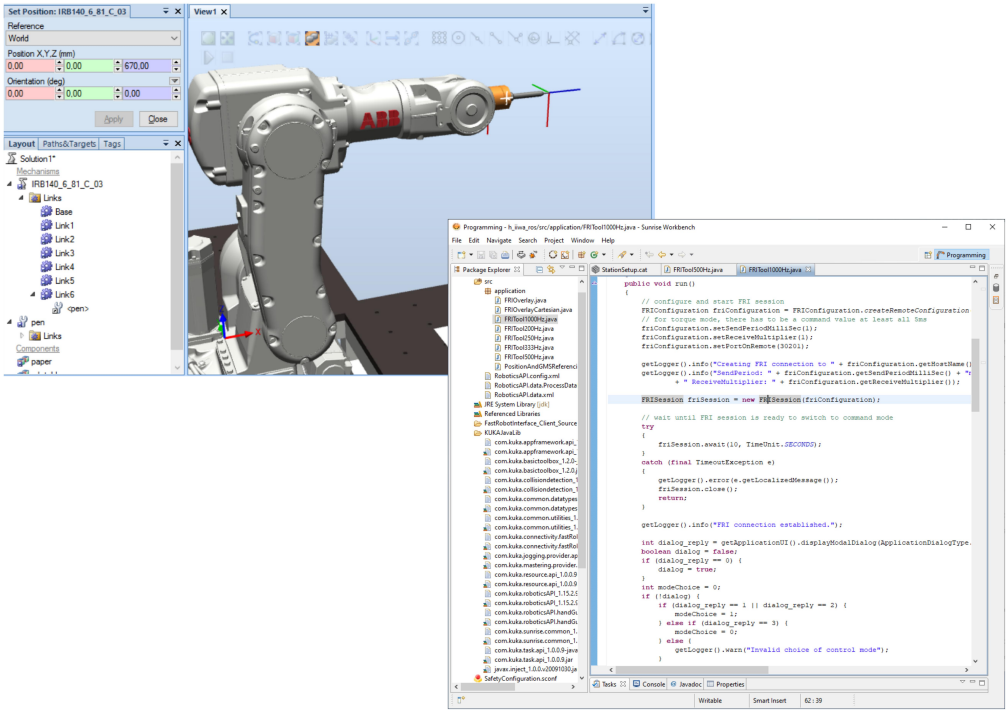


Figure 3: Robot programming environments of *ABB, RobotStudio* on the left and *Sunrise Workbench* by *KUKA* on the right. UR does not provide a programming environment.

The primary advantages of conventional industrial robot programming solutions lie in their established reliability and safety [BM03]. These systems are often deployed in safety-critical applications where manufacturers can assure that the robot will operate safely, especially in scenarios where it works in close proximity to human operators. This reliability is a crucial factor in typical industrial settings, where consistent and safe operation is paramount.

However, these traditional programming solutions also come with notable disadvantages. One significant drawback is their lack of flexibility. Additionally, they tend to be costly, partly due to the absence of accessible simulation environments that can accurately simulate physical interactions for contact-rich tasks. This limitation means that robot programs often cannot be written, tested, and refined in a simulated environment. As a result, testing and development must occur on the actual robot, leading to increased costs because the robot is occupied for the duration of the testing.

In the realm of industrial robotics, the company *ArtiMinds* offers a constrasting approach. They provide a robot-agnostic programming environment enabling

users to program robots visually, with the added flexibility of executing these programs on various robot models. This versatility is achieved by not writing the robot program in a robot-specific language. Instead, robot-specific code is generated as an output [Art19]. This approach represents a significant shift from traditional methods, offering enhanced flexibility and potentially reducing the costs associated with robot programming.

2.2 Motion Representations for Robotic Arms

A crucial aspect of actuated robot arms is the representation of their motion. This involves determining how the motion and its goal are conceptualized and executed. Typically, the goal is defined in Cartesian space, which means that it is specified as a position and orientation of the end effector (EE). The motion itself can be represented in two primary spaces: joint space or Cartesian space [Sic09]. In joint space, the motion is typically represented as a trajectory of the joint angles, while in Cartesian space, it is represented as a trajectory of the end-effector pose. However, since robots are ultimately controlled in joint space, any motion defined in Cartesian space must be converted using an inverse kinematics solver or a controller.

It is also vital to distinguish between the trajectories and the control strategy. The trajectory itself refers to the desired time-parameterized motion path of the robot, whereas the control strategy is the method employed to follow this trajectory [Sic09].

In the realm of industrial robotics, programming point-to-point motions, executed either as Cartesian linear motions or joint motions, is still prevalent. This approach, while straightforward and easy to program, lacks flexibility and is not well-suited for contact-rich tasks.

A popular formulation for motion primitives in robotics are dynamic movement primitives (DMPs). These are implemented through stable non-linear dynamical systems that are inspired by biological systems [Sch06a; Sch06b]. DMPs offer an elegant formulation that guarantees convergence to a specified target. They can react to external perturbations and generalize to different goals [Sav+23]. There is a vibrant community around DMPs and many extensions have been proposed [Kar+17; KRJ17; KRJ18]. Most notably in this context are compliant movement primitives [Pet+14; Den+16] that are suitable for contact-rich tasks. DMPs can be efficiently learned from data and are often utilized in imitation learning (IL). A comprehensive survey and a tutorial on DMPs are available in [Sav+23].

Another significant motion representation involves the use of Gaussian mixture models (GMMs) with dynamical systems [KB11]. Like DMPs, these systems ensure global asymptotic stability and can react to perturbations while generalizing to different goals. The authors introduce a learning method called *stable estimator of dynamical systems* to learn the parameters of the dynamical systems. The method is an optimization approach to statistically encode a motion as a first-order nonlinear ordinary differential equation with the use of Gaussian mixtures.

Both DMPs and dynamical systems with GMMs are robust methods that can impliclity encode motions and are learned from demonstrations. While their ability to be learned can often be a low entry barrier, it is not always possible to provide such demonstrations. Especially in industrial environments with frequently changing tasks, it can become cumbersome to provide demonstrations for every new task and additional infrastructure for storing and re-using data is necessary [BCD16].

An alternative approach to generate motions are constrained-based methods. These methods are designed to produce motions that adhere to a set of predefined constraints. There are two primary approaches: 1) constrained-based task space control [ADS14] and 2) constrained sampling-based control. In the former, a differentiable function is used to define a task space and constraints are enforced on the derivative [ADS14]. A recent example application is the combined online control of a mobile platform and a robot arm with compliant motions [Cal+22]. A collection of sampling-based algorithms, the second category, is implemented in the open motion planning library (OMPL). OMPL is a widely used library for motion planning [SMK12] that is integrated in the *MoveIt!* framework and therefore also in the ROS ecosystem. A recent development in this area is *Giskard* [SBB22], which, like MoveIt, produces constrained and smooth trajectories. However, both OMPL and *Giskard* are offline approaches that produce trajectories that are executed by a controller which limits their reactivity. Typically these controllers are also a stiff position controllers that are not suitable for contact-rich tasks. In the task frame formalism (iTasC) formalism [DS+07], the desired motion and sensor values are declaratively formulated as constraints on the kinematic chain.

In [Rov+18], the combination of BTs with motion generators (MGs) is suggested. This representation is not only suitable for contact-rich tasks without requiring explicit teaching, but it also supports the integration of knowledge. A more detailed introduction will be provided in Subsection 4.3.

2.3 Contact-rich Tasks

While moving a robot arm in free space is a well-studied problem, many industrial tasks involve complex interactions with the environment. Examples include assembly tasks that require the insertion of parts, pushing components into fixtures, or wiping surfaces [SKK22]. In these scenarios, the robot arm must engage in physical contact with its surroundings, presenting unique challenges. Predicting the contact points and the resulting forces can be difficult, complicating the task of designing model-based reasoners capable of fully parameterizing these tasks.

Traditionally, addressing these challenges involved manually tuning the behavior parameters like in [RGK17]. However, this approach is not only time-consuming and costly, but is also inflexible and reliant on expert knowledge.

Most standard robot programming solutions are not well suited for contact-rich tasks, particularly those used in typical industrial settings. Solutions that offer compliant control, such as impedance control or force control, are not always available or applicable. Additionally, these conventional methods often lack compatibility with ML techniques, which can be crucial for adapting to the dynamic nature of contact-rich environments.

Simulating contact-rich tasks for the purpose of training robots is another area fraught with difficulties. Accurately modeling physical interactions in a simulation environment is a complex endeavor. Consequently, much of the research in RL for contact-rich tasks falls short of transferring the learned policies to real-world robots [Elg+23]. This gap between simulation and practical application remains a significant hurdle in the field [Elg+23].

3 Knowledge Representation and Reasoning

In the realm of artificial intelligence (AI), knowledge representation plays a pivotal role in encoding information about the world in a format that is both machine-interpretable and human-readable [CA22, p.428]. This dual nature of knowledge representation is crucial, as it serves to bridge the gap between human understanding and machine processing [Cel20]. By translating complex real-world data into a structured format, it allows machines, such as robots, to comprehend and interact with their environment more effectively [SK16, p.330].

Knowledge bases are central to this process, acting as repositories that collate and interconnect information from diverse sources. These bases not only store knowledge, but also define the relationships between different pieces of information, enabling comprehensive reasoning about the world [SK16, p.338].

This section first introduces the concept of knowledge representation and ontologies. Then, reasoning is introduced as a way to use the knowledge to answer questions about the world.
Together, this contributes to a world model that a robot system can use to put its capabilities into a context to solve underdetermined tasks. In Subsection 4.4, it will be shown how this knowledge can be integrated into robot skills.

3.1 Knowledge Representation

Knowledge representation in AI is a fundamental process for encoding information about the world in a machine-interpretable format [CA22]. This process is essential to store and manipulate complex structured data. Knowledge representation formats are typically designed to be human-readable, facilitating an effective interface between human operators, automated processes, and AI systems.

Central to knowledge representation are facts or assertions about various entities within a system [CA22]. A critical aspect of formal knowledge representation systems is the underlying assumption about the completeness of the knowledge. Under the *open-world assumption*, the lack of evidence for a fact does not imply its negation [RND10, p.417]. For example, if a robotic system lacks data indicating the presence of a table in a room, it does not conclusively mean the table is absent. In contrast, the *closed-world assumption* posits that if a fact is not known to be true, it is considered false [RND10, p.417]. An example of this can be seen in employee databases, where the absence of a person's record implies

that they are not an employee of the company. The partially open-world assumption offers a nuanced approach, assuming general incompleteness of knowledge, but considering it complete within specific domains or contexts [RSN16].

The representation of knowledge encompasses various methodologies, each offering different levels of expressivity. However, there exists an inherent trade-off between the expressivity of a knowledge representation system and its computational efficiency. More expressive systems, while capable of representing complex relationships and abstractions, tend to be more challenging to use [Baa+07, p.43].

First-order logic represents one of the most expressive forms of knowledge representation, allowing for detailed and nuanced descriptions of relationships and entities [Baa+07, p.6]. However, its complexity often results in slower processing by reasoning systems. On the other end of the spectrum, *propositional logic* offers a less complex and more computationally efficient alternative, albeit at the cost of reduced expressivity, particularly in representing inter-entity relationships. *Description logics* ([Baa+07]) strike a balance between these two extremes, offering greater expressivity than propositional logic but with less complexity than first-order logic, thereby enabling more efficient processing by reasoning systems [Baa+07, p.43].

One fundamental approach to knowledge representation is the utilization of graphs in which nodes represent entities and edges denote the relationships between these entities. A prevalent format for such representation is the resource description framework (RDF), a standard model widely adopted for data interchange on the web [HKR09].

In RDF, knowledge is structured as a directed graph composed of triple statements:

subject predicate object

The subject and the object represent nodes in the graph and the predicate is an edge. The predicates are also referred to as traits, properties or aspects. The nodes within this framework can be either entities or literals [HKR09]. Entities represent tangible or conceptual elements in the world, such as processes, robots, humans, or physical objects. Both entities and predicates are uniquely identified using uniform resource identifiers (URIs) [HKR09, p.21]. Literals, on the other hand, are basic data types such as strings or numbers [HKR09, p.22]. It is important to note that in RDF triples, the subject must always be an entity, whereas the object can be either an entity or a literal [HKR09, p.24]. Addition-

```
1  scalable:Ur5-2 a scalable:Ur5, owl:NamedIndividual ;
2    rdfs:label "scalable:ur5"^^xsd:string ;
3    skiros:BaseFrameId "cora:Robot-1"^^xsd:string ;
4    skiros:CartesianGoalAction
         "/cartesian_trajectory/goal_action"^^xsd:string ;
5    skiros:OverlayMotionService
         "/cartesian_trajectory/overlay_motion"^^xsd:string ;
6    skiros:CartesianStiffnessTopic
         "/cartesian_param_filter/stiffness_goal"^^xsd:string ;
7    skiros:CartesianWrenchTopic
         "/cartesian_param_filter/force_goal"^^xsd:string ;
8    skiros:CompliantController "compliance_controller"^^xsd:string ;
9    skiros:JointConfigurationController
         "scaled_pos_traj_controller"^^xsd:string ;
10   skiros:DiscreteReasoner "AauSpatialReasoner"^^xsd:string ;
11   skiros:FrameId "scalable:Ur5-2"^^xsd:string ;
12   skiros:LinkedToFrameId "ur5e_base_link"^^xsd:string ;
13   skiros:MoveItGroup "manipulator"^^xsd:string ;
14   skiros:MoveItReferenceFrame "ur5e_base_link"^^xsd:string ;
15   skiros:MoveItTCPLink "ur5e_tcp_link"^^xsd:string ;
16   skiros:hasA scalable:WsgGripper-3 .
```

Listing 1: An excerpt from the semantic description of a *UR 5e* robot in the *Turtle* format. It includes some of the necessary knowledge to parameterize motion skills and perform spatial reasoning, such as the coordinate frames in lines 14-15. In line 16 the relation to the WM entry of the gripper is defined. That entry has its own description, such as communication ports or capabilities.

ally, the subject or object cannot be unconnected and the predicate cannot be a literal.

When multiple triples are combined, they form a labeled, directed multigraph, characterized by nodes with multiple incoming and outgoing edges. RDF data can be serialized in various formats, with *Turtle* [HKR09, p.25] and *RDF/XML* [HKR09, p.27] being the most common ones. An excerpt of the knowledge representation of a robot system can be seen in Listing 1.

Knowledge in RDF is typically stored in knowledge bases, with each base usually dedicated to a specific domain. However, it is also feasible to construct knowledge bases encompassing multiple domains. An illustrative example of this is *Wikidata* [VK14], which aims to aggregate all data relevant to *Wikipedia*. These knowledge bases can be interconnected to create a comprehensive knowledge graph, facilitating the reuse of knowledge and enabling complex queries about the world [CA22].

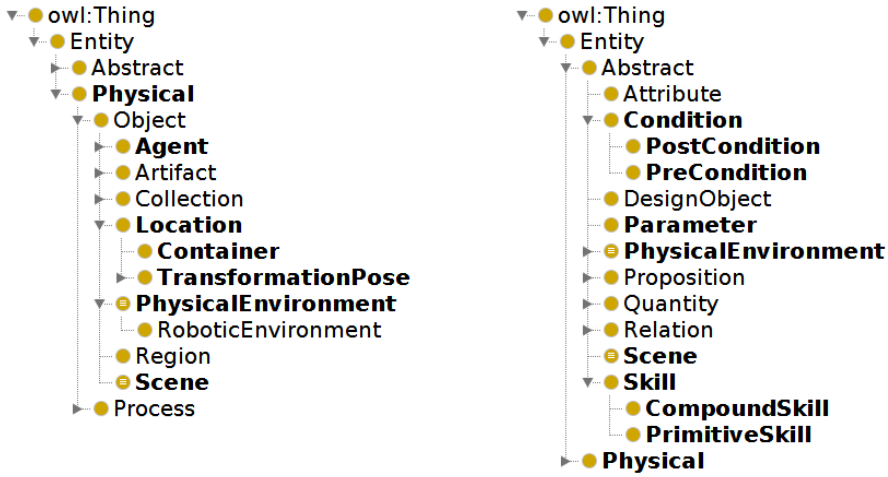


Figure 4: Two excerpts from the SkiROS2 ontology. Physical objects like "robot" and "container" are formalized. Furthermore, abstract concepts like "skills" and "conditions" are defined.

3.2 Ontologies

In computer science, an ontology is a taxonomy of types, properties and relationships between entities within a specific domain.[Baa+07]. It represents a formal and explicit specification of a set of concepts and the interconnections between these concepts [Gru93].

Ontologies are broadly categorized into two categories: upper ontologies and domain ontologies. Upper ontologies are general-purpose ontologies that are not specific to a particular domain [SK16, p.333]. They describe general concepts such as space, time or matter. An example for an upper ontology is the suggested upper merged ontology (SUMO) [NP01] that defines terms such as "agent". Domain ontologies are ontologies that are specific to a field [SK16, p.333].

To represent ontologies, various standards are employed, including the resource description framework schema (RDFS) and the Web Ontology Language (OWL). RDFS is a an extension of RDF. It is a simple, yet powerful ontology language that extends RDF by adding basic vocabulary to describe properties and classes. OWL is a more expressive ontology language that is also based on RDF. In addition to the vocabulary of RDFS, it adds more vocabulary to describe properties and classes. It also adds more vocabulary for describing relations between classes and properties such as cardinality, equality, disjointness, etc. OWL is a World Wide Web Consortium (W3C) recommendation.

The design of ontologies is a complex task and generally an engineering process that is never fully completed. There are different methodologies for designing ontologies. One of the most common is the *NeOn Methodology* [SGF12]. It is a methodology for building ontologies and ontology networks and suggests procedures for ontology development.

In the field of robotics, several domain ontologies have been developed. The Core Ontology for Robotics and Automation (CORA) [15i] serves as a foundational ontology in systems such as SkiROS2 (***Paper*** 1). *Rosetta* [MJ13] represents another domain-specific ontology for industrial robotics. In the realm of service robotics, the Socio-physical Model of Activities (SOMA) [BPB18], developed as part of the *KnowRob* framework, adapts concepts from the *OpenCyc* ontology [Len95] for common sense reasoning and is utilized for generating reactive action plans in household tasks [Lei+19]. *Cyc* is particularly comprehensive. It contains about 1.5 million general concepts and more than 25 million general rules and assertions [CA22, p.424]. A comprehensive review of ontology-based knowledge representation in robotic systems is provided in [Oli+19].

3.3 Reasoning

Reasoning is the process of utilizing existing knowledge to infer new information or answer specific questions about the world [SK16, p.338]. It is the process of deriving new knowledge from existing knowledge. This process involves the application of logical techniques such as deduction and induction to derive new knowledge from established facts.

Deductive reasoning involves inferring specific facts from general principles or known truths [ICK16, p.22]. For instance, if it is known that a robot is located in a room and that the room is within a building, one can deductively infer that the robot is in the building. Deductive reasoning is characterized by its precision and certainty in conclusions drawn from given premises.

Conversely, inductive reasoning entails formulating general rules or principles based on specific observations or facts [ICK16, p.22]. Using the previous example, observing that a robot is in a room within a building might lead to the inductive generalization that robots are typically found in buildings. Inductive reasoning is essential for hypothesis formation and theory development, though it carries a degree of uncertainty compared to deduction.

Reasoning plays a crucial role in contextualizing actions within AI systems, particularly in robotics. It enables the automatic deduction of necessary know-

```
1     PREFIX foaf: <http://xmlns.com/foaf/0.1/>
2     SELECT ?name
3           ?email
4     WHERE
5       {
6         ?person a foaf:Person .
7         ?person foaf:name ?name .
8         ?person foaf:mbox ?email .
9       }
```

Listing 2: An example of a SPARQL query that outputs the names and email addresses of all persons.

ledge, allowing for the dynamic parameterization of actions based on their context [CA22, p.413]. Together with reasoning, the abstract formulation of the knowledge makes sure that the knowledge can be applied to situations that were unknown when the knowledge was specified [CA22, p.416].

SPARQL protocol and rdf query language (SPARQL) is a powerful tool to query RDF graphs and to reason about the information they contain [HKR09, p.262]. SPARQL queries, as demonstrated in Listing 2, are supported by systems such as the SkiROS2 world model in ***Paper*** 1. Additionally, in SkiROS2 reasoning capabilities can be manually coded into skills, and custom reasoners, such as the *AAU Spatial Reasoner*, can be implemented for specialized applications.

Prolog, a logic programming language rooted in first-order logic, is frequently employed for reasoning tasks [RND10, p.339]. It operates on a formal system of rules to derive new facts and is characterized by its declarative nature, where the control flow is not explicitly defined by the programmer. Prolog-based systems typically adhere to the closed-world assumption, assuming that any statement not known to be true is false [RND10, p.339].

An exemplary implementation of a knowledge-based architecture with integrated reasoning capabilities is *KnowRob (Knowledge processing for robots)* [TB09]. KnowRob employs Prolog-based reasoning, primarily follows the closed-world assumption [RND10, p.339], but can also accommodate the partially open-world assumption through the use of 'computables' [TB13]. Its second-generation architecture features a logic interface to a hybrid reasoning kernel, illustrating the advanced integration of reasoning in robotic systems [Bee+18].

4 Robot Skills

Robot skills, in the context of robotics and artificial intelligence, are a type of action that are typically defined as adaptable and re-parameterizable building blocks for robot programs [Bøg+12]. These skills can be essential components in the repertoire of a robotic system, enabling it to perform a variety of actions and operations. The concept of robot skills has been described in the literature as "predefined building blocks" [Bøg+12], "reusable robot programs" [Ste17], or "recurring actions" [Rov17]. Central to the utility of robot skills is their ability to translate abstract symbolic reasoning into physical actuation, interfacing high-level decision-making processes with low-level control mechanisms [KB17].

This section introduces the notion and important aspects of robot skills. It will also discuss procedure representations for sequencing and combining skills, the use of behavior trees (BTs) as a mathematical model for skill representation, and the integration of motion generators. Finally, knowledge integration into robot skills is discussed.

4.1 Skill Aspects and Procedure Representation

In robotic systems, the execution of tasks fundamentally relies on the robot's ability to perform a sequence of actions through its actuators [Bøg+12]. When these actions are defined as building blocks with specific semantics, they are referred to as "robot skills" [Bøg+12]. This contrasts with the actions in RL, which are often incremental and executed at a granular level.

Robot skills are characterized by their modularity and reusability, essential properties that enable the construction of complex procedures from simpler components [Ped+16]. This modularity also facilitates the easy exchange and adaptation of skills to different tasks and contexts [Bru+07]. The parameterization of skills is another crucial aspect, allowing for their customization to meet the specific requirements of a task and to align with the current operational context [Bru+07]. The level of autonomy in skill execution varies; some skills function as primitive building blocks, while others encompass more complex procedures with perception, internal reasoning and error handling capabilities.

Pre- and post-conditions can be used as part of the skill model [Bøg+12] and provide semantics. They define necessary prerequisites in order to perform a skill and additionally also check after the execution whether the intended goal is achieved. A concrete version of these checks is suggested in [PHK14] and used

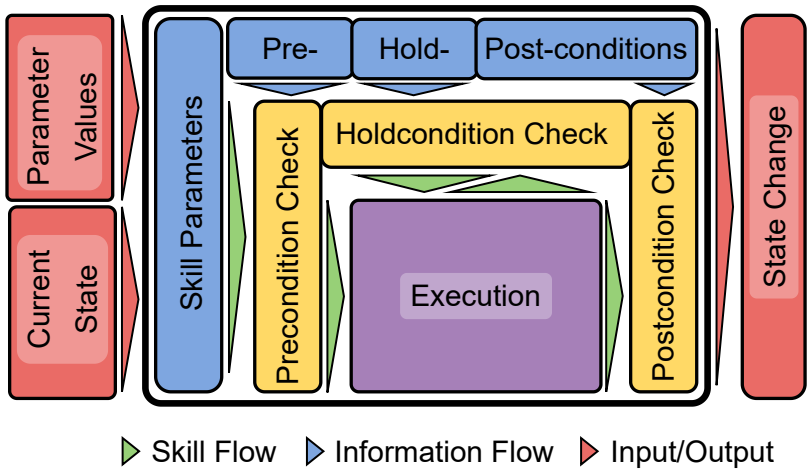


Figure 5: The SkiROS2 skill model with the pre-, hold- and post-conditions that guard the execution. ©2023 IEEE

in that line of research [Rov+17; RGK17; Rov+18]. The skill model is shown in Figure 5. It is noteworthy that this definition of skill based on conditions allows for a broad interpretation of what constitutes a skill. This openness facilitates the integration of pre-existing procedures into the skill ecosystem, enhancing adaptability and ease of adoption.

A comprehensive review of action representations across three domains — vision, robotics, and AIs — is provided in [Krü+07], highlighting the varied interpretations of actions in these fields. A detailed taxonomy and systematic classification of action representations in robotics are further explored in [Zec+19].

Robot skills can be categorized based on their level of granularity, typically divided into lower-level skills, which are closely tied to hardware, and higher-level skills, which are constructed from these foundational skills [Zec+19]. An example of this differentiation is seen in the SkiROS2 skill model, that distinguishes between primitive and compound skills. Primitive skills are semantically atomic and directly implementable, while compound skills enable the combination of multiple primitive or compound skills into more intricate procedures using BTs.

Procedure Representation

In addition to the aspects of the skill model, the choice of procedure representation has a huge impact on the modularity and efficiency. Among the

popular models for procedure representation are finite state machines (FSMs), hierarchical finite state machines (HFSMs) and Petri nets. Traditionally, many robot programming solutions have relied on FSMs [HMU01]. However, FSMs are often found to be less suitable for complex tasks due to their lack of modularity and scalability issues, as discussed in recent studies [BZS21; Iov+23b].

HFSMs [GLL99] extend the basic FSM framework by incorporating hierarchical structures, thereby enhancing modularity and scalability. This structural improvement makes HFSMs more adaptable than traditional FSMs, though they still encounter limitations when dealing with highly complex tasks. Petri nets [Ros04], on the other hand, offer a more expressive model for procedure representation compared to HFSMs. The increased expressiveness of Petri nets allows for a more nuanced representation of procedures, but this comes with a higher level of complexity.

In recent developments, an increasing number of robot programming solutions are adopting behavior trees (BTs) [CÖ17a]. BTs provide a flexible and scalable approach, making them particularly effective for a wide range of tasks. Their structure supports dynamic decision-making and adaptability, essential qualities for robotic systems operating in unpredictable or variable environments.

4.2 Behavior Trees

I don't think there is a reason to move away from the behavior-tree concept.
— Torsten Kroeger, CTO of Intrinsic.ai[2]

Behavior trees (BTs) are a mathematical model for procedure representation that is based on graph theory. They surpass finite state machines (FSMs) in expressiveness [BZS21] and offer greater modularity [BZS22], making them particularly suitable for complex robotic tasks. Initially described in the context of computer games [CÖ14], BTs have since found successful applications in robotics, as evidenced by recent surveys and analyses [Iov+22; Ghz+20].

A BT is defined as a directed acyclic graph $G(V, E)$ that comprises $|V|$ nodes and $|E|$ edges. It consists of *control flow nodes* or *processors*, and *execution nodes*. The classical formulation defines four types of *control flow nodes*: 1) *sequence*, 2) *selector*, 3) *parallel* and 4) *decorator* [Mar+14]. Each BT has an initial node

[2] On the question on how the program flow can be dynamically adapted to changes in the environment. `https://www.youtube.com/watch?v=kr3AqXjK-pk&t=2594s`

without parents, defined as *Root*, and one or more nodes without children, called *leaves*. An example of a BT is shown in Figure 6.

The execution of a BT involves periodically injecting a *tick* signal into the *Root*, which then propagates through the tree according to the *control flow nodes* and the return states of their child nodes. By convention, this signal propagates from left to right, allowing for the prioritization of certain paths [CÖ17a].

The *sequence* node functions as a logical *AND*, succeeding only if all its children succeed, and failing if any child fails. The *selector* node, also called *fallback*, represents a logical *OR*. It fails only if all its children fail, ceasing to tick further children upon the success of any child. The *parallel* node forwards ticks to all its children, failing if any child fails. Decorator nodes are used to implement custom functions such as altering return signals.

The leaves of the BT are the *execution nodes* that, when ticked, perform an operation and output one of the three signals: *success*, *failure* or *running*. In particular, execution nodes subdivide into 1) *action* and 2) *condition* nodes. An action performs its operation iteratively at every tick, returning running while it is not done, and success or fail otherwise. A condition never returns running: it performs an instantaneous operation and returns always *success* or *failure*.

In [RGK17], the integration of condition nodes into action nodes within the extended behavior tree (eBT) format is proposed. This integration not only streamlines the tree, making it more compact and interpretable, but also facilitates interfacing with hierarchical task network (HTN), as discussed in Section 5. An example of an eBT is illustrated in Figure 6.

A key advantage of BTs is their modularity and high level of *separation of concerns* [Der08]. Unlike finite state machines (FSMs) [HMU01] which follow a *one-way control transfer*, BTs enable a *two-way control transfer* due to their return signal mechanism. This is akin to the difference between *GOTO* statements versus function calls in programming, where function calls enhance readability and reusability [CÖ14; BZS21; BZS22]. Furthermore, the periodic *tick* signal allows reactive behaviors as shown in [Rov+18; Mar+14; May+21].

BT implementations typically utilize a blackboard for information exchange, often implemented as a volatile key-value store, ensuring that the information persists until the BT execution is halted.

For a comprehensive understanding of BT in robotics, readers are referred to the BT book [CÖ17a] as well as a recent survey on BTs in robotics [Iov+22]. A further analysis on the usage of BTs in robotics is given in [Ghz+20].

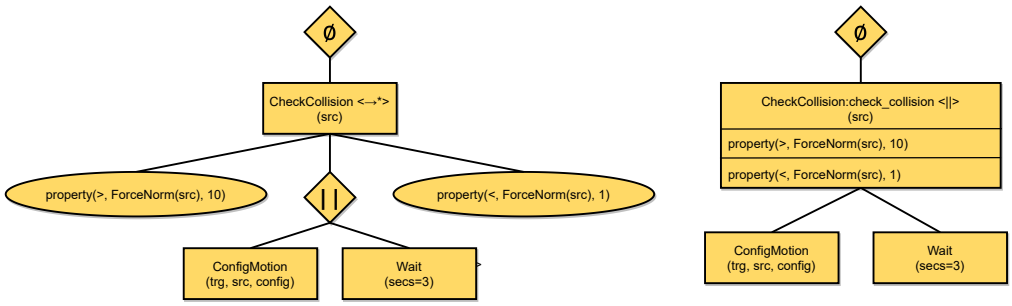


Figure 6: A BT to check for a collision with a precondition of having a force above $10\,\mathrm{N}$ and a postcondition of having a force below $1\,\mathrm{N}$. As a mitigation a new motion is commanded while waiting for $3\,\mathrm{s}$. The left side shows a standard BT and the right side the more compact eBT formalism.

4.3 Combining Behavior Trees with Motion Generators

In [Rov+18], a new way to model skills is suggested. Previous work typically created skills as self-contained primitive blocks. In contrast to that, the authors suggest a composable structure that allows for concurrent motion primitives with interferences. To achieve this, it exploits the additive property of MGs. In the suggested structure, a BT is used to dynamically activate motion primitives when conditions trigger.

A key idea is to have small and composable building blocks to create force-sensitive robot motions. The robot arm is controlled in the Cartesian end-effector space. A *motion generator (MG)* is defined as a Cartesian impedance controller that can 1) superimpose a varying Cartesian wrench over the motion and 2) follow safety requirements by limiting velocities, accelerations and torques. These requirements are addressed by the Cartesian impedance controller implementation in [MS22].

This concept is particularly effective in implementing recovery behaviors in scenarios with uncertainties. For example, in a piston insertion use case similar to Figure 1, superimposed motion primitives are activated for error mitigation upon detection of deviations in the insertion process. Furthermore, this formulation proves beneficial in contact-rich tasks, offering a significant advantage over DMPs by eliminating the need to teach specific motions. Moreover, the explicit setting of motion parameters, as opposed to implicit formulations like neural network (NN) or DMPs, facilitates a more seamless interface with reasoning systems and knowledge integration.

Another notable benefit of this approach is its high level of abstraction from individual joint control, which accelerates the learning process for RL policies [VGK19]. This abstraction also simplifies the transfer of learned behaviors from simulation environments to real-world applications, as discussed in the survey on simulation-to-reality transfer in robotics [Cha+19] and shown in ***Paper*** 2.

4.4 Knowledge Integration

The integration of knowledge into robot skills is a critical step in the development of scalable and adaptable robotic systems [CA22, p.428]. With established skill models and procedure representations in place, the focus of this section shifts to how knowledge can be effectively incorporated into these skills.

Knowledge representation plays an essential role in decoupling knowledge from the procedural code [CA22, p.424]. This separation is crucial for reusing skills across different tasks and contextualizing them according to varying requirements [KB17]. It also facilitates the easy exchange of knowledge without necessitating code modifications, a key factor for scalable skill-based systems. Given the dynamic nature of knowledge, a robust skill model should support the reusability of skills even as the underlying knowledge evolves.

Despite its importance, many existing skill platforms and procedure representation implementations lack provisions for knowledge integration. In the realm of FSMs, popular implementations such as *SMACH* [Ros], *SMACC2* [Git], and *FlexBE* [SKv16] do not inherently support knowledge integration. However, research conducted at Lund University, particularly within the *ROSETTA* and *SMERobotics* projects, has demonstrated viable approaches to achieve this integration [SM13; MJ13; Ste+15; SM15; Top+18].

In service robotics, the *KnowRob* framework exemplifies the successful integration of knowledge into robot skills [TB13; Bee+18]. The cognitive robot abstract machine (CRAM) [BMT10; BKV23] within this framework generates action plans based on the represented knowledge, showcasing the practical application of knowledge integration in robotic systems.

While popular BT implementations like *BehaviorTree.CPP*[3] and *pytrees*[4], as well as comprehensive robotic frameworks like *CoSTAR* [Pax+17], do not offer direct knowledge integration, SkiROS2 stands out as a platform that effectively incorporates structured knowledge integration within BTs. This capability dis-

[3] `https://github.com/BehaviorTree/BehaviorTree.CPP`
[4] `https://github.com/splintered-reality/py_trees`

tinguishes SkiROS2 as a unique and advanced solution in the field of robotic skill modeling.

5 Task Planning

Task planning, also known as action planning, is a critical capability in robotics, enabling robots to strategize their actions to achieve specific goals [GNT04]. This process involves the formulation of high-level plans that guide the robot's actions in pursuit of its objectives.

In 1971, the Stanford Research Institute Problem Solver (STRIPS) was introduced [FN71] and with it, robot control has been among the first applications for planning [SK08, p.217]. STRIPS marked a significant milestone in the application of planning algorithms in robotics, as evidenced by its implementation in the control of the robot *SHAKEY* [Nil84]. One of the seminal challenges identified early in the development of planning systems was the need for a robot control architecture that could effectively bridge the gap between abstract high-level plans and concrete low-level robot control [Sic09]. Additionally, this architecture needed to account for the inherent uncertainties present in real-world environments [Sic09].

To address a planning problem, it is essential to define a planning domain, which encompasses the potential actions, their prerequisites, and their effects. Ideally, solving a planning problem would require a comprehensive simulation of the world. However, this approach is often impractical due to limitations in available information and the prohibitive computational costs involved. Consequently, a trade-off similar to that in knowledge representation and reasoning (as discussed in Section 3) emerges between expressivity and efficiency. To manage this trade-off, planning domains are frequently simplified to symbolic states, reducing complexity while retaining essential information for decision making [SK08, p.215].

5.1 PDDL and Hierarchical Task Networks

In the field of robotic planning, the planning domain definition language (PDDL) has emerged as the de facto standard to describe planning domains [SK08, p.216]. PDDL [Aer+98] has seen various extensions, such as *Multi-Agent PDDL* [Kov12], to enhance its capabilities. PDDL is also used in *ROSPlan* [Cas+15] and *PlanSyS2* [Mar+21]. In SkiROS2, PDDL is utilized in combination with the eBT format. SkiROS2 can automatically generate planning domain descriptions from the pre- and post-conditions of skills [Cro+17; RGK17]. The knowledge stored in the WM and the defined planning goal are used to create a problem description, thereby obviating the need for users to specify anything in PDDL

other than the goal.

Hierarchical task networks (HTN) are a method for automated planning and decision making [GNT04, Chap.11]. A key feature is that they focus on decomposing complex tasks into simpler subtasks. HTN are more focused on how to perform a task and to decompose the actions rather than searching for a goal state. In [RGK17], a combination of BTs and HTN is suggested that combines the advantages of deliberate planning with the dynamic expansion and the reactivity of BTs. This method is implemented in ***Paper*** 1 and utilized in ***Papers*** 2, 4, 5, 6.

5.2 Backchaining of Behavior Trees

A different type of planning method is suggested in [CAÖ19].

This method leverages the capabilities of BTs in conjunction with backchaining to develop a reactive sequence of actions that lead to a specified goal state. The algorithm starts from the goal state and iteratively selects actions to achieve this goal. When it encounters preconditions that are not yet satisfied, the BT is expanded to include actions that can fulfill these preconditions.

A key advantage of this approach is the reactivity of the resulting BT without explicit replanning. The robot can immediately adapt to changes in the environment, re-performing actions if necessary and skipping actions that are no longer required. This reactivity makes BTs particularly suitable for collaborative tasks where the environment may change dynamically. Additionally, unlike static planning methods, the BT in this approach is not fixed and can be expanded during runtime to respond to new or changing conditions in the environment.

This backchaining algorithm has been applied in various research works, including [Sty+22; Gus+22; Iov+23a], and is also utilized in ***Paper*** 7. For a comprehensive review of planning with BTs, including the backchaining method, readers are referred to Section 4.2 of [Iov+22], which provides an extensive overview of this topic.

6 Robot Learning

Learning is a general term that can be described as "as the ability to improve the system's own performance or knowledge based on its experience" [SK08, p.219]. This experience is typically garnered through data acquired from interactions with the environment or a human operator.

Historically, learning in robotics was grounded in symbolic representations, where new knowledge was generated through methods such as deduction, induction, or analogy [SK08, p.219]. However, the landscape of learning has shifted significantly with the advent of ML, particularly statistical learning methods used for tasks such as data classification. The breakthroughs in deep learning, exemplified by *DanNet* [Cir+11] and *AlexNet* [KSH12] post-2012, coupled with increased computational power, have made statistical ML methods popular.

This section delves into different facets of robot learning, beginning with LfD as a method for imparting skills to robots. LfD is instrumental in teaching robots by example, enabling them to replicate and adapt these behaviors for similar tasks. Following this, the focus shifts to the learning with BTs. The section then introduces RL as a technique for policy learning in robotics. RL involves a trial-and-error approach in which robots learn optimal behaviors based on feedback, but it also presents challenges, such as dealing with complex real-world environments and the need for efficient learning algorithms. Lastly, Bayesian optimization (BO) is explored as a method for policy search, offering a systematic way to optimize policy parameters, especially in scenarios where policy evaluation is resource-intensive.

6.1 Learning from Demonstration

Learning from demonstration (LfD) is a way to teach robots by showing them how to perform a task. The goal is to learn a mapping between world states and actions [Arg+09]. LfD methods are especially useful when the operator does not have sufficient knowledge to program the robot. The term LfD is sometimes used interchangeably with "imitation learning" [SK08, p.1995] or "programming by demonstration" [SK08, p.1371][Arg+09]. However, some authors see programming by demonstration as a rather limited direct creation of programs without generalization.

In LfD there are three main methods [KP14]:

1. **Kinesthetic Teaching**: Here, the robot is physically guided by an operator to perform tasks, allowing for direct and intuitive teaching.

2. **Teleoperation**: In this method, the robot is remotely controlled, for example, via a joystick, enabling the operator to demonstrate tasks from a distance.

3. **Observational Learning**: The robot observes a human operator performing tasks. A key challenge in this method is the robot's ability to interpret human actions and solve the correspondence problem [Arg+09].

The research in LfD intersects with the one in the motion representations in Subsection 2.2. Many of the motion representations like DMPs build upon demonstrations, generalize them and execute them on robot systems.

Examples for observational learning are a line of research around Ramirez-Amaro et al. [Alb+11; RBC14; RBC15; RBC17] where the behavior of a human is observed, segmented and then imitated on a robot.

A programming by demonstration approach with kinesthetic teaching is introduced by Stenmark et al. in [SHT17]. The explicit challenges of synchronized dual-arm motions are covered in [Ste+17]. With this approach, the learned skills can also be saved in a knowledge base and reused in other tasks.

Since many LfD methods like [SHT17] are online methods that require direct access to the robot, they may be less suitable for industrial robots. The worker does not only need to be physically present at the robot, but the system is also taken out of production during the teaching. Advanced LfD approaches necessitate additional infrastructure for analyzing, storing, and generalizing from demonstrations. Demonstrating contact-rich tasks is particularly challenging and requires careful consideration.

A comprehensive survey on LfD methods and their categorization is provided in [Arg+09]. Furthermore, the "Handbook of Robotics" has dedicated section on learning from humans [SK16, Sec.74] and programming by demonstration [SK08, Sec.59].

6.2 Learning with Behavior Trees

The primary methodologies for learning BTs encompass RL, evolutionary methods and LfD. These methods generally focus on either refining the structure of the BTs or optimizing their parameterization.

The structural learning of BTs has been explored in several studies, including [LBC10; Per+11; CPO18]. A novel line of research employs genetic programming to evolve the structure of BTs, as demonstrated in [Iov+21]. In [Sty+22], the learning process is integrated with inputs from the backchaining planner discussed in Subsection 5.2. Furthermore, [Iov+23a] combines structural learning with LfD.

The LfD approach used in [Iov+23a] itself is introduced in [Gus+22]. It eases demonstration efforts by automatically deducing reference frames and preconditions. Another approach is shown in [Fre+19; Wat+22] where they learn a decision tree first and then convert it into a BT. In [Gug+23] a method is presented that learns BTs from the execution traces of the robots.

A more comprehensive review of learning applications with BT is provided in Section 4.1 of [Iov+22].

6.3 Reinforcement Learning (RL)

ML methods can be divided into three major categories [RND10, p.694]:

1. Supervised learning
2. Unsupervised learning
3. Reinforcement learning (RL)

Supervised learning methods utilize data that is labeled with the correct outcome to learn a classifier [RND10, p.695]. A classical example is to distinguish cats and dogs or to recognize written numbers in an image. Supervised learning is used in ***Paper*** 6 to learn the Performance and Feasibilty Model (PerF). The second category of methods is unsupervised learning. These methods do not rely on labeled data, but instead just use the data itself [RND10, p.695]. A classical example is an automatic categorization of data points into different clusters. A special subcategory within unsupervised learning are self-supervised learning methods [RND10, p.986]. Self-supervised methods use the data itself to generate the labels. A classical example is to predict the next frame in a video sequence. Here, the next frame can be used as a labeled data point for the prediction based on the current frame.

The third category of methods are reinforcement learning (RL) methods. The important distinction from the other two methods is that there is not a defined

dataset [RND10, p.695]. Learning is performed via trial and error [SB18, pp.369]. In RL, data is generated through interaction with the environment, meaning the robot's decisions directly influence the data it encounters. The primary goal in RL is to develop a policy, a strategy that maximizes a reward. This policy maps states to actions, determining the action to be taken given the current state of the system. The reward signal, typically a scalar value, is provided by the environment and used to evaluate the actions of the policy. Unlike supervised learning, the policy in RL must learn the optimal actions through a process of trial-and-error [SB18, pp.369].

A unique challenge in RL, not present in supervised or unsupervised learning, is the balance between *exploration* and *exploitation* [SB18, p.3]. The agent must explore the environment to discover optimal actions while also exploiting its existing knowledge to make effective decisions. This balance is crucial for the success of an RL-based system as it navigates complex and often unpredictable environments [SB18, p.26].

Markov Decision Processes

RL problems are commonly formulated as Markov decision process (MDP) [SB18, p.47]. A MDP is a tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R} \rangle$ that consists of a state space $\mathcal{S}$, an action space $\mathcal{A}$, a transition function $\mathcal{P}(\mathbf{s}, \mathbf{a})$ and a reward function $\mathcal{R}(\mathbf{s}, \mathbf{a}, \mathbf{s}')$. A fundamental characteristic of MDPs is their Markovian property, signifying that the probability of transitioning to a future state depends solely on the current state and action, and not on the sequence of past states. This property simplifies the decision-making process by focusing only on the present state. The transition function $\mathcal{P}$ defines the probability of transitioning from one state to another state given an action. The reward function $\mathcal{R}(\mathbf{s}, \mathbf{a}, \mathbf{s}')$ defines the reward that is obtained when transitioning from one state to another state given an action. The goal is to find a policy π that maximizes the expected return J [SB18, p.53].

The implementation of ***Papers*** 3, 4, 5, 6 strongly follow this definition. The state space $\mathcal{S}$ is the joint configuration of the robot and the pose of objects. The action space $\mathcal{A}$ is the reference pose of the controller, the stiffnesses and the wrench applied with the EE. The reward functions $\mathcal{R}$ are the reward function that is defined in the respective papers.

Classes of Reinforcement Learning Algorithms

One criterion for evaluating RL methods can be on a spectrum from *value-based methods* to *policy-search methods*. Value-based methods learn a value function that is used to select the best action. Policy-search methods directly search for the optimal policy [RND10, p.848].

Value-based Methods: Value-based methods focus on learning a value function, which is then utilized to select the most advantageous action [RND10, p.27]. This category includes two primary types of value functions: the state-value function $V(s)$ and the state-action-value function $Q(s, a)$. The state-value function $V(s)$ represents the expected return when starting in state s and adhering to a policy π. Conversely, the action-value function $Q(s, a)$ denotes the expected return when in state s, taking action a, and subsequently following policy π. The policy π is derived from the value function by choosing the action associated with the highest value [SB18, p.27]. Learning the value function can be achieved through a model of the environment or direct interaction with the environment.

However, even with function approximation ([SB18, p.198]) value-based methods face challenges when scaling to high-dimensional state spaces, a phenomenon known as the curse of dimensionality. This limitation becomes particularly evident in complex tasks, such as those that involve contact-rich interactions. For example, in the research presented in ***Papers*** 3, 4, 5, 6, the state space has a dimensionality of 14, with an additional 7 dimensions for each object involved. The action space itself has a dimensionality of 19, as detailed in the respective papers.

Policy Search Instead of building a model of all the states, policy search methods directly search for the optimal policy [RND10, p.848]. This class of methods is "the real renaissance" [KCC13] of RL methods in robotics.

In order to optimize for policy parameters, they follow the policy search formulation [DNP13; Cha+19; Cha+17]. A dynamical system is modelled in the form:

$$\mathbf{s}_{t+1} = \mathbf{s}_t + M(\mathbf{s}_t, \mathbf{a}_t) \tag{1}$$

with continuous valued states $\mathbf{s} \in \mathbb{R}^E$ and continuous valued actions $\mathbf{a} \in \mathbb{R}^U$. The transition dynamics can generally be modeled by the simulation of the robot or the real robot system: $M(\mathbf{s}_t, \mathbf{a}_t)$.

In a single-objective case, the goal is to find a policy $\pi, \mathbf{a} = \pi(\mathbf{s}|\theta)$ with policy parameters θ such that $\mathbf{a}$ maximizes the expected long-term reward when executing the policy for T time steps:

$$J = \mathbb{E}\left[\sum_{t=1}^{T}\sum_{u=1}^{U} r_u(\mathbf{s}_t, \mathbf{a}_t, \mathbf{s}_{t+1})|\theta\right], \tag{2}$$

with $r_s(\mathbf{s}_t, \mathbf{a}_t, \mathbf{s}_{t+1})$ being the immediate reward of reward function u for being in state $\mathbf{s}$ and performing action $\mathbf{a}$ at time step t.

Policy search methods in RL can be categorized into model-based and model-free approaches. Model-based methods involve learning a model of the environment, which can then be used to simulate environmental dynamics and facilitate action planning [PN17]. An illustrative example is the pendulum swing-up task, where the dynamics of the pendulum can either be explicitly modeled using differential equations or can be learned efficiently through interaction with the environment.

Model-free RL methods, on the other hand, do not rely on an environmental model. Instead, they focus directly on learning an optimal policy. These methods are often lauded for their efficiency in finding effective policies without the need for an explicit environmental model. A comprehensive survey on policy search methods is provided in [DNP13], which discusses various classes of policy search methods along with their respective advantages and disadvantages. A more recent survey [Cha+19] focuses on efficient policy search methods, highlighting recent advancements in the field.

Despite their efficiency, policy-search methods are sometimes criticized for being sample inefficient, particularly because they do not learn a model of the environment. However, in many robotics applications, developing an accurate model of the environment can be prohibitively complex and costly. For example, in contact-rich tasks as explored in ***Papers*** 3, 4, 5, 6, a comprehensive model would need to encompass the dynamics of the robot, objects, friction, contact forces, contact points, stiffnesses, dampings, and friction at the contact surfaces. Chatzilygeroudis et al. [CM18] address this challenge by using a simulator as a base model and augmenting it with a residual model that corrects the simulator's predictions, thereby enhancing the accuracy and applicability of the model in complex robotic tasks.

The dynamics are formulated as follows:

$$\mathbf{s}_{t+1} = \mathbf{s}_t + M(\mathbf{s}_t, \mathbf{a}_t) + f(\mathbf{s}_t, \mathbf{a}_t), \tag{3}$$

With $f(\mathbf{s}_t, \mathbf{a}_t)$ being the residual model between simulation and reality. However, with contact-rich tasks, we saw that such updates of the state were too difficult

to learn to improve the policy search in simulation. Instead, we used a model-free policy search method that directly searches for the optimal policy.

6.4 RL in Robotics

Robotics is both a challenging and promising field for RL. The field is challenging due to the high-dimensional continuous state and action spaces, prolonged training durations and significant safety concerns [KP14]. However, it is also promising, as robots have the capability to directly manipulate the physical world and address real-world problems.

RL is particularly useful for tasks that are difficult to teach in a way that robots can replicate, such as certain contact-rich tasks [Elg+23]. In some scenarios, even the instructor may not know the optimal solution. Additionally, RL offers adaptability to new task variations or environmental changes.

However, robotic RL faces distinct challenges due to the nature of hardware interaction. Actions, such as arm trajectories, need to be smooth and continuous. The learned policy must ensure safety for the environment, the robot and humans, even when learning [SB18, p.477]. Moreover, learning should be sample efficient, as data collection in robotics is often resource-intensive [Elg+23]. Many tasks do not inherently reset the environment, necessitating manual intervention by operators. The true state of the system may be unobservable or subject to noise, and the policy must operate in real-time [KP14].

Incorporating prior information or biases into the learning process can be beneficial to address these challenges. However, this requires that the learning method and the policy to be compatible with such integrations.

An efficient model-based RL approach is introduced in [Cha+17], which typically employs Gaussian process (GP) regression to learn a model. This approach is further extended in [CM18] to learn a residual model that corrects the predictions of a simulator.

Deep RL has been successfully applied to learn reference points and stiffnesses in compliant controllers [Yan+22], as well as for complex in-hand manipulation tasks, such as cube manipulation [And+20]. In [NLZ22], which forms the baseline in ***Paper*** 7, deep neural network (DNN) are used to select and parameterize behavior primitives.

While these approaches are targeted, there are instances where deep RL has been effectively applied in an end-to-end manner [Lev+16]. Recent work also explores

the integration of language into the learning process [Bro+23], expanding the scope and capabilities of RL in robotics.

Sim2Real Transfer

All models are wrong, but some are useful.
— George Box

In the field of RL, the concept of Sim2Real transfer addresses the challenges of training RL algorithms in real-world robotic systems. Due to efficiency constraints and safety considerations, particularly in industrial settings with fragile and costly environments, it is often impractical or risky to train RL methods directly on actual robot systems [Elg+23]. Consequently, the practice of training in simulation and subsequently transferring the learned policies to real systems, known as Sim2Real transfer, has become a vital approach.

The transfer from simulation to reality is not always straightforward. Discrepancies between the simulated and real environments can hinder the effective application of RL methods trained in simulation [SB18, p.476]. This issue is particularly pronounced when the simulation lacks sufficient accuracy, leading to policies that may not perform as expected in the real world.

One strategy to enhance the transferability of policies is through domain randomization [JDJ17; Pen+18; Pau+20; Meh+20]. This technique involves randomizing various aspects of the simulation to encompass a broader range of potential states. By exposing the policy to diverse simulated conditions, it becomes more robust and adaptable, increasing the likelihood of a successful transfer to real-world scenarios. This approach has been utilized in ***Papers*** 3, 4, 5, 7.

The level of control inputs significantly impacts the transferability of policies. Policies based on lower-level control states, such as joint torques, tend to transfer less effectively than those using higher-level controls like Cartesian control [Cha+19]. Lower-level controls are more susceptible to minor environmental variations, whereas higher-level controls, such as Cartesian impedance control [MS22], often include self-stabilizing elements that facilitate transfer [KP14].

In ***Papers*** 3, 4, 6, policies learned in simulation are transferred and evaluated on real robot systems. ***Paper*** 5 leverages the most successful set of parameter from simulation to expedite learning on real systems. Additionally, ***Paper*** 7

assesses policies learned in the simulation within other unseen simulated environments, providing a validation mechanism for their effectiveness.

Advantages and Limitations of RL in Robotics

RL methods offer significant potential in robotics, enabling systems to adapt and learn autonomously. This adaptability is a substantial advantage over traditional model-based designs, which require pre-specification of all parameters and behaviors. RL can effectively bridge gaps that are not addressed by conventional models.

RL has demonstrated remarkable success, achieving superhuman performance in various domains. Notable examples include mastering complex board games like Go [Sil+16], excelling in high-dimensional single-agent (e.g. the computer game Starcraft [Vin+19]) and multi-agent computer games (e.g. Dota [Ope+19]). RL shifts the focus from detailing problem solutions to the design of learning problems, allowing the system to discover solutions autonomously [SB18, p.469].

Despite these achievements, RL faces significant challenges, particularly in the industrial robotics sector [Elg+23]. Many RL algorithms require extensive data, and surrogate models such as simulators, necessary for accurate and safe training, can be difficult and costly to develop. Robot downtime for learning in an assembly line, for instance, incurs high costs [Elg+23].

The complexity of RL methods, their dependence on hyperparameters, and the difficulty in applying them in real-world scenarios pose additional challenges. A critical concern in industrial robotics is safety, which encompasses human safety, environmental protection, and robot integrity. During the initial exploration phases, RL algorithms may exhibit risky behaviors, which are unacceptable in industrial settings. Surveys on safe RL in robotics, such as those by Garcia et al. [GF15] and Brunke et al. [Bru+22], discuss these challenges and present various approaches to ensure safety.

Furthermore, in case of misbehaviors of learned policies, problem analysis must be possible and realistic mitigation strategies must exist. Both these requirements are difficult to achieve with DNN.

Incorporating existing processes, robot capabilities, and experiences as priors into RL solutions is essential in industrial contexts. Additionally, the solutions must adhere to specific requirements, such as maximum force constraints on parts. The policy representation and the learning strategy must be capable of integrating these elements.

Recent research has explored the injection of priors into deep RL. In [Yan+22] the authors demonstrated learning NN policies with demonstrations from a FSM. The concept of learning new tasks from previously learned policies is explored in [YSS22], and the transfer of policies between robot systems using CycleGANs is presented in [YSS23].

The general limits and potentials of deep learning in robotics are comprehensively discussed in [Sün+18],

6.5 Bayesian Optimization (BO)

A way to perform policy search is to use BO [Sha+16]. BO is a method to optimize black-box functions that are expensive to evaluate. The goal is to find the global optimum of the black-box function with as few evaluations as possible. BO is a sequential optimization method and the steps are outlined in Figure 7. A key feature of BO is that it uses a surrogate model to approximate the black-box function [Sha+16]. Common representations for the surrogate model are GPs [RW06] or random forests that are trained on the data points that are evaluated. Both models are probabilistic models that can be used to estimate the uncertainty of the model [Sha+16]. This uncertainty is used by the acquisition function to select the next best point to evaluate. The acquisition function is a heuristic that balances exploration and exploitation. A common acquisition function is expected improvement (EI) that estimates the expected improvement over the currently best point [Fra18].

BO is not "model-based" in the definition given in Section 6.3 since it does not use or learn a model of the dynamics of the environment. Instead, it learns a surrogate model of the black-box function, hence the total rewards for an episode. However, BO can be combined with model-based methods to learn a model of the environment and then use BO to optimize the policy [Cha+17; CM18].

One advantage of BO is that it can optimize mixed input design spaces where variables can be continuous, discrete, ordinal, or categorical [NKO19]. This makes it well-suited for robotics problems where the problem space can often be mixed.

A plain implementation of BO is not well-suited for high-dimensional problems. However, there are extensions that can handle high-dimensional problems such as *TuRBO* [Eri+19] or *BAxUS* [PNP22]. Based on the assumption that there is a smaller effective subspace, methods such as [PNP22; PNP23; Hel+23] can

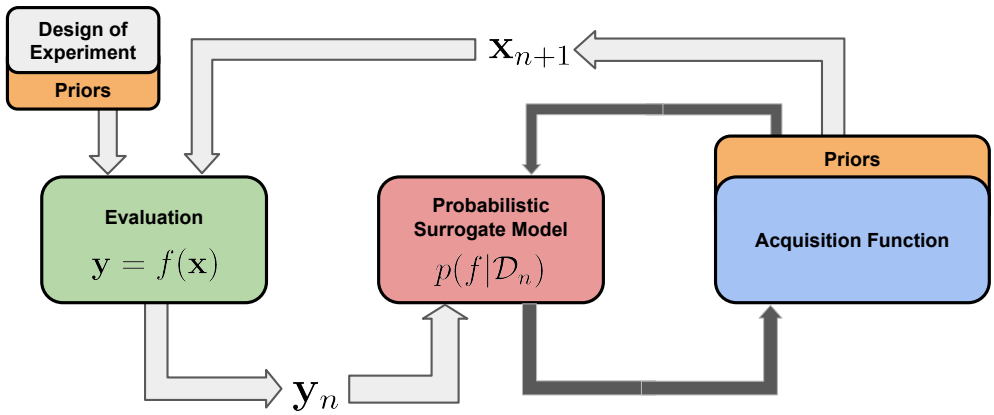


Figure 7: The learning loop of Bayesian optimization (BO). The parameter set $\mathbf{x}$ is evaluated by running an episode. An internal probabilistic surrogate model is built and used by the acquisition function to select the next parameter set to evaluate. Additionally, it is marked where the prior information with πBO [Car+22] is injected.

effectively tackle problems with hundreds of dimensions.

The performance of BO can be improved if prior knowledge is available. While some methods can utilize priors for the surrogate model such as the choice of the GP kernel,injecting a prior for the location of the optimum is an emerging topic. Such a prior can be easier to formulate for non-experts since it is just necessary to formulate the expected location of the optimum in parameter space. In [Sou+21] this prior in the parameter space is used to calculate a pseudo-posterior that is used together with the EI acquisition function. More recently in [Car+22], the prior injection method πBO is suggested that weights the acquisition function. Its application in the learning problem is depicted in Figure 7. πBO is compatible with a wide range of acquisition functions and can be used with different surrogate models. This method is utilized in ***Papers*** 5, 7.

7 Approach

There is substantial evidence that accomplishing manipulation tasks requires robotic agents, as well as the human brain, to employ a combination of learning, planning, and other reasoning methods.
— Michael Beetz[5]

Building upon the problem scope and research objectives outlined in Section 1, and the insights into the current state of the art provided in Section 2, this section delineates the approach adopted to achieve the research goals. This approach integrates the methods discussed in Section 3 through Section 6, aiming to develop a comprehensive framework for learning with robot skills.

A key decision in this approach is not to focus on learning the structure of the policy. Recent studies, such as [Iov+21], have indicated that learning the structure of a BT with genetic programming is a relatively slow process, often requiring hundreds of thousands of episodes. Even with the integration of imperfect solutions from planners as priors [Sty+22], the learning rate remains impractical for real robot systems, particularly in industrial contexts. If such an approach is combined with the complex task of searching for optimal skill parameters, this appears to be beyond the scope for practical industrial applications.

This section begins with an introduction to the specific skill model employed, followed by a short overview of the motion representation and execution methods. Subsequently, the learning pipeline for new tasks is described in detail. In Section 7.4, the learning process and outcomes are elaborated upon, providing insights into the effectiveness and applicability of the proposed methods.

A crucial aspect of this approach is the incorporation of prior knowledge and the management of task variations. This involves integrating existing expertise and process understanding into the learning framework, ensuring that the developed solutions are not only effective but also aligned with the practical constraints and requirements of industrial robotics.

[5] [CA22, p.416]

7.1 Skills and SkiROS2

The utilization of skills as reusable building blocks has emerged as a promising approach to manage the complexity of modern industrial robots. SkiROS2, as detailed in ***Paper*** 1, exemplifies such a skill-based robot control platform. It features a layered hybrid control system, integrating explicit knowledge representation and task-level planning.

The architecture is shown in Figure 8. ***Paper*** 1 describes the components in detail, but a brief description of the components is provided here:

> **Skill manager (SM)**: Central to the operation of SkiROS2, the skill manager oversees the loading and execution of skills. It encompasses the BT execution engine, BT expansion mechanism, and automatic parameterization of skills.
>
> **World model (WM)**: Serving as the central knowledge base, the WM maintains the current state of the world, reading ontologies and the vocabulary that is stored in scenes. It supports queries and custom reasoning, facilitating its use by the SM, individual skills and the task manager. An excerpt of a robot description is shown in Listing 1.
>
> **Task Manager**: This component is responsible for generating task-level plans. It processes planning goals in PDDL, formulating planning domains and problem descriptions based on the WM's knowledge. The task manager then invokes a planner to create a plan, typically executed by the SM.

The skill model in SkiROS2, depicted in Figure 9, consists of two main elements:

1. **Skill description**: It defines the skill semantically, specifying parameters and pre-, hold-, and post-conditions.
2. **Skill implementation**: Implementations of skill descriptions.

The skill implementations themselves exist in two granularities:

- **Primitive skills**: Primitive skills are semantically atomic. The implementation is in Python and the primitive skill skeleton provides functions such as *onInit*, *onStart* or *execute*. Primitive skills also have access to the WM and can query and modify the knowledge.

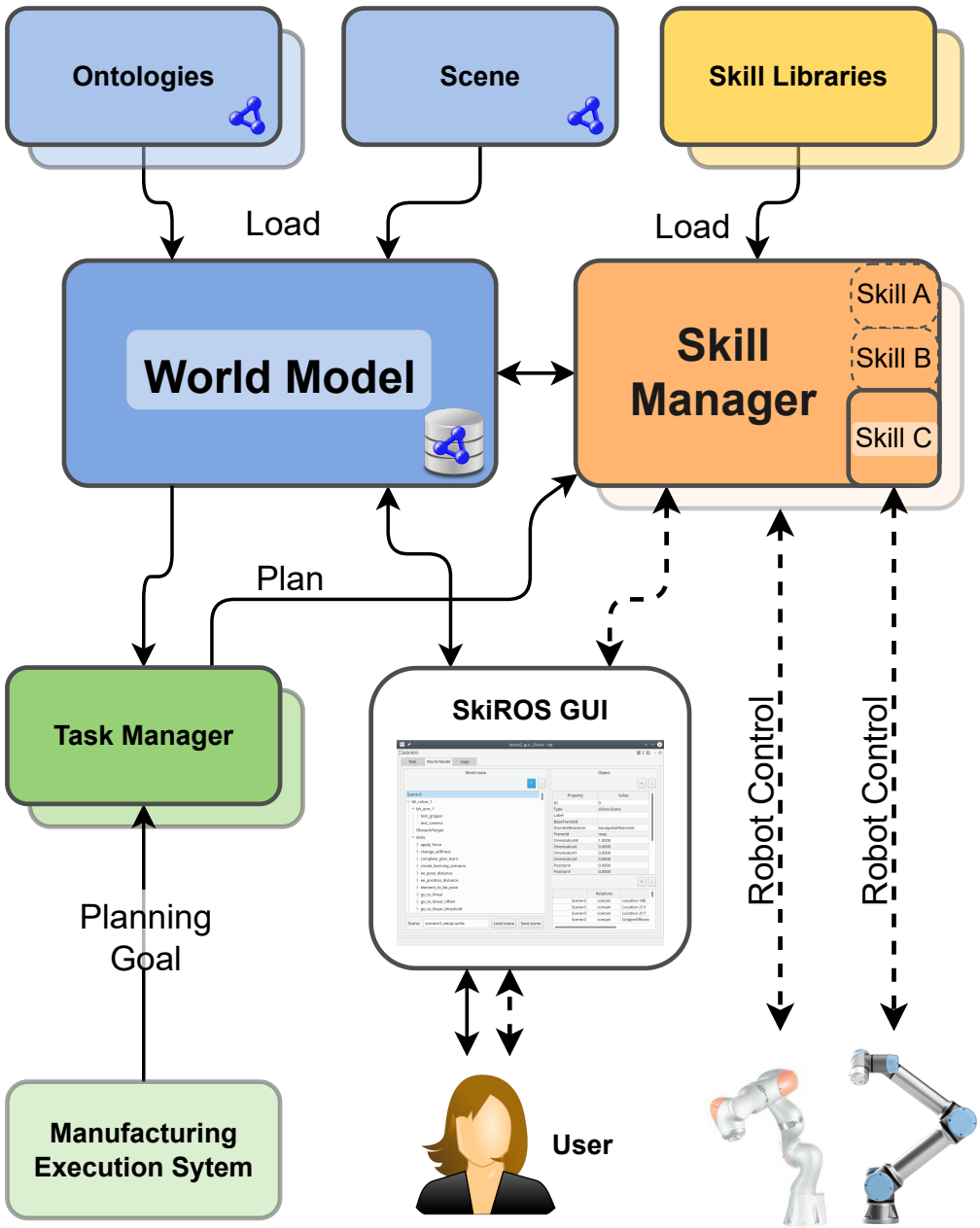


Figure 8: The architecture of SkiROS2. The world model (WM) stores the knowledge and the relations. The skill manager (SM) executes the skills that ultimatelly interface with the hardware.The task manager creates task-level plans. The graphical user interface (GUI) provides a low entry hurdle to interact with the system.

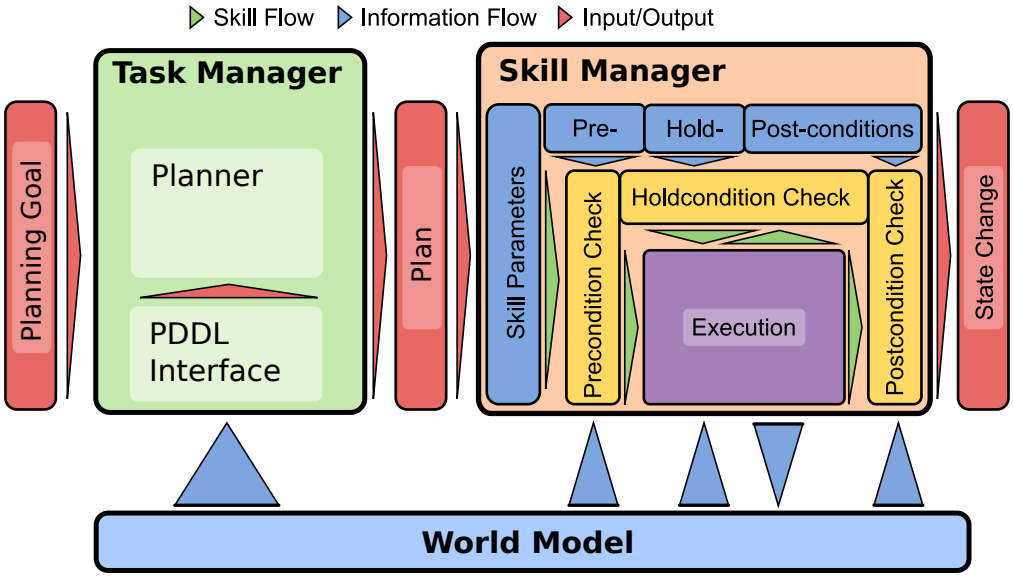


Figure 9: The skill model and the access to knowledge in the WM. The task manager creates a plan based on the knowledge in the WM. The skill manager executes the skills and skills can access the WM to query and modify the knowledge. Furthermore, the skill manager automatically parameterizes the skills based on the knowledge in the WM and performs the post-condtiion check.

- **Compound skills**: Compound skills are skills that are composed of other compound skills or primitive skills. They are implemented as BTs.

SkiROS2's versatility is demonstrated in ***Paper*** 2, where it was used to perform contact-rich wiping tasks on various surfaces with different robots, showcasing its robot-agnostic and task-agnostic capabilities in combination with planning.

In the context of learning with robot skills, it is assumed that a set of skills exists that can address the task at hand. The pre- and post-conditions of these skills are leveraged in conjunction with a planner to tackle new tasks, illustrating the platform's adaptability and potential in robotic learning.

7.2 Motion Representation and Execution

SkiROS2 is designed with flexibility in mind, making no specific assumptions about the type of motion representation or the category of robot used. However, this thesis concentrates on manipulators, considering their widespread applica-

tion and relevance in complex tasks.

To effectively address a broad spectrum of challenging tasks, a motion representation that is both explicit and modular is essential. The key requirements include the ability to apply controlled wrenches with the end effector (EE) and to vary the stiffness of the compliant control. The chosen representation should obviate the need for additional downtime associated with teaching and be suitable for contact-rich tasks. To meet these criteria, we adopt the formulation of BTs with motion generators (MGs) as introduced in [Rov+18].

Trajectories can be generated using tools such as the Cartesian trajectory generator[6], which facilitates the application of overlay motions. These overlay motions can be employed for various purposes, such as searching for a hole or creating the wiping motion demonstrated in ***Paper*** 2.

For position-controlled arms, such as the UR arms, compliant control can be achieved through forward dynamics control [SRD17]. In contrast, for torque-controlled arms such as the *KUKA iiwa*, as shown in Figure 1, or the *Franka Emika Robot (Panda)*, Cartesian impedance control is a more natural and stable choice. The Cartesian impedance controller implementation in [MS22][7] offers the advantage of being applicable not only to real robots, but also seamlessly integrable into simulations.

7.3 Learning Pipeline

The main learning pipeline is introduced in ***Paper*** 4 and shown in Figure 10. It is also used in ***Papers*** 5, 6 and in a modified form also in ***Paper*** 7. It consists of the following steps:

1. **Task goal specification**: The task is specified by the operator. The operator can use a GUI to specify the goal in PDDL.
2. **Task Planning**: The skills that are required to solve the task are identified and the plan is created. The parameters that are not specified in the plan are automatically identified and a learning scenario is created.
3. **Learning Scenario completion**: The learning scenario is complemented by adding reward functions and specifying the parameters of the learning process. Additionally, priors for the optimum can be injected or the learning can be constrained to a specific region of the parameter space.

[6] `https://github.com/matthias-mayr/cartesian_trajectory_generator`
[7] `https://github.com/matthias-mayr/Cartesian-Impedance-Controller`

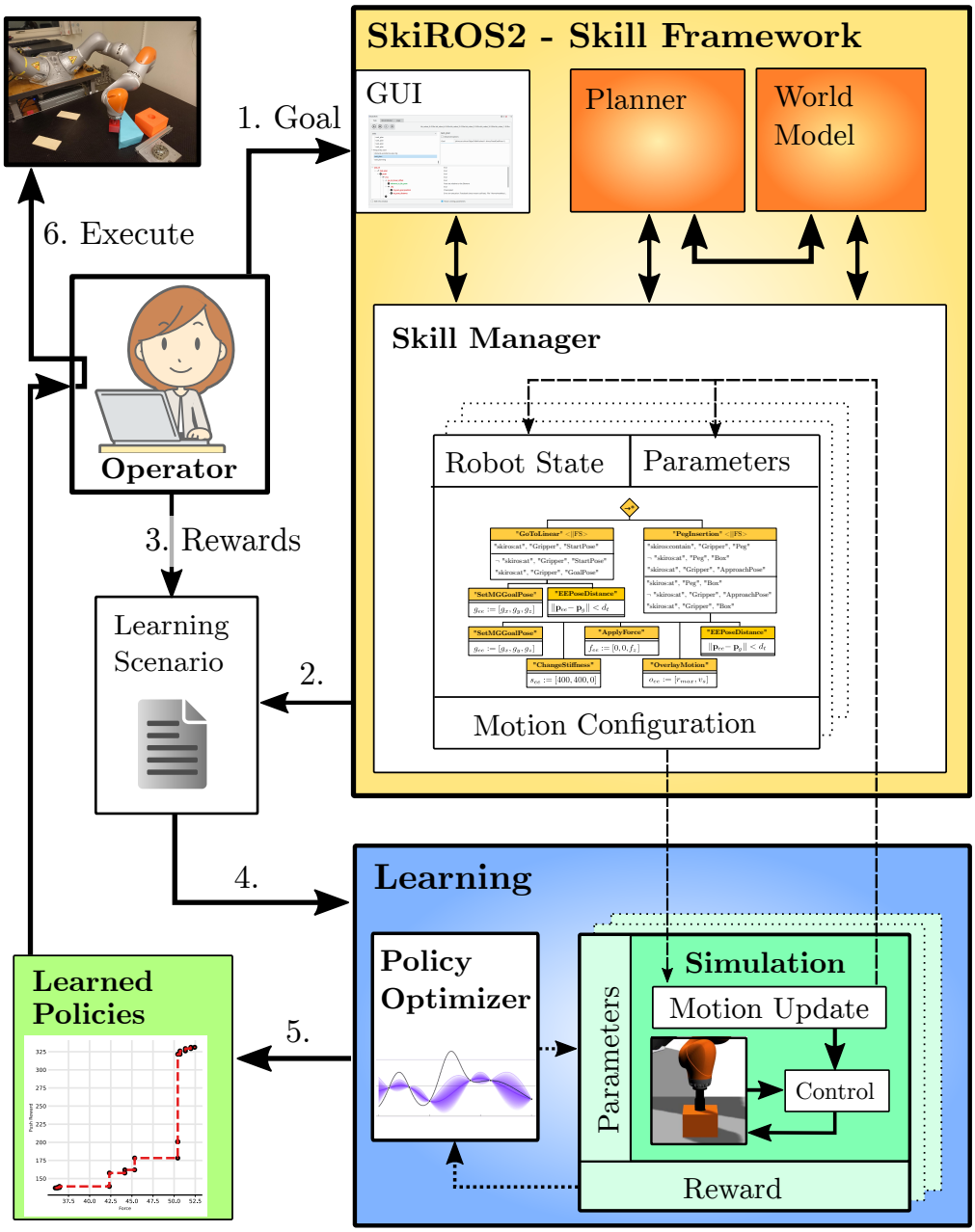


Figure 10: The learning pipeline. 1) a goal for the task is specified and given to the planner. 2) The unspecified parameters in the skills in the plan are identified and a learning scenario is created. 3) The reward functions are added. 4) learning is started. 5) the learning outcome is provided to the operator and 6) the chosen policy is executed on the target system.

4. **Learning**: The learning is started in the respective domain, which is typically a simulation, but can also be a real system. The policy optimization is performed with a black-box optimization approach such as BO.

5. **Learning Outcome**: Upon completion of the learning process, the outcomes are presented to the operator.

6. **Policy Selection**: The operator selects the policy that is used for the execution of the task on the real robot system.

7.4 Learning and Learning Outcome

After the PDDL goal of the task is specified, the structure of the BT policy is provided by the planner. Together with the current state of the scene of the WM, the plan is saved in order to perform the same learning episodes based on the same knowledge. A learning scenario, as exemplified in Listing 3, is then created by identifying the unspecified parameters in the skills. The types of parameters and their possible values can be automatically read from the WM. The learning scenario description is completed by adding the reward functions and specifying the parameters of the learning process.

Robot tasks often require to balance multiple considerations, such as minimizing force application, maintaining safe distances from objects, and preferring solutions with lower velocities. These considerations frequently compete with the primary task goal, posing a challenge in achieving an optimal trade-off. In traditional RL problems, secondary objectives are often overlooked, with the reward functions focusing solely on the primary task goal. An alternative approach involves combining reward functions using a weighted sum, which necessitates careful tuning.

A more nuanced approach is to actively frame the problem as a multi-objective one. Each reward function is assigned to a specific objectives. In case multiple reward functions are necessary for a single objective, weights can be still be assigned to reward functions that contribute to the same objective. In our experience it is much easier to weight reward function within a single objective. To facilitate this process, a library of reward functions is provided, offering convenience and flexibility in defining the learning objectives.

The learning process, as depicted in step 5 of Figure 10, focuses on optimizing the parameters of the skills. To enhance the robustness of the learned policy, it can be evaluated multiple times with domain randomizations. While this concept was initially introduced in ***Paper*** 4 with a fixed number of evaluations,

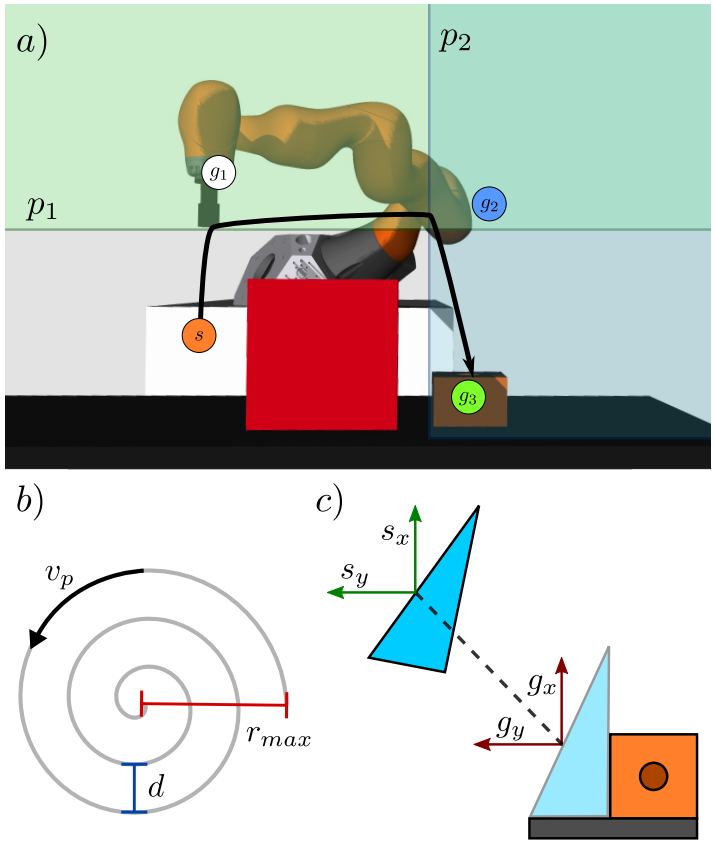


Figure 11: A depiction of learnable parameters of different tasks. a) An obstacle avoidance task with the parametric goal points g_1 and g_2 and the adjustable thresholds p_1 and p_2 in one possible motion configuration. b) The spiral of the search motion a of peg insertion is defined by the pitch d, the maximal radius r_{max} and the path velocity v_p. In addition, a downward force that is applied by the EE is set. c) The learnable offsets for the start and goal location of a push task. ©2022 IEEE

```
1  "application_name": "push_task",
2  "max_fun_evals": 100,
3  "episode_length": 12,
4  "evals_per_param_config": 7,
5  "learning_platform": "sim",
6  "scene": "polyhedron",
7  "domain_randomization": true,
8  "optimizer": "hypermapper",
9  "optimizer_config":
10 {
11    "design_of_experiment": {
12       "doe_type": "random sampling",
13       "number_of_samples": 8
14    },
15    "input_parameters": {
16       "offset_x_start": {
17          "values": [
18             -0.3,
19             0.3
20          ],
21          "parameter_type": "real",
22          "prior" : "custom_gaussian",
23          "custom_gaussian_prior_means": [0.1],
24          "custom_gaussian_prior_stds": [0.1]
25       },
26       [...]
27    },
28    "models": {
29       "model": "gaussian_process"
30    },
31 },
32 "rewards": {
33    "EndEffectorReferencePosition": {
34       "objective": "force",
35       "type": "EndEffectorReferencePosition",
36       "link_name": "peg",
37       "weight": 1.0,
38       [...]
39 },
40 "robot": {
41    "setup_name": "bh_rss_polyhedron",
42    "tool": "peg_70mm_hr",
43 },
44 [...]
```

Listing 3: An excerpt from the underlying learning scenario definition. It contains the configuration of the learning process, the configured reward functions (Lines 32-39) and the robot system (Lines 40-43). The parameters that are to be learned are defined after line 15 and a prior is injected (Lines 21-24).

in ***Paper*** 7 it was developed to an adaptive approach based on the uncertainty of the rewards, further refining the learning outcomes.

In the research presented in ***Paper*** 3, the learning process employs the covariance matrix adaptation evolution strategy (CMA-ES) [Han06], a black-box optimization algorithm renowned for its efficacy in high-dimensional problems involving real numbers. CMA-ES, a gradient-free algorithm, utilizes a covariance matrix to adaptively modify the search distribution, making it particularly suitable for complex optimization tasks in robotics. A key advantage of using CMA-ES in this context is its ability to return the mean of the last distribution rather than merely the best observed solution. This feature is crucial for providing operators with a policy that demonstrates robustness to minor perturbations, such as those encountered in Sim2Real transfer. This robustness is essential to ensure that the learned policies are not only effective in simulation, but also transferable and reliable in real-world applications.

Despite its strengths, CMA-ES has limitations, particularly in multi-objective optimization scenarios and in handling diverse parameter types, such as categorical variables. These limitations led to the adoption of Bayesian optimization (BO) [BCd10] in subsequent research, as detailed in ***Paper*** 4. BO has since been employed in ***Papers*** 5, 6, 7, offering a more versatile and effective approach for optimizing complex robotic tasks.

Learning Outcome

In a single-objective problem setting, the learning outcome is typically a single policy that is returned to the operator. Additionally, it is also possible to return some more policies in case the best one did not meet the expectations.

In contrast, a multi-objective problem setting yields a set of policies, each representing an optimal trade-off between the various objectives. This collection of policies forms what is known as the Pareto frontier, as illustrated in Figure 12. Each point on this frontier, depicted as a blue dot in the figure, corresponds to a policy with a unique parameter set. The ideal solutions for the task, characterized by high task performance and low safety penalty, would be located in the upper left corner of that Pareto frontier. However, such ideal solutions are often unattainable, necessitating a compromise between competing objectives. The hypervolume indicator (HVI), defined as the area between the Pareto-optimal points and a reference point, serves as a valuable metric for evaluating the extent of learning and assessing convergence.

A critical aspect of the learning outcome is its interpretability and the potential for mitigation. The outcome in this context is a BT policy utilizing well-crafted, existing skills. The parameters of these skills are often semantically well-defined

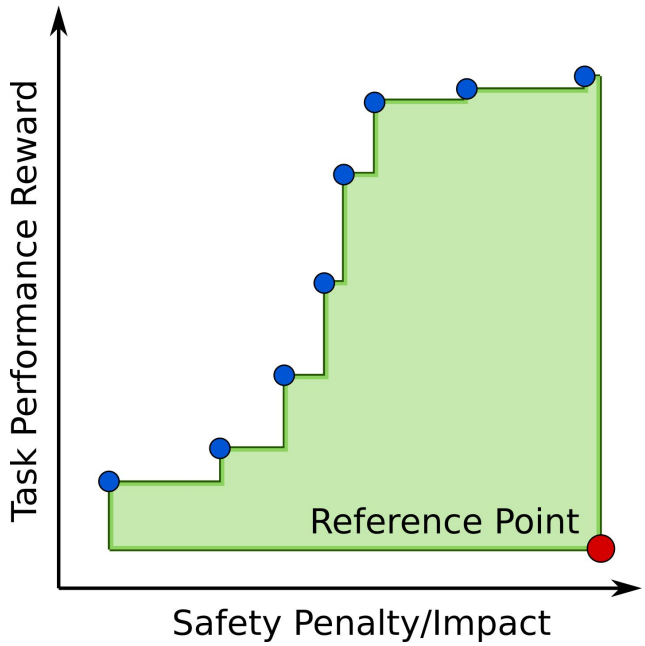


Figure 12: A sketch of the Pareto frontier of a learning problem. The Pareto frontier is the set of all non-dominated solutions marked in blue. The lower right red dot is the reference point to calculate the hypervolume indicator (HVI). The area of the HVI is shown in green.

and interpretable, enabling the operator to comprehend the policy and, if necessary, adjust the skill parameters to mitigate undesired behavior. This level of interpretability and control stands in stark contrast to policies derived from DNN, which are typically less interpretable and offer limited opportunities for operator intervention to correct undesired behaviors.

7.5 Prior Knowledge

The high-dimensional and continuous nature of state and action spaces in robotics presents significant challenges in doing the right thing at the right time. This complexity is particularly pronounced in robot learning, where algorithms often struggle to identify effective policies within a reasonable timeframe. However, leveraging prior knowledge can significantly enhance the learning process.

Prior knowledge about the preconditions and effects of actions serves as a valuable resource in the learning process. This knowledge, typically used in planners, is effectively employed in ***Papers*** 4, 5, 6, 7. During the learning phase, the

search spaces for parameters can be constrained to values predetermined by skill engineers, offering a more directed and efficient exploration of the action space.

The proposed learning strategy incorporates a feasibility metric alongside the traditional reward calculation. This metric evaluates the practicality of each policy, considering factors such as proximity to fragile obstacles. Such feasibility assessments help identify invalid policies that might be overlooked due to inaccuracies in reward formulations. This approach is exemplified in ***Paper*** 6 as part of the Performance and Feasibilty Model (PerF) model.
Furthermore, BO frameworks like [NKO19] allow to define constraints on combinations of parameters with rules such as $x_1 + x_2 < 128$ with x_1 and x_2 being parameters.

In ***Paper*** 5, we assess how parameter priors for the optimum injected with πBO [Car+22] can impact the learning of tasks. The results show that a single prior given by an operator can drastically increase the learning speed. Furthermore, it is shown that learning can also profit from a set of previous experiences by utilizing multiple parameter sets to form a multivariate prior. A transfer between tasks is also performed the cascaded version of our approach in ***Paper*** 7. Since most tasks include the usage of multiple skills, the modular structure of BTs can be exploited. In the cascaded version of *BeBOP*, the first subtask is learned in isolation first and once the learning has converged, the best-performing policy is used as a prior to learn the next bigger BT.

An interesting insight from ***Paper*** 5 is that well-formulated operator priors not only expedite, but can also enhance safety. For instance, in a peg insertion task, the average force applied during learning with operator priors was substantially lower than without priors, while still learning much faster. Similarly, in an obstacle avoidance task, the use of operator priors significantly reduced the number of forceful collisions. Further safety assurances could be integrated using approaches such as *DeROS* [Ada+16] or *ROSSMarie* [RRK23], which facilitate the formulation of explicit safety rules.

7.6 Task Variations

In the context of Industry 4.0 and modern manufacturing, the capability to adapt to new and unseen task variations is a critical attribute of any robotic system. Ideally, a zero-shot transfer, where a new task is solved immediately without additional training, is possible. However, when this is not feasible, a rapid and efficient learning strategy becomes essential.

The skill-based approach, coupled with knowledge representation in a WM as exemplified by SkiROS2, facilitates the transfer of learning outcomes across various tasks. For example, the peg insertion strategy detailed in ***Papers*** 3, 4, 5, 7 is defined in EE space relative to the hole's coordinate frame. If the WM reflects a new pose of the hole and it is kinematically reachable, the policy is expected to transfer effectively. Similarly, in obstacle avoidance tasks, the conditions in the BT and intermediate goal points can be linked to the object, enabling the transfer of learned policies.

However, this transferability is not universal across all tasks. In some instances, aligning policies with elements in the WM can be challenging. An example is the pushing task in ***Papers*** 4, 5, where factors such as the start and goal locations, as well as the object's center of mass, must be considered.

In response to these challenges, ***Paper*** 6 introduces a model that learns to adapt skill parameters to unseen task variations. This model (*PerF*), based on supervised learning, does not require an understanding of the underlying task structure. It is trained solely on the parameters, rewards, and feasibility metrics from a set of training task variations.

In scenarios where such a model cannot be developed, the techniques for utilizing priors, as discussed in the previous section, offer an efficient method for learning new task variations. These techniques enable rapid adaptation to new scenarios, ensuring that the robotic system remains versatile and effective in a dynamic manufacturing environment.

8 Conclusions

We believe that the evolution of robotic systems necessitates a paradigm shift towards enhanced flexibility and adaptability, catering to the diverse and evolving tasks and environments they will encounter. The research presented herein can contribute to this transformation, leveraging robot skills and knowledge representation to meet the demands of next-generation robotic systems.

The utilized skill model underpins a modular and scalable system design, facilitating easy extension and adaptation to novel tasks and environments. This model, combined with effective task planning, empowers the system to autonomously select and integrate skills to accomplish specific tasks. The integration of a knowledge model further enriches this process, enabling the contextualization of robot behavior and the incorporation of prior knowledge. We have shown how to utilize this in a wide range of scenarios, including contact-rich tasks that are executed on different real robot systems.

A significant contribution of this research is the demonstration of how robotic systems can learn and enhance their performance. The learning framework not only aligns with the use of robot skills but also effectively constrains the learning space, ensuring adherence to desired behaviors. This aspect is particularly crucial in industrial settings, where robots operate in delicate and costly environments.

The ability to formulate priors based on operator experience or historical data enhances the learning process. Additionally, framing the learning challenge as a multi-objective problem provides operators with a spectrum of solutions, facilitating trade-offs between various objectives. The use of BTs in skill formulation further enables modular learning, allowing skills to be independently learned, reused, and recombined in different contexts.

The presented type of learning strategy is dramatically faster state-of-the-art RL algorithms like *HIRO* and *MAPLE* while using the same behavior primitives. By requiring only about 5 % of the training time, this also make learning on real systems much more feasible. Moreover, the produced learning outcome is an interpretable policy that allows for adaptation and mitigations.

While many industrial robot systems are still programmed traditionally, this research demonstrates the feasibility of employing various AI techniques in unison to address the challenges faced by robotic systems. It represents a step towards realizing robots that can be easily reconfigured and learn from their experiences, aligning with the vision of adaptable and intelligent robotic systems.

References

[Ada+16] Sorin Adam et al. “Rule-Based Dynamic Safety Monitoring for Mobile Robots”. In: *Journal of Software Engineering for Robotics* 7 (2016), pp. 120–141.

[Aer+98] Constructions Aeronautiques et al. “Pddl the Planning Domain Definition Language”. In: *Technical Report, Tech. Rep.* (1998).

[ADS14] Erwin Aertbeliën and Joris De Schutter. “eTaSL/eTC: A Constraint-Based Task Specification Language and Robot Controller Using Expression Graphs”. In: *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*. Sept. 2014, pp. 1540–1546.

[Alb+23] Alexandre Albore et al. “Skill-Based Design of Dependable Robotic Architectures”. In: *Robotics and Autonomous Systems* 160 (Feb. 2023), p. 104318.

[Alb+11] Sebastian Albrecht et al. “Imitating Human Reaching Motions Using Physically Inspired Optimization Principles”. In: *2011 11th IEEE-RAS International Conference on Humanoid Robots*. IEEE, 2011, pp. 602–607.

[And+20] OpenAI: Marcin Andrychowicz et al. “Learning Dexterous In-Hand Manipulation”. In: *The International Journal of Robotics Research* 39.1 (2020), pp. 3–20.

[Arg+09] Brenna D. Argall et al. “A Survey of Robot Learning from Demonstration”. In: *Robotics and Autonomous Systems* 57.5 (May 2009), pp. 469–483.

[Art19] Artiminds Robotics GmbH. *ArtiMinds Robot Programming Suite*. 2019.

[Baa+07] Franz Baader et al., eds. *The Description Logic Handbook: Theory, Implementation and Applications*. 2nd ed. Cambridge: Cambridge University Press, 2007.

[BKV23] Michael Beetz, Gayane Kazhoyan and David Vernon. *The CRAM Cognitive Architecture for Robot Manipulation in Everyday Activities*. Apr. 2023. arXiv: `2304.14119 [cs]`.

[BMT10] Michael Beetz, Lorenz Mösenlechner and Moritz Tenorth. “CRAM — A Cognitive Robot Abstract Machine for Everyday Manipulation in Human Environments”. In: *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*. Oct. 2010, pp. 1012–1017.

[Bee+18] Michael Beetz et al. “Know Rob 2.0 — A 2nd Generation Knowledge Processing Framework for Cognition-Enabled Robotic Agents”. In: *2018 IEEE International Conference on Robotics and Automation (ICRA)*. May 2018, pp. 512–519.

[BPB18] Daniel Beßler, Mihai Pomarlan and Michael Beetz. “OWL-enabled Assembly Planning for Robotic Agents”. In: *Proceedings of the 2018 International Conference on Autonomous Agents*. AAMAS ’18. Stockholm, Sweden, 2018.

[BZS21] Oliver Biggar, Mohammad Zamani and Iman Shames. “An Expressiveness Hierarchy of Behavior Trees and Related Architectures”. In: *IEEE Robotics and Automation Letters* 6.3 (July 2021), pp. 5397–5404.

[BZS22] Oliver Biggar, Mohammad Zamani and Iman Shames. “On Modularity in Reactive Control Architectures, with an Application to Formal Verification”. In: *ACM Transactions on Cyber-Physical Systems* 6.2 (Apr. 2022), 19:1–19:36.

[BM03] Biggs, Geoffrey and MacDonald, Bruce. “A Survey of Robot Programming Systems”. In: *Proceedings of the Australasian Conference on Robotics and Automation, 2003* (2003).

[BCD16] Aude G. Billard, Sylvain Calinon and Rüdiger Dillmann. “Learning from Humans”. In: *Springer Handbook of Robotics*. Ed. by Bruno Siciliano and Oussama Khatib. Springer Handbooks. Cham: Springer International Publishing, 2016, pp. 1995–2014.

[Bøg+12] Simon Bøgh et al. “Does Your Robot Have Skills?” In: *Proceedings of the 43rd International Symposium on Robotics*. VDE Verlag GmbH, 2012.

[BCd10] Eric Brochu, Vlad M. Cora and Nando de Freitas. *A Tutorial on Bayesian Optimization of Expensive Cost Functions, with Application to Active User Modeling and Hierarchical Reinforcement Learning*. Dec. 2010. arXiv: `1012.2599 [cs]`.

[Bro+23] Anthony Brohan et al. *RT-2: Vision-Language-Action Models Transfer Web Knowledge to Robotic Control*. July 2023. arXiv: `2307.15818 [cs]`.

[Bru+07] Davide Brugali et al. “Trends in Robot Software Domain Engineering”. In: *Software Engineering for Experimental Robotics*. Ed. by Davide Brugali. Springer Tracts in Advanced Robotics. Berlin, Heidelberg: Springer, 2007, pp. 3–8.

[Bru+22] Lukas Brunke et al. “Safe Learning in Robotics: From Learning-Based Control to Safe Reinforcement Learning”. In: *Annual Review of Control, Robotics, and Autonomous Systems* 5.1 (May 2022), pp. 411–444.

[Cal+22] Umut Caliskan et al. “Dual Constraint-Based Controllers for Wheeled Mobile Manipulators”. In: *2022 IEEE 18th International Conference on Automation Science and Engineering (CASE)*. Aug. 2022, pp. 1001–1008.

[CA22] Angelo Cangelosi and Minoru Asada. *Cognitive Robotics*. MIT Press, 2022.

[Car+22] Carl Hvarfner et al. “piBO: Augmenting Acquisition Functions with User Beliefs for Bayesian Optimization”. In: *International Conference on Learning Representations*. 2022.

[Cas+15] Michael Cashmore et al. “ROSPlan: Planning in the Robot Operating System”. In: *Proceedings of the International Conference on Automated Planning and Scheduling* 25 (Apr. 2015), pp. 333–341.

[Cel20] Irene Celino. “Who Is This Explanation for? Human Intelligence and Knowledge Graphs for eXplainable AI”. In: *Knowledge Graphs for eXplainable Artificial Intelligence: Foundations, Applications and Challenges*. IOS Press, 2020, pp. 276–285.

[CM18] Konstantinos Chatzilygeroudis and Jean-Baptiste Mouret. “Using Parameterized Black-Box Priors to Scale Up Model-Based Policy Search for Robotics”. In: *2018 IEEE International Conference on Robotics and Automation (ICRA)*. May 2018, pp. 5121–5128.

[Cha+17] Konstantinos Chatzilygeroudis et al. “Black-Box Data-Efficient Policy Search for Robotics”. In: *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Sept. 2017, pp. 51–58.

[Cha+19] Konstantinos Chatzilygeroudis et al. “A Survey on Policy Search Algorithms for Learning Robot Controllers in a Handful of Trials”. In: *IEEE Transactions on Robotics* (2019), pp. 1–20.

[Cir+11] Dan Claudiu Ciresan et al. “Flexible, High Performance Convolutional Neural Networks for Image Classification”. In: *Twenty-Second International Joint Conference on Artificial Intelligence*. Citeseer, 2011.

[CPO18] M. Colledanchise, R. Nattanmai Parasuraman and P. Ogren. “Learning of Behavior Trees for Autonomous Agents”. In: *IEEE Transactions on Games* (2018), pp. 1–1.

[CAÖ19] Michele Colledanchise, Diogo Almeida and Petter Ögren. “Towards Blended Reactive Planning and Acting Using Behavior Trees”. In: *2019 International Conference on Robotics and Automation (ICRA)*. May 2019, pp. 8839–8845.

[CÖ14] Michele Colledanchise and Petter Ögren. “How Behavior Trees Modularize Robustness and Safety in Hybrid Systems”. In: *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*. Sept. 2014, pp. 1482–1488.

[CÖ17a] Michele Colledanchise and Petter Ögren. *Behavior Trees in Robotics and AI: An Introduction*. Chapman \& Hall/CRC Press, Aug. 2017.

[Cro+17] Matthew Crosby et al. “Integrating Mission and Task Planning in an Industrial Robotics Framework”. In: *Twenty-Seventh International Conference on Automated Planning and Scheduling*. June 2017.

[DS+07] Joris De Schutter et al. “Constraint-Based Task Specification and Estimation for Sensor-Based Robot Systems in the Presence of Geometric Uncertainty”. In: *The International Journal of Robotics Research* 26.5 (May 2007), pp. 433–455.

[DNP13] Marc Peter Deisenroth, Gerhard Neumann and Jan Peters. “A Survey on Policy Search for Robotics”. In: *Foundations and Trends® in Robotics* 2.1–2 (Aug. 2013), pp. 1–142.

[Den+16] Miha Deniša et al. “A Review of Compliant Movement Primitives”. In: *Robot Control*. InTech, Oct. 2016.

[Der08] Derek Greer. *The Art of Separation of Concerns · Aspiring Craftsman*. http://aspiringcraftsman.com/2008/01/03/art-of-separation-of-concerns/. Mar. 2008.

[Elg+23] Íñigo Elguea-Aguinaco et al. “A Review on Reinforcement Learning for Contact-Rich Robotic Manipulation Tasks”. In: *Robotics and Computer-Integrated Manufacturing* 81 (June 2023), p. 102517.

[Eri+19] David Eriksson et al. “Scalable Global Optimization via Local Bayesian Optimization”. In: *Advances in Neural Information Processing Systems*. Vol. 32. Curran Associates, Inc., 2019.

[FN71] Richard E. Fikes and Nils J. Nilsson. “Strips: A New Approach to the Application of Theorem Proving to Problem Solving”. In: *Artificial Intelligence* 2.3-4 (Dec. 1971), pp. 189–208.

[Fra18] Peter I. Frazier. *A Tutorial on Bayesian Optimization*. July 2018. arXiv: `1807.02811 [cs, math, stat]`.

[Fre+19] K. French et al. “Learning Behavior Trees From Demonstration”. In: *2019 International Conference on Robotics and Automation (ICRA)*. May 2019, pp. 7791–7797.

[GF15] Javier Garcıa and Fernando Fernandez. “A Comprehensive Survey on Safe Reinforcement Learning”. In: *Journal of Machine Learning Research* 16 (2015), pp. 1437–1480.

[GNT04] Malik Ghallab, Dana Nau and Paolo Traverso. *Automated Planning: Theory and Practice*. Elsevier, 2004.

[Ghz+20] Razan Ghzouli et al. “Behavior Trees in Action: A Study of Robotics Applications”. In: *Proceedings of the 13th ACM SIGPLAN International Conference on Software Language Engineering*. SLE 2020. New York, NY, USA: Association for Computing Machinery, Nov. 2020, pp. 196–209.

[GLL99] A. Girault, Bilung Lee and E.A. Lee. “Hierarchical Finite State Machines with Multiple Concurrency Models”. In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 18.6 (June 1999), pp. 742–760.

[Gru93] Thomas R. Gruber. “A Translation Approach to Portable Ontology Specifications”. In: *Knowledge Acquisition* 5.2 (June 1993), pp. 199–220.

[Gug+23] Simona Gugliermo et al. “Learning Behavior Trees From Planning Experts Using Decision Tree and Logic Factorization”. In: *IEEE Robotics and Automation Letters* 8.6 (June 2023), pp. 3534–3541.

[Gus+22] Oscar Gustavsson et al. “Combining Context Awareness and Planning to Learn Behavior Trees from Demonstration”. In: *2022 31st IEEE International Conference on Robot and Human Interactive Communication (RO-MAN)*. Aug. 2022, pp. 1153–1160.

[Han06] Nikolaus Hansen. “Towards a New Evolutionary Computation”. In: *Studies in Fuzziness and Soft Computing* 192 (2006), pp. 75–102.

[Hel+23] Erik Orm Hellsten et al. *High-Dimensional Bayesian Optimization with Group Testing*. Oct. 2023. arXiv: 2310.03515.

[HKR09] Pascal Hitzler, Markus Krötzsch and Sebastian Rudolph. *Foundations of Semantic Web Technologies*. Chapman & Hall/CRC, 2009.

[HMU01] John E Hopcroft, Rajeev Motwani and Jeffrey D Ullman. “Introduction to Automata Theory, Languages, and Computation”. In: *Acm Sigact News* 32.1 (2001), pp. 60–65.

[15i] "IEEE Standard Ontologies for Robotics and Automation". In: *IEEE Std 1872-2015* (Apr. 2015), pp. 1–60.

[Iov+21] Matteo Iovino et al. "Learning Behavior Trees with Genetic Programming in Unpredictable Environments". In: *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2021, pp. 4591–4597.

[Iov+22] Matteo Iovino et al. "A Survey of Behavior Trees in Robotics and AI". In: *Robotics and Autonomous Systems* 154 (Aug. 2022), p. 104096.

[Iov+23a] Matteo Iovino et al. "A Framework for Learning Behavior Trees in Collaborative Robotic Applications". In: *2023 IEEE International Conference on Automation Science and Engineering (CASE)* (2023).

[Iov+23b] Matteo Iovino et al. "On the Programming Effort Required to Generate Behavior Trees and Finite State Machines for Robotic Applications". In: *2023 IEEE International Conference on Robotics and Automation (ICRA)*. London, United Kingdom: IEEE, May 2023, pp. 5807–5813.

[ICK16] Irving M. Copi, Carl Cohen and Kenneth McMahon. *Introduction to Logic*. 14th ed. New York: Routledge, July 2016.

[JDJ17] Stephen James, Andrew J. Davison and Edward Johns. "Transferring End-to-End Visuomotor Control from Simulation to Real World for a Multi-Stage Task". In: *Proceedings of the 1st Annual Conference on Robot Learning*. PMLR, Oct. 2017, pp. 334–343.

[KRJ17] Martin Karlsson, Anders Robertsson and Rolf Johansson. "Autonomous Interpretation of Demonstrations for Modification of Dynamical Movement Primitives". In: *2017 IEEE International Conference on Robotics and Automation (ICRA)*. May 2017, pp. 316–321.

[KRJ18] Martin Karlsson, Anders Robertsson and Rolf Johansson. "Convergence of Dynamical Movement Primitives with Temporal Coupling". In: *2018 European Control Conference (ECC)*. June 2018, pp. 32–39.

[Kar+17] Martin Karlsson et al. "Two-Degree-of-Freedom Control for Trajectory Tracking and Perturbation Recovery during Execution of Dynamical Movement Primitives". In: *IFAC-PapersOnLine*. 20th IFAC World Congress 50.1 (July 2017), pp. 1923–1930.

[KB17] Gayane Kazhoyan and Michael Beetz. “Programming Robotic Agents with Action Descriptions”. In: *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Sept. 2017, pp. 103–108.

[KB11] S. M. Khansari-Zadeh and A. Billard. “Learning Stable Nonlinear Dynamical Systems With Gaussian Mixture Models”. In: *IEEE Transactions on Robotics* 27.5 (Oct. 2011), pp. 943–957.

[KP14] Jens Kober and Jan Peters. “Reinforcement Learning in Robotics: A Survey”. In: *Learning Motor Skills: From Algorithms to Robot Experiments*. Ed. by Jens Kober and Jan Peters. Springer Tracts in Advanced Robotics. Cham: Springer International Publishing, 2014, pp. 9–67.

[KCC13] Petar Kormushev, Sylvain Calinon and Darwin G. Caldwell. “Reinforcement Learning in Robotics: Applications and Real-World Challenges”. In: *Robotics* 2.3 (Sept. 2013), pp. 122–148.

[Kov12] Daniel L Kovacs. “A Multi-Agent Extension of PDDL3.1”. In: *Proceedings of the 3rd Workshop on the International Planning Competition (IPC). 22nd International Conference on Automated Planning and Scheduling (ICAPS-2012)* (2012), p. 19.

[KSH12] Alex Krizhevsky, Ilya Sutskever and Geoffrey E Hinton. “Imagenet Classification with Deep Convolutional Neural Networks”. In: *Advances in neural information processing systems* 25 (2012).

[Kru+19] Volker Krueger et al. “Testing the Vertical and Cyber-Physical Integration of Cognitive Robots in Manufacturing”. In: *Robotics and Computer-Integrated Manufacturing* 57 (June 2019), pp. 213–229.

[Krü+07] Volker Krüger et al. “The Meaning of Action: A Review on Action Recognition and Mapping”. In: *Advanced Robotics* 21.13 (Jan. 2007), pp. 1473–1501.

[Lei+19] Daniel Leidner et al. “Cognition-Enabled Robotic Wiping: Representation, Planning, Execution, and Interpretation”. In: *Robotics and Autonomous Systems* 114 (Apr. 2019), pp. 199–216.

[Len95] Douglas B. Lenat. “CYC: A Large-Scale Investment in Knowledge Infrastructure”. In: *Communications of the ACM* 38.11 (Nov. 1995), pp. 33–38.

[Lev+16] Sergey Levine et al. “End-to-End Training of Deep Visuomotor Policies”. In: *The Journal of Machine Learning Research* 17.1 (2016), pp. 1334–1373.

[LBC10] Chong-U Lim, Robin Baumgarten and Simon Colton. “Evolving Behaviour Trees for the Commercial Game DEFCON”. In: *Applications of Evolutionary Computation*. Ed. by Cecilia Di Chio et al. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2010, pp. 100–110.

[MJ13] Stenmark Maj and Malec Jacek. “Knowledge-Based Industrial Robotics”. In: *Frontiers in Artificial Intelligence and Applications* (2013), pp. 265–274.

[Mar+21] Francisco Martín et al. “PlanSys2: A Planning System Framework for ROS2”. In: *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Sept. 2021, pp. 9742–9749.

[Mar+14] A. Marzinotto et al. “Towards a Unified Behavior Trees Framework for Robot Control”. In: *2014 IEEE International Conference on Robotics and Automation (ICRA)*. May 2014, pp. 5420–5427.

[MS22] Matthias Mayr and Julian M Salt-Ducaju. “A C++ Implementation of a Cartesian Impedance Controller for Robotic Manipulators”. In: *arXiv preprint arXiv:2212.11215* (2022). arXiv: 2212.11215.

[May+21] Matthias Mayr et al. “Learning of Parameters in Behavior Trees for Movement Skills”. In: *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2021 IEEE/RSJ International Conference on Intelligent Robots and Systems. IEEE, 2021, pp. 7572–7579.

[Meh+20] Bhairav Mehta et al. “Active Domain Randomization”. In: *Conference on Robot Learning*. PMLR, May 2020, pp. 1162–1176.

[NKO19] Luigi Nardi, David Koeplinger and Kunle Olukotun. “Practical Design Space Exploration”. In: *2019 IEEE 27th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*. Oct. 2019, pp. 347–358.

[NLZ22] Soroush Nasiriany, Huihan Liu and Yuke Zhu. “Augmenting Reinforcement Learning with Behavior Primitives for Diverse Manipulation Tasks”. In: *2022 International Conference on Robotics and Automation (ICRA)*. IEEE, 2022, pp. 7477–7484.

[NP01] Ian Niles and Adam Pease. “Towards a Standard Upper Ontology”. In: *Proceedings of the International Conference on Formal Ontology in Information Systems - Volume 2001*. FOIS ’01. New York, NY, USA: Association for Computing Machinery, Oct. 2001, pp. 2–9.

[Nil84] N Nilsson. “Shakey the Robot. Technical Report 323”. In: *SRI, Menlo Park, CA* (1984).

[Oli+19] Alberto Olivares-Alarcos et al. “A Review and Comparison of Ontology-Based Approaches to Robot Autonomy”. In: *The Knowledge Engineering Review* 34 (2019/ed).

[Ope+19] OpenAI et al. *Dota 2 with Large Scale Deep Reinforcement Learning*. Dec. 2019. arXiv: `1912.06680 [cs, stat]`.

[PNP22] Leonard Papenmeier, Luigi Nardi and Matthias Poloczek. “Increasing the Scope as You Learn: Adaptive Bayesian Optimization in Nested Subspaces”. In: *Advances in Neural Information Processing Systems* 35 (Dec. 2022), pp. 11586–11601.

[PNP23] Leonard Papenmeier, Luigi Nardi and Matthias Poloczek. “Bounce: Reliable High-Dimensional Bayesian Optimization for Combinatorial and Mixed Spaces”. In: *Thirty-Seventh Conference on Neural Information Processing Systems*. Nov. 2023.

[Pau+20] Supratik Paul et al. “Robust Reinforcement Learning with Bayesian Optimisation and Quadrature”. In: *The Journal of Machine Learning Research* 21.1 (2020), pp. 6020–6050.

[Pax+17] Chris Paxton et al. “CoSTAR: Instructing Collaborative Robots with Behavior Trees and Vision”. In: *2017 IEEE International Conference on Robotics and Automation (ICRA)*. May 2017, pp. 564–571.

[PHK14] M. R. Pedersen, D. L. Herzog and V. Krüger. “Intuitive Skill-Level Programming of Industrial Handling Tasks on a Mobile Manipulator”. In: *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*. Sept. 2014, pp. 4523–4530.

[Ped+16] Mikkel Rath Pedersen et al. “Robot Skills for Manufacturing: From Concept to Industrial Deployment”. In: *Robotics and Computer-Integrated Manufacturing* 37 (Feb. 2016), pp. 282–291.

[Pen+18] Xue Bin Peng et al. “Sim-to-Real Transfer of Robotic Control with Dynamics Randomization”. In: *2018 IEEE International Conference on Robotics and Automation (ICRA)*. May 2018, pp. 3803–3810.

[Per+11] Diego Perez et al. “Evolving Behaviour Trees for the Mario AI Competition Using Grammatical Evolution”. In: *Applications of Evolutionary Computation*. Ed. by Cecilia Di Chio et al. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2011, pp. 123–132.

[Pet+14] Tadej Petrič et al. “Online Learning of Task-Specific Dynamics for Periodic Tasks”. In: *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*. Sept. 2014, pp. 1790–1795.

[PN17] Athanasios S. Polydoros and Lazaros Nalpantidis. “Survey of Model-Based Reinforcement Learning: Applications on Robotics”. In: *Journal of Intelligent & Robotic Systems* 86.2 (May 2017), pp. 153–173.

[RBC14] Karinne Ramirez-Amaro, Michael Beetz and Gordon Cheng. “Automatic Segmentation and Recognition of Human Activities from Observation Based on Semantic Reasoning”. In: *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2014, pp. 5043–5048.

[RBC15] Karinne Ramirez-Amaro, Michael Beetz and Gordon Cheng. “Understanding the Intention of Human Activities through Semantic Perception: Observation, Understanding and Execution on a Humanoid Robot”. In: *Advanced Robotics* 29.5 (Mar. 2015), pp. 345–362.

[RBC17] Karinne Ramirez-Amaro, Michael Beetz and Gordon Cheng. “Transferring Skills to Humanoid Robots by Extracting Semantic Representations from Observations of Human Activities”. In: *Artificial Intelligence*. Special Issue on AI and Robotics 247 (June 2017), pp. 95–118.

[RW06] Carl Edward Rasmussen and Christopher K. I. Williams. *Gaussian Processes for Machine Learning*. Adaptive Computation and Machine Learning. Cambridge, Mass: MIT Press, 2006.

[RSN16] Simon Razniewski, Ognjen Savkovic and Werner Nutt. “Turning the Partial-Closed World Assumption Upside Down”. In: *Proceedings of the 10th Alberto Mendelzon International Workshop on Foundations of Data Management, Panama City, Panama, May 8-10, 2016*. Vol. 1644. CEUR-WS, 2016.

[RRK23] Momina Rizwan, Christoph Reichenbach and Volker Krüger. “ROSSMARie: A Domain-Specific Language to Express Dynamic Safety Rules and Recovery Strategies for Autonomous Robots”. In: *Second Workshop on Quality and Reliability Assessment of Robotic Software Architectures and Components*. London, UK, June 2023.

[Ros04] J Rosell. “Assembly and Task Planning Using Petri Nets: A Survey”. In: *Proceedings of the Institution of Mechanical Engineers, Part B: Journal of Engineering Manufacture* 218.8 (Aug. 2004), pp. 987–994.

[RGK17] F. Rovida, B. Grossmann and V. Krüger. “Extended Behavior Trees for Quick Definition of Flexible Robotic Tasks”. In: *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Sept. 2017, pp. 6793–6800.

[Rov+18] F. Rovida et al. “Motion Generators Combined with Behavior Trees: A Novel Approach to Skill Modelling”. In: *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Oct. 2018, pp. 5964–5971.

[Rov17] Francesco Rovida. “A Software Platform to Develop and Execute Kitting Tasks on Industrial Cyber-Physical Systems”. PhD thesis. Apr. 2017.

[Rov+17] Francesco Rovida et al. “SkiROS—A Skill-Based Robot Control Platform on Top of ROS”. In: *Robot Operating System (ROS): The Complete Reference (Volume 2)*. Ed. by Anis Koubaa. Studies in Computational Intelligence. Cham: Springer International Publishing, 2017, pp. 121–160.

[RND10] Stuart J. Russell, Peter Norvig and Ernest Davis. *Artificial Intelligence: A Modern Approach*. 3rd ed. Prentice Hall Series in Artificial Intelligence. Upper Saddle River: Prentice Hall, 2010.

[Ryt23] Rytter, Martin. *Five Ways to Program a Cobot*. https://www.universal-robots.com/developer/insights/five-ways-to-program-a-cobot/, https://www.robots.com/developer/insights/five-ways-to-program-a-cobot/. 2023.

[Git] *SMACC2*. ROBOSOFT AI. Nov. 2023.

[Sav+23] Matteo Saveriano et al. “Dynamic Movement Primitives in Robotics: A Tutorial Survey”. In: *The International Journal of Robotics Research* 42.13 (2023), pp. 1133–1184.

[Sch06a] Stefan Schaal. “Dynamic Movement Primitives -A Framework for Motor Control in Humans and Humanoid Robotics”. In: *Adaptive Motion of Animals and Machines*. Ed. by Hiroshi Kimura et al. Tokyo: Springer, 2006, pp. 261–280.

[Sch06b] Stefan Schaal. “Dynamic Systems: Brain, Body, and Imitation”. In: *ACTION TO LANGUAGE VIA THE MIRROR NEURON SYSTEM* (2006), p. 177.

[SRD17] S. Scherzinger, A. Roennau and R. Dillmann. “Forward Dynamics Compliance Control (FDCC): A New Approach to Cartesian Compliance for Robotic Manipulators”. In: *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Sept. 2017, pp. 4568–4575.

[SKv16] Philipp Schillinger, Stefan Kohlbrecher and Oskar von Stryk. “Human-Robot Collaborative High-Level Control with Application to Rescue Robotics”. In: *2016 IEEE International Conference on Robotics and Automation (ICRA)*. May 2016, pp. 2796–2802.

[Sch17] Klaus Schwab. *The Fourth Industrial Revolution*. Crown, Jan. 2017.

[Sha+16] Bobak Shahriari et al. “Taking the Human Out of the Loop: A Review of Bayesian Optimization”. In: *Proceedings of the IEEE* 104.1 (Jan. 2016), pp. 148–175.

[Sic09] Bruno Siciliano, ed. *Robotics: Modelling, Planning and Control*. Advanced Textbooks in Control and Signal Processing. London: Springer, 2009.

[SK08] Bruno Siciliano and Oussama Khatib, eds. *Springer Handbook of Robotics*. Berlin: Springer, 2008.

[SK16] Bruno Siciliano and Oussama Khatib, eds. *Springer Handbook of Robotics*. Springer Handbooks. Cham: Springer International Publishing, 2016.

[Sil+16] David Silver et al. “Mastering the Game of Go with Deep Neural Networks and Tree Search”. In: *Nature* 529.7587 (Jan. 2016), pp. 484–489.

[Ros] *Smach - ROS Wiki*. http://wiki.ros.org/smach.

[Sou+21] Artur Souza et al. “Bayesian Optimization with a Prior for the Optimum”. In: *Machine Learning and Knowledge Discovery in Databases. Research Track*. Ed. by Nuria Oliver et al. Cham: Springer International Publishing, 2021, pp. 265–296.

[SBB22] Simon Stelter, Georg Bartels and Michael Beetz. “An Open-Source Motion Planning Framework for Mobile Manipulators Using Constraint-Based Task Space Control with Linear MPC”. In: *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Oct. 2022, pp. 1671–1678.

[Ste+15] M. Stenmark et al. “On Distributed Knowledge Bases for Robotized Small-Batch Assembly”. In: *IEEE Transactions on Automation Science and Engineering* 12.2 (Apr. 2015), pp. 519–528.

[Ste17] Maj Stenmark. “Intuitive Instruction of Industrial Robots: A Knowledge-Based Approach”. Doctoral Thesis (Compilation). Lund: Department of Computer Science, Lund University, May 2017.

[SHT17] Maj Stenmark, Mathias Haage and Elin Anna Topp. "Simplified Programming of Re-usable Skills on a Safe Industrial Robot: Prototype and Evaluation". In: *Proceedings of the 2017 ACM/IEEE International Conference on Human-Robot Interaction*. HRI '17. New York, NY, USA: Association for Computing Machinery, Mar. 2017, pp. 463–472.

[SM13] Maj Stenmark and Jacek Malec. "A Helping Hand: Industrial Robotics, Knowledge and User-Oriented Services". In: *AI-based Robotics Workshop, 2013 IEEE/RSJ International Conference on Intelligent Robots and Systems* (2013), p. 6.

[SM15] Maj Stenmark and Jacek Malec. "Knowledge-Based Instruction of Manipulation Tasks for Industrial Robotics". In: *Robotics and Computer-Integrated Manufacturing*. Special Issue on Knowledge Driven Robotics and Manufacturing 33 (June 2015), pp. 56–67.

[Ste+17] Maj Stenmark et al. "Knowledge for Synchronized Dual-Arm Robot Programming". In: *2017 AAAI Fall Symposium Series*. 2017.

[Sty+22] Jonathan Styrud et al. "Combining Planning and Learning of Behavior Trees for Robotic Assembly". In: *2022 International Conference on Robotics and Automation (ICRA)*. May 2022, pp. 11511–11517.

[SGF12] Mari Carmen Suárez-Figueroa, Asunción Gómez-Pérez and Mariano Fernández-López. "The NeOn Methodology for Ontology Engineering". In: *Ontology Engineering in a Networked World*. Ed. by Mari Carmen Suárez-Figueroa et al. Berlin, Heidelberg: Springer, 2012, pp. 9–34.

[SMK12] Ioan A. Sucan, Mark Moll and Lydia E. Kavraki. "The Open Motion Planning Library". In: *IEEE Robotics & Automation Magazine* 19.4 (Dec. 2012), pp. 72–82.

[Sün+18] Niko Sünderhauf et al. "The Limits and Potentials of Deep Learning for Robotics". In: *The International Journal of Robotics Research* 37.4-5 (Apr. 2018), pp. 405–420.

[SKK22] Markku Suomalainen, Yiannis Karayiannidis and Ville Kyrki. "A Survey of Robot Manipulation in Contact". In: *Robotics and Autonomous Systems* 156 (Oct. 2022), p. 104224.

[SB18] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. Second edition. Adaptive Computation and Machine Learning Series. Cambridge, Massachusetts: The MIT Press, 2018.

[TB09] Moritz Tenorth and Michael Beetz. “KNOWROB — Knowledge Processing for Autonomous Personal Robots”. In: *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems*. Oct. 2009, pp. 4261–4266.

[TB13] Moritz Tenorth and Michael Beetz. “KnowRob: A Knowledge Processing Infrastructure for Cognition-Enabled Robots”. In: *The International Journal of Robotics Research* 32.5 (Apr. 2013), pp. 566–590.

[Top+18] Elin A. Topp et al. “Ontology-Based Knowledge Representation for Increased Skill Reusability in Industrial Robots”. In: *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Oct. 2018, pp. 5672–5678.

[VGK19] Patrick Varin, Lev Grossman and Scott Kuindersma. “A Comparison of Action Spaces for Learning Manipulation Tasks”. In: *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2019, pp. 6015–6021.

[Vin+19] Oriol Vinyals et al. “Grandmaster Level in StarCraft II Using Multi-Agent Reinforcement Learning”. In: *Nature* 575.7782 (Nov. 2019), pp. 350–354.

[VK14] Denny Vrandečić and Markus Krötzsch. “Wikidata: A Free Collaborative Knowledgebase”. In: *Communications of the ACM* 57.10 (Sept. 2014), pp. 78–85.

[Wat+22] Adam Wathieu et al. “RE:BT-Espresso: Improving Interpretability and Expressivity of Behavior Trees Learned from Robot Demonstrations”. In: *2022 International Conference on Robotics and Automation (ICRA)*. May 2022, pp. 11518–11524.

[YSS22] Quantao Yang, Johannes A. Stork and Todor Stoyanov. “MPR-RL: Multi-Prior Regularized Reinforcement Learning for Knowledge Transfer”. In: *IEEE Robotics and Automation Letters* 7.3 (July 2022), pp. 7652–7659.

[YSS23] Quantao Yang, Johannes A. Stork and Todor Stoyanov. “Learn from Robot: Transferring Skills for Diverse Manipulation via Cycle Generative Networks”. In: *2023 IEEE 19th International Conference on Automation Science and Engineering (CASE)*. Aug. 2023, pp. 1–6.

[Yan+22] Quantao Yang et al. “Variable Impedance Skill Learning for Contact-Rich Manipulation”. In: *IEEE Robotics and Automation Letters* 7.3 (July 2022), pp. 8391–8398.

[Zec+19] Philipp Zech et al. “Action Representations in Robotics: A Taxonomy and Systematic Classification”. In: *The International Journal of Robotics Research* 38.5 (Apr. 2019), pp. 518–562.

Acronyms

AI artificial intelligence

API application programming interface

BO Bayesian optimization

BT behavior tree

CMA-ES covariance matrix adaptation evolution strategy

CORA Core Ontology for Robotics and Automation

CRAM cognitive robot abstract machine

DMP dynamic movement primitive

DNN deep neural network

DT digital twin

eBT extended behavior tree

EE end effector

EGM external guided motion

EI expected improvement

FRI fast robot interface

FSM finite state machine

GMM Gaussian mixture model

GP Gaussian process

GUI graphical user interface

HFSM hierarchical finite state machine

HTN hierarchical task network

HVI hypervolume indicator

IL imitation learning

iTasC task frame formalism

LfD learning from demonstration

MDP Markov decision process

MES manufacturing execution system

MG motion generator

ML machine learning

NN neural network

OMPL open motion planning library

OWL Web Ontology Language

PerF Performance and Feasibilty Model

PDDL planning domain definition language

RL reinforcement learning

RDF resource description framework

RDFS resource description framework schema

ROS robot operating system

SM skill manager

SOMA Socio-physical Model of Activities

SPARQL SPARQL protocol and rdf query language

STRIPS Stanford Research Institute Problem Solver

SUMO suggested upper merged ontology

UR Universal Robots

URI uniform resource identifier

W3C World Wide Web Consortium

WM world model

Scientific Publications

Contribution Statements

Table 1: Overview of contributions in each paper included in the thesis.

Paper	Concept	Implementation	Evaluation	Writing
I	◔	◔	◑	●
II	◕	◕	◑	◕
III	●	●	◕	◕
IV	◕	◕	◑	◑
V	◕	◕	●	◑
VI	◑	◔	◑	◑
VII	◑	◑	◔	◑

The dark portion of the circle represents the amount of work and responsibilities assigned to Matthias Mayr for each individual step:

- ◔ Matthias Mayr was a minor contributor to the work
- ◑ Matthias Mayr was a contributor to the work
- ◕ Matthias Mayr led and did a majority of the work
- ● Matthias Mayr led and did almost all of the work

Volker Krueger is the main supervisor of Matthias Mayr and was involved in the papers. He contributed to the writing and provided feedback and guidance. He also contributed to the discussions of the design of the concepts and the evaluation of the results.

Paper i: SkiROS2: A skill-based Robot Control Platform for ROS

I designed the concept of the paper and did the complete writing of the paper. I also lead two of the three use-cases presented in the paper. Francesco Rovida lead the piston insertion use-case and implemented the code of SkiROS2.

Paper ii: Using Knowledge Representation and Task Planning for Robot-agnostic Skills on the Example of Contact-Rich Wiping Tasks

I designed the wiping task as an example for the use of compliant control and robot-agnostic skill usage. I also implemented and configured the compliant control solutions and lead the creation of the skills. Faseeh Ahmad contributed to the implementation of the skills. He also lead the experiments on the Universal Robots (UR) that were conducted together with Alexander Duerr. I performed the experiments on the *KUKA iiwa*, evaluated the data and wrote the majority of the paper. Faseeh Ahmad and Alexander Duerr contributed to the writing of the paper.

Paper iii: Learning of Parameters in Behavior Trees for Movement Skills

I selected the reinforcement learning (RL) algorithm and implemented the RL pipeline with behavior trees (BTs). I also lead the experiments and wrote the majority of the paper. Faseeh Ahmad helped with the experiments and participated in later discussions. Konstantinos Chatzilygeroudis is the code owner of the RL framework and participated in the discussions about the next steps. Luigi Nardi provided advice on the parameterization of the optimization algorithms and evaluation advice. All authors contributed to the paper writing process.

Paper iv: Skill-based Multi-objective Reinforcement Learning of Industrial Robot Tasks with Planning and Knowledge Integration

I designed the concept and outlined the framework. I also implemented the RL part and the majority of the integration of SkiROS2. Faseeh Ahmad implemented the planning integration. Luigi Nardi had an advisory role in the

use of Bayesian optimization (BO). The experiments were performed by Faseeh Ahmad and me. I had the largest part in the design and writing of the paper. Faseeh Ahmad and the other authors contributed to the writing of the paper.

Paper v: Learning Skill-based Industrial Robot Tasks with User Priors

I designed the concept and provided most of the implementation. Carl Hvarfner implemented the policy optimization with priors for the optimum. Luigi Nardi had an advisory role in the experiment design and evaluation. I conducted all the experiments in simulation and on the real robot. Carl Hvarfner contributed to the data analysis. I wrote the largest individual share of the paper with contributions from all co-authors.

Paper vi: Learning to Adapt the Parameters of Behavior Trees and Motion Generators (BTMGs) to Task Variations

Together with Faseeh Ahmad we outlined the concept of the paper. I contributed the RL pipeline and the BT implementation. Faseeh Ahmad extended it with the necessary additions to support task variations. He also implemented the model learning and querying. Faseeh Ahmad lead the experiments and I helped with the design and discussion. He wrote majority of the paper. The other author and I helped with the writing of the paper.

Paper vii: BeBOP – Combining Reactive Planning and Bayesian Optimization to Solve Robotic Manipulation Tasks

Jonathan Styrud and I developed the concept and eventually implemented it based on Jonathan Styrud's integration of BTs with the baseline learning approach. The reactive planner integration was provided by Jonathan Styrud and the BO integrations by me. Discussions and iterations were between me and Jonathan Styrud and were later joined by Erik Hellsten. The new uncertainty measure was developed by Jonathan Styrud and Erik Hellsten. Jonathan Styrud evaluated the baseline approach and a majority of the results with me providing the rest. Jonathan Styrud had the largest share in the writing process. I contributed the initial version of the BO chapter that was later taken over by Erik Hellsten. I edited all sections, suggested re-organizations and improvements as well as proof-read the paper.

Paper I

Paper 1

SkiROS2: A skill-based Robot Control Platform for ROS

Matthias Mayr
Lund University
matthias.mayr@cs.lth.se

Francesco Rovida
RiACT ApS, Denmark
f.rovida@riact.eu

Volker Krueger
Lund University
volker.krueger@cs.lth.se

Abstract

The need for autonomous robot systems in both the service and the industrial domain is larger than ever. In the latter, the transition to small batches or even "batch size 1" in production created a need for robot control system architectures that can provide the required flexibility. Such architectures must not only have a sufficient knowledge integration framework. It must also support autonomous mission execution and allow for interchangeability and interoperability between different tasks and robot systems. We introduce *SkiROS2*, a skill-based robot control platform on top of ROS. SkiROS2 proposes a layered, hybrid control structure for automated task planning, and reactive execution, supported by a knowledge base for reasoning about the world state and entities. The scheduling formulation builds on the extended behavior tree model that merges task-level planning and execution. This allows for a high degree of modularity and a fast reaction to changes in the environment. The skill formulation based on pre-, hold- and post-conditions allows to organize robot programs and to compose diverse skills reaching from perception to low-level control and the incorporation of external tools. We relate SkiROS2 to the field and outline three example use cases that cover task planning, reasoning, multisensory input, integration in a manufacturing execution system and reinforcement learning.

1 Introduction

Modern intelligent robots require an increasing system complexity in order to perform the increasingly complex tasks demanded from them. Especially in view of greater autonomy, this complexity needs to be matched by the system architecture used to program and control the robots. With more and more integrated systems that coordinate different partial solutions, there is a need for interoperability and a common framework for communication, control and task planning.

In the industrial robotics field this can be seen in the Industry 4.0 movement that advocates such a transition, but several aspects of current practice present barriers to it. Many robot control systems currently rely heavily on "implicit" knowledge representation. Typically this is implemented by using if-else statements in the actual code. This often inhibits the growth of a control platform, as well as the interchangeability and interoperability between different tasks or robot systems, as this knowledge is hidden and often only known to the programmer herself/himself. Furthermore, vendor lock-in into the robot programming platform of a specific manufacturer is widespread.

These barriers have also been identified in the past and early platforms such as *ClaraTy* [Vol+01] or *LAAS* [Ben+09] provided first architectures. For knowledge integration frameworks, the system around the *Rosetta* ontology [MJ13; SM15] and *Knowrob* [TB09; TB13] created a strong foundation. However, the former is not publicly available and the latter targets the different needs of service robotics.

In this context, we introduce *SkiROS2*, a skill-based robot control framework for ROS. It utilizes knowledge representation in a resource description framework (RDF) graph that supports an open and explicit formulation of knowledge. The skill model is based on *pre-conditions* that are checked before a skill is executed, *hold-conditions* that also need to be satisfied while the skill is running, and *post-conditions* that are checked after the skill execution. SkiROS2 supports reasoning to infer skill parameters, and it allows the implementation and integration of custom reasoners, such as a spatial reasoner. Built-in task planning allows it to utilize robot capabilities to automatically construct a planning domain and problem description without manual input from a domain expert. Furthermore, it is capable of orchestrating multiple robot systems. SkiROS2 is open source, written in Python and based on BTs. It is the successor platform of SkiROS1 [Rov+17; RK15] that was written in C++ and did not use BTs.

In this paper, we discuss the requirements for our system architecture and assess

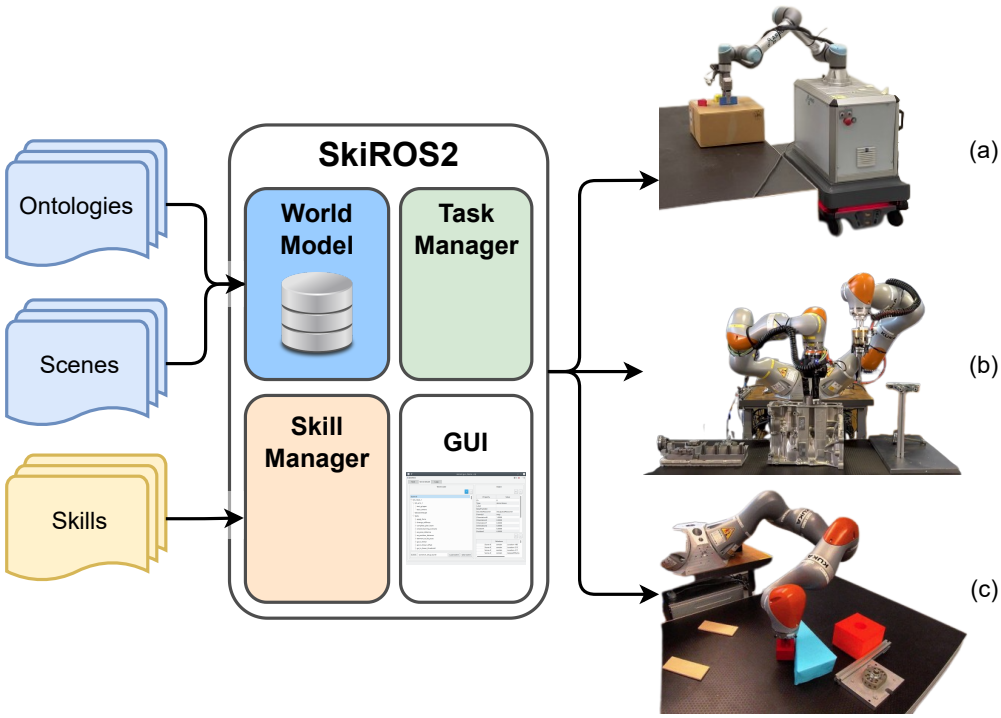


Figure 1.1: SkiROS2 can be used with different ontologies, scene descriptions and skills to solve a variety of tasks with different robot platforms. (a): Task-level planning with the mobile manipulation of objects. (b): Precise and sensitive piston insertion. (c): Learning to push by combining planning, reasoning and reinforcement learning.

the state-of-the-art in the field and we show how our solution SkiROS2 fulfills these requirements. We introduce its individual modules such as the skill manager, the task manager and the world model.

We think that this approach establishes a modern system architecture for intelligent autonomous robot systems that allows to design and organize robot skills for advanced tasks.

2 Background and Related Work

This section introduces related work on the axes of robot control strategies, cognitive approaches as well as knowledge integration. Finally, a brief comparison with other platforms is provided.

2.1 Robot Control Strategies

Three control strategies dominate the research community: deliberative, reactive, and hybrid control [KSB16].

Deliberative Systems

This architecture follows the ”sense, plan, act” paradigm. An observation of the world is translated into a symbolic representation [KSB16]. Reasoning is then utilized to plan a sequence of actions. While the actions are executed, new observations are not taken into account. This typically leads to slower reaction times, since the sequence of sensing and planning can often be time-consuming.

Reactive Systems

Reactive systems utilize a concurrently running finite state machines (FSM). These FSMs directly connect the input to the output. An important distinction is that they do not build up an internal representation. Their extension, behavior-based systems, also allows to assign priorities to behaviors and therefore inhibit output of low-priority ones.

Layered Hybrid Systems

In order to utilize the advantages of both aforementioned systems, reactivity and planning, layered hybrid systems were formulated. Typically, a higher planning level determines a mid- or long-term strategy, and a lower level is able to react directly to observations. Most often a synchronization layer is added to bridge between high-level reasoning and low-level control. Most modern robot control systems follow this approach.

2.2 Cognitive Systems

Besides having an architecture that allows tasks to be performed, another important question is what cognitive abilities are necessary and how these abilities are structured. Cognition can be defined as the process of a system to perceive the environment, act to pursue goals, anticipate the outcome of events, adapt the actions to changing circumstances and learn from experiences.

The *cognitivist approach* performs processing to obtain a symbolic knowledge representation [KSB16; VHF11]. This abstracted symbolic representation of the world allows to reason about it. The representation is typically designed and interpreted by humans, which means that it can be understood and complemented with knowledge from other sources.

In contrast to that, the *emergent paradigm* means an inductive organization from observations [VHF11]. Machine learning techniques and the automatic induction of ontologies are part of this field. It comes at the cost of being dependent on experiences.

Hybrid approaches aim to combine both approaches, explicit knowledge representation and learning from experiences [VHF11].

For our work, the cognitivist approach is the most relevant. However, a recent line of research extends it to a hybrid approach by incorporating learning from experiences into the system [May+22a; May+22c; AMK23].

2.3 Knowledge Integration

An important aspect of autonomous systems is the type of knowledge integration. It decides on the support for reasoning and inference, the possibility of integrating encyclopedic knowledge, and the expressiveness. The latter should be weighted against the efficiency of the reasoning algorithms [KSB16].

The available formalisms in the field of robotics include model logic, temporal logic, and predicate logic such as Prolog and Golog. Description logics are widely used to define ontologies. In particular the Web Ontology Language (OWL) that is based on the resource description framework (RDF) gained a lot of attention. A recent review and comparison of ontology-based approaches is presented in [Oli+19]. Projects such as *Knowrob* [TB09; TB13] created a strong foundation in the field of service robotics, while, for example, *Rosetta* [MJ13] is explicitly designed for the industrial domain.

2.4 Robot Control Platforms

We relate our work to other frameworks that are available as open-source software or provide clear guidelines for their task planning, task execution, and knowledge integration. Table 1.1 presents them along with criteria such as the application area, knowledge modeling and task planning.

Table 1.1: A comparison of different robot control platforms. The existence or lack of a feature is shown with ✓and ✗. The "-" indicates a lack of information.

Project Name/ Group	**Application**	**Knowledge Modeling**	**Task Planning**	**Scheduling**	**Middleware**	**Open Source**
SkiROS2	Industrial	OWL-DL	PDDL	eBT	ROS	✓
CoSTAR [Pax+17]	Industrial	-	Visual	BT	ROS	✓
GTax [Huc14]	Industrial	SysMl	PDDL	SysMl	-	✗
Balakirsky et al. [Bal+13]	Industrial	OWL-DL	PDDL	CRCL	ROS	✗
Stenmark et al. [MJ13; SM15]	Industrial	OWL-DL	PDDL	State charts	-	✗
Knowrob/CRAM/EASE [TB09; TB13]	Service	Prolog, OWL-DL	CPL	CPL	ROS	✓
CAST [Wya+10]	Service	Proxies	MAPL	MAPL	BALT	partially
LAAS [Ben+09]	General	-	lxTeT	Open-PRS	Bip/GenoM	partially
ClaraTy [Vol+01]	General	-	Corba	TDL	-	✓
SmartMDSD [Sta+16]	General	-	SmartTdl	SmartTdl	SmartSoft	✓
FlexBE [SKv16]	General	✗	(Synthesis Interface)	FSM	ROS	✓

Most of the related frameworks explicitly target the industrial domain [MJ13; SM15; Pax+17; Huc14] and [Bal+13]. Others, such as *Knowrob + CRAM* [TB09; TB13] or *CAST* [Wya+10] address the different needs of service robotics instead. Task-level planning is implemented by most of the frameworks. COSTAR [Pax+17] implements visual neural network-based planning [Pax+19] while [SKv16] offers an interface that can be used by planners. For the scheduling we see that only few implement reactive methods such as BTs. As a middleware, most frameworks use the robot operating system (ROS), which is advantageous for creating interoperability and using network effects. Many, but not all platforms are available as open-source software for other researchers to study, use, and improve. The comparison with existing frameworks shows that SkiROS2 has a unique combination attributes that enables it to be a modern autonomous robot control platform.

2.5 Behavior Trees

A behavior tree (BT) is a formalism for the representation and execution of procedures. BTs have emerged in the gaming industry, but are also becoming widespread in robotics [Iov+22]. A BT is a directec acyclic and rooted graph consisting of nodes and edges [CÖ17a]. The root node is used to periodically inject an enabling signal called *tick* that traverses through the tree according to the conditions, state of skills and their connectors called *processors*. These processors allow to link child nodes in different procedural ways. Examples are a sequence (logical AND) or a selector (logical OR). When the *tick* reaches a leaf node, it executes one cycle of the action or condition. Actions can modify the system configuration and return one of the three signals *success*, *running* or *failure*. Condition checks are atomic and can only return *success* or *failure*. To pass information between different nodes, a common approach is to use a set of shared variables on a *blackboard*. For a full formalization of BTs in the context of robotics, we refer to [CÖ17a].

The classical BT formulation is complemented by a formalism to define *extended behavior trees (eBTs)* in [RGK17]. In eBTs, scripted and planned procedures are merged into a unified representation, so that an eBT describes both the execution and its effects on the world state. This is achieved by combining the flexibility of BTs with hierarchical task network (HTN) planning. In contrast to classical BT, the pre- and post-condition nodes are embedded into the eBT to use them for task-level planning. To achieve real modularity, eBT allows procedural abstraction by allowing different implementations for the same type of action. Additional pre-conditions then allow to choose the right implementation to use

at run-time. For example, select a different opening strategy depending if you need to open a manual door or an automatic one that opens with a button. Furthermore, the eBT formalism allows to optimize the execution of the planned sequence [RGK17].

3 Design Considerations

On the basis of common applications in the field and our own experience, we identified the relevant requirements and design considerations. This section presents and discusses them in the context of robot control systems.

3.1 Control Strategy

Out of the main branches of robot control structures introduced in Section 2.1, the layered hybrid approach is the most applicable to the target domain: longer-term task goals require planning and reasoning while the execution should be able to react quickly to observations from the real world. Therefore, SkiROS2 uses a deliberate planning level and a lower-level implementation with BTs [CÖ17a].

3.2 Multi-Robot Orchestration

Challenging modern Industry 4.0 tasks are rarely content with a single robot. In addition, because of the importance of collaboration with humans, a robot control platform needs to be able to orchestrate several actuators simultaneously. In a multi-robot setup it is important that all actuators maintain a coherent world state to prevent failures and synchronization problems. SkiROS2 allows to start an arbitrary number skill managers for different robot systems that can share a single world model (WM), thus being able to simultaneously orchestrate a fleet of robots that have a common understanding of the world.

3.3 Knowledge Representation

To avoid an implicit representation of knowledge, for example in the form of "if, else" statements in code, it is important to have means to *explicitly* formulate and organize knowledge. Autonomous robot systems can also benefit from knowledge integration for abstract reasoning and especially in the industrial robotics domain, structured knowledge is often available. There are many logic

formalisms and the choice needs to balance *expressivity* and *efficiency*. Furthermore, we find the usability and availability of ontologies, the set of concepts and their relation in a target domain, to be important. Therefore, SkiROS2 uses the OWL and a set of established ontologies such as the *Core Ontology for Robotics and Automation (CORA)*.

3.4 Manufacturing Execution Integration

A platform for industrial robot skill execution needs to be able to interact with higher-level systems such as a manufacturing execution system (MES). Although the manual or automated start of individual skills is important, the concept of a skill-based platform really excels when higher-level goals can be sent to robot control systems [Kru+16]. A robot control system then needs to plan and execute autonomously. Furthermore, the execution state and modification of a shared understanding of the world such as a WM or a digital twin (DT) must be reported back [Kru+16].

3.5 Stakeholders

Robot control systems have different stakeholders that need to be addressed. Besides special needs for a vertical integration described in the previous section, users also have different expectations and needs. Most systems differentiate between developers and end users. Developers are assumed to have an in-depth understanding of the matter and need tools to support their processes. With a robot control platform, it can be assumed that developers design and implement skills. They can also understand and form ontologies. On the other hand, end users need lower entry hurdles. In the robotics context, this usually means addressing them with a graphical user interface (GUI) that abstracts away the underlying processes.

3.6 Middleware

Finally, the middleware as a communication system and application programming interface (API) are an important choice for every larger software project. The robotics domain with its different actuators, sensors and software solutions is particularly diverse and support for a wide variety of robotic systems can easily surpass the capabilities of companies and research groups.

Although there are other robotic middleware systems such as *OROCOS* [Bru01], the robot operating system (ROS) [Qui+09] became a commonly used and increasingly popular middleware solution for robotic systems and their programs. The ROS libraries are a good basis for integrated robot systems since they provide a standard communication and programming interface. Therefore, based on ROS, a control system such as SkiROS2 gets immediate access to a wide range of tools, drivers and trained users.

4 Architecture of SkiROS2

This section outlines the architecture of the system shown in Fig. 1.2. The *skill manager* and the *world model (WM)* form the core. One or more task managers can be used to accept high-level goals. During the prototyping phase, a GUI supports the users.

4.1 Skill Model

In this section we introduce the SkiROS2 skill model with its components and types of skills. We define a skill as a parametric procedure that changes the world from some initial state to some new state [Bøg+12]. This definition deliberately leaves room for a wide range of skills from deep learning-based object localization to lower-level motor control. The skill execution flow is shown in Fig. 1.3. In SkiROS2 skills can have different complexities: we refer here to primitive (atomic) skills and compound skills. The latter consist of any amount of primitive or compound skills.

Skill Description: The skill description defines the actions of a skill on a semantic level. A skill description includes its zero or more parameters and zero or more pre-, hold- and post-conditions. An example is shown in Lst. 1.1. Both primitive and compound skills always implemented exactly one skill description. However, an implementation is allowed to complement or further specify a skill description. This paradigm is useful to have multiple skill implementations for different hardware or specific scenarios. As an example for this, we can take a gripper actuation skill. A simple general skill description can be universal and have a boolean parameter "OpeningState" as well as a "Gripper" parameter that refers to a concept in the WM such as `Element("rparts:GripperEffector")`. Different gripper hardware needs different implementations of this description. To enable this, a skill implementation for a specific gripper can then modify the skill description of the parameter "Gripper" to handle only a single type of

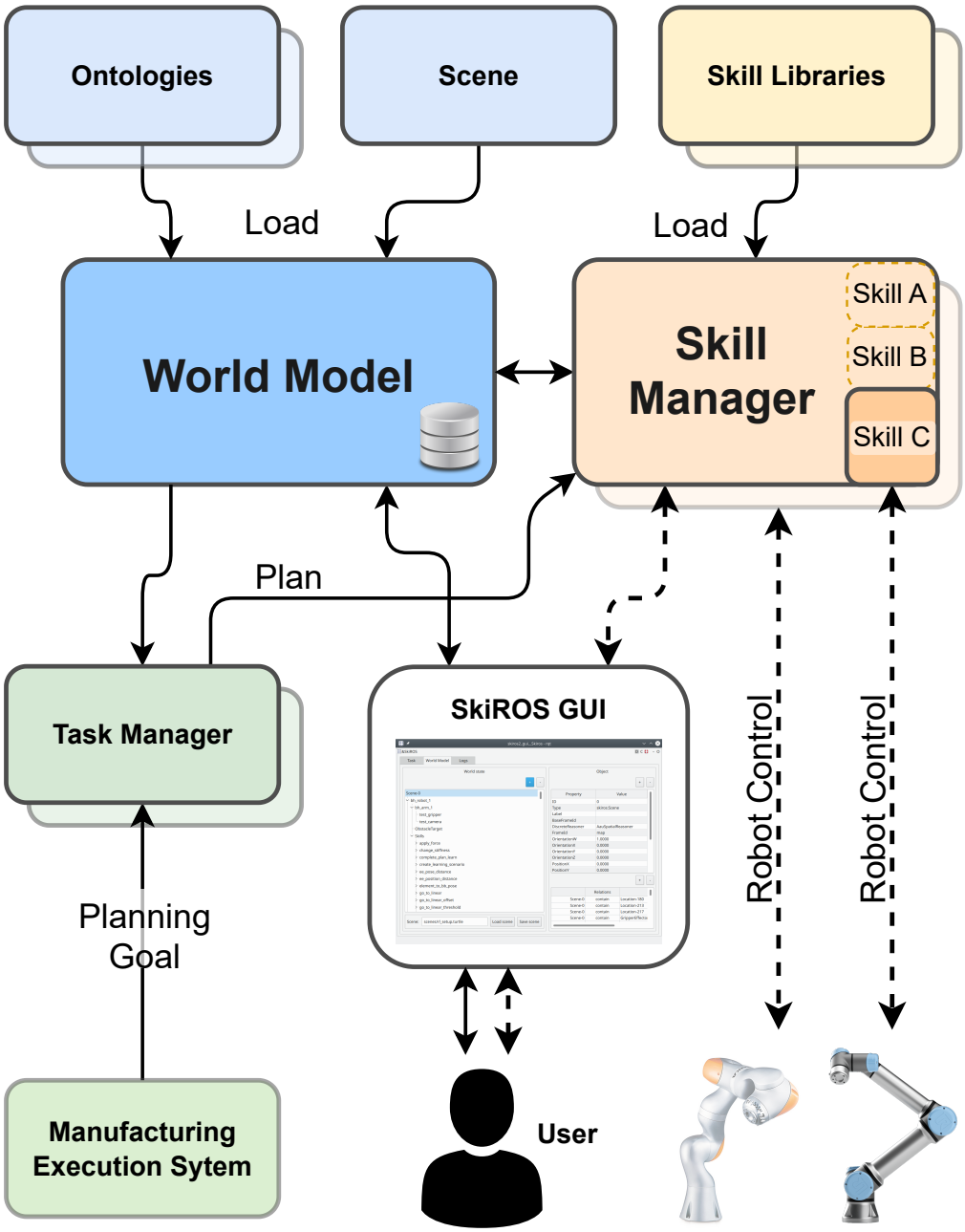


Figure 1.2: An outline of the SkiROS2 architecture. The world model stores the knowledge about the relations, environment and the skills. The skill manager loads and executes the skills. Dashed lines show control flows and solid lines information flows. Shaded blocks indicate possible multiple instances.

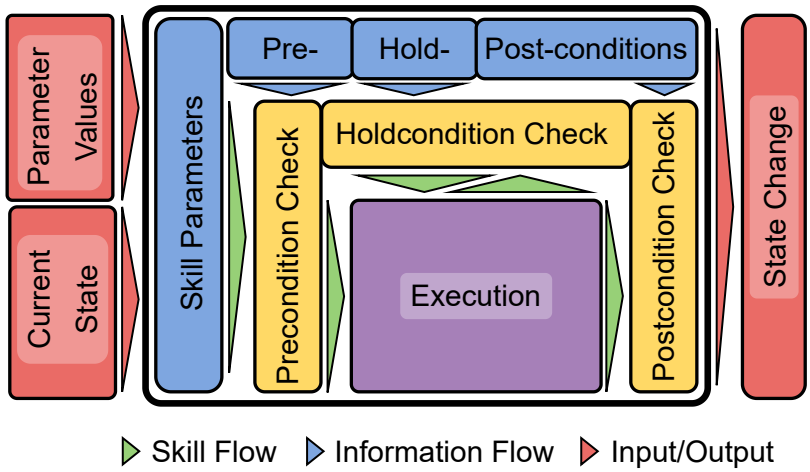


Figure 1.3: The conceptual model of a skill in SkiROS2. Pre-, and hold-conditions ensure that the skill is only executed in the correct world state. Post-conditions check if the desired changes have been achieved.

gripper that would be a subtype of the concept "rparts:GripperEffector", such as "scalable:RobotiqGripper". When executing a compound skill that utilizes this gripper actuation skill description, SkiROS2 will automatically select the matching implementation.

Skill Parameters: The skill parameters define the input and output of a skill. Furthermore, they can also be used for reasoning together with pre- and post-conditions. Parameters come in three flavors: 1) required, 2) optional, and 3) inferred. In contrast to the optional parameters, the required parameters must be set to execute a skill. The inferred parameters do not need to be set, but can automatically be reasoned about with the pre- and post-conditions. As an example, the *pick* skill in Lst. 1.1 has a required parameter *Arm*, could have an optional parameter *arm maximum velocity* and has an inferred parameter *Gripper*. If the parameter *Arm* is known, we can utilize the relations in the WM to automatically know which gripper is attached to it. The optional parameter for the maximum arm velocity can be used to overwrite a default value.

Skill parameters have either a fundamental data type such as `string`, `int`, `float`, `bool`, `list`, `dict` or are WM element in the form `Element("schema")`. The *schema* refers to a concept that is defined in the ontologies, and during execution it is grounded to a specific instance of that concept. This way its associated attributes and relations can be utilized by the skills. In the *pick* example above, the parameters *Gripper* and *Arm* would be WM element types `Element("rparts:GripperEffector")` and `Element("scalable:Arm")`.

Pre-, Hold- and Post-conditions: Preconditions describe the necessary world state under which a skill is to be executed. Furthermore, they can be used to infer parameters of a skill such as the parameters *Container* and *Arm* in the example in Lst. 1.1. Preconditions are complemented by *hold-conditions* that need to hold during the entire execution and are checked at every tick. Finally, the post-conditions specify the expected effects of a skill if the execution succeeded. Therefore, they are checked after the execution of the nodes has succeeded, and the skill will fail if the post-conditions do not hold.

All types of conditions have one of four types: 1) verification of relations in the WM such as "Arm has a Gripper"; 2) check the existence of an element property, e.g. the gripper has an attribute that specifies the finger length; 3) compare the value of a property, for example having gripper finger longer than 0.1 m; or finally 4) check for abstract relations in the ontology to verify that two elements are allowed to have a specified relation according to the ontology. In contrast to SkiROS1 [Rov+17], primitives can also have pre-, post- and hold-conditions, which increases robustness and allows to infer parameters.

Primitive Skills: Primitive skills, or short primitives, are the atomic actions in the SkiROS2 skill model. They typically implement behaviors that actively change the world, such as opening a gripper or moving a robot arm. Lst. 1.2 shows the primitive code skeleton that offers Python functions for the skill initialization, startup, execution, preemption and cleanup . Primitives always have a skill description such as the one in Lst. 1.1 that defines them on a semantic level and specifies the input and output parameters. When a skill is running, an `execute` function of the skill is called whenever the primitive is ticked as part of a BT and it must return one of the three signals *success*, *running* or *failure*.

Compound Skills: As an important extension to SkiROS1, this version formulates compound skills. They allow one to connect arbitrary amounts of other compound and primitive skills to define more complex behaviors. An example is given in Lst. 1.3. Compound skills are modeled with BTs and their processors. This enables developers to fully utilize the modularity and reactiveness of the BTs by reusing existing skills and to formulate reactions to changes in the world. SkiROS2 formulates the following processors:

- *Serial*: Sequential execution (logical AND)
- *SerialStar*: Sequential execution with memory
- *Selector*: Fallback (logical OR)
- *SelectorStar*: Fallback with memory

```
class Pick(SkillDescription):
 def createDescription(self):
  # ======= Parameters =========
  self.addParam("Container", Element("skiros:Location"),
      ParamTypes.Inferred)
  self.addParam("Gripper", Element("rparts:GripperEffector"),
      ParamTypes.Inferred)
  self.addParam("Object", Element("skiros:Product"),
      ParamTypes.Required)
  self.addParam("Arm", Element("rparts:ArmDevice"),
      ParamTypes.Required)
  # ======= PreConditions =========
  self.addPreCondition(self.getPropCond("EmptyHanded",
      "skiros:ContainerState", "Gripper", "=", "Empty", True))
  self.addPreCondition(self.getRelationCond("ObjectInContainer",
      "skiros:contain", "Container", "Object", True))
  # ======= HoldConditions =========
  self.addHoldCondition(self.getRelationCond("RobotAtLocation",
      "skiros:at", "Robot", "Container", True))
  # ======= PostConditions =========
  self.addPostCondition(self.getPropCond("EmptyHanded",
      "skiros:ContainerState", "Gripper", "=", "Empty", False))
  self.addPostCondition(self.getRelationCond("Holding",
      "skiros:contain", "Gripper", "Object", True))
```

Listing 1.1: The skill description of a pick skill. It defines the parameters, pre-, hold- and post-conditions and can be automatically used for task-level planning.

- *Parallel First Stop*: Process children in parallel until one succeeds or fails
- *Parallel First Fail*: Parallel run children until all succeed

Furthermore, there are decorators such as *NoSuccess* or *NoFail* that modify the return signals.

When including other skills in a compound skill, it is possible to explicitly set the parameters of the included skill if they are of any of the fundamental types. It is also possible to remap the parameters that are WM elements. For example, the ”pick” skill in Lst. 1.1 has the parameter ”Container”. In the example in Lst. 1.3, the primitive to update the world model (”wm_move_object”) expects an input parameter ”StartLocation”. It is possible to map the references for these parameters, so they can be used in both skills. This can also be used to resolve naming conflicts.

```
1  class my_primitive(PrimitiveBase):
2    def createDescription(self):
3      self.setDescription(MyPrimitive(), self.__class__.__name__)
4    def onInit(self):
5      """ Called once when loading skills - it is not loaded on
           False """
6      return True
7    def onPreempt(self):
8      """ Called when skill is requested to stop. """
9      return self.success("Preempted")
10   def onStart(self):
11     """ Called just before 1st execute """
12     return True
13   def execute(self):
14     """ Main execution function. Returns: self.fail, self.step or
           self.success """
15     return self.success("Executed")
16   def onEnd(self):
17     """ Called just after last execute OR preemption """
18     return True
```

Listing 1.2: The code skeleton for a primitive skill. The functions allow to define the initialization, execution and reaction to a preemption. In line 3 it is stated which skill description (such as the one in Lst. 1.1) the primitive implements.

```
1  class pick_fake(SkillBase):
2    def createDescription(self):
3      self.setDescription(Pick(), self.__class__.__name__)
4
5    def expand(self, skill):
6      """ In this function the BT is defined """
7      skill.setProcessor(SerialStar())
8      skill(
9        self.skill("Wait", "wait", specify={"Duration": 1.0}),
10       self.skill("WmMoveObject", "wm_move_object",
11              remap={"StartLocation": "Container", "TargetLocation":
                    "Gripper"}),
12     )
```

Listing 1.3: An example for a mockup pick skill that implements the example description in Lst 1.1. In line 7 the processor of the BT is set. In line 9 the parameter "Duration" of the skill "wait" is set to a concrete value and in line 10 parameter remappings are done.

4.2 Skill Manager

Every robot has its own skill manager. The skill manager loads a set of specified skills from the skill libraries. It initializes them and complements the WM with semantic information such as the representation of the skill parameters and the skill conditions. After the initialization phase, the skill manager offers services to start, stop and debug skills, as well as monitor their execution. Since the skill manager is also executing the skills and accepts skill plans from the task manager, it is a core component of the platform. Whenever a skill or skill sequence is started, the skill manager creates a new task with a unique ID. Within a task, the parameters of the executed skills are shared on a blackboard. This allows to exchange information such as calculated poses or even camera images between the skills.

4.3 World Model

The world model server stores the ontologies and the instances. As such, it loads the relevant ontologies that specify the known concepts (schemas) at the start. Furthermore, it can load a specific scene that contains the instances, also called vocabulary of the concepts that are specified in the ontologies. A typical scene includes the semantic description of the specific robot system, objects in the environment or known locations. The scene is complemented by the skill manager with information about the available skills.

The information in the WM can be utilized for reasoning and for the parameterization of skills. It also exposes an API that is used inside of the skills to read from and write to the WM. Furthermore, custom reasoners can be plugged in. As an example for such reasoners, *skiros2_std_lib* implements a spatial reasoner that can transform coordinates and calculate Allen intervals algebra [Rov+18].

4.4 Task Manager

In Section 3 we outlined the importance of the vertical integration into systems that provide tasks such as MES. SkiROS2 addresses this by providing a sophisticated task planning integration that builds on the planning domain definition language (PDDL) [Aer+98]. It utilizes the *temporal fast downward planner*[1] [EMR12] to guarantee the finding of an optimal skill sequence. In contrast to other solution, SkiROS2 has a fully automated generation of the

[1] `http://gki.informatik.uni-freiburg.de/tools/tfd/`

problem and planning domain based on the current state of the WM. This includes the vocabulary of the scenes like the available robot as well as all the skills with their pre- and post-conditions. This generation drastically simplifies usability, as it is not necessary to maintain a separate planning domain.

The resulting plan can be sent to the skill manager and can become an executed task. It is then automatically converted into an executable eBT [RGK17] and the skill manager will expand the branches of the eBT automatically.

4.5 ROS Integration and User Interface

SkiROS2 is well integrated into robot operating system (ROS) [Qui+09]. While SkiROS2 utilizes ROS as a middleware itself, it also features many convenient integrations that ease the use and implementation of skills. The WM and its elements are fully integrated with the ROS transformation system *tf*. As such, WM elements can be linked to frames that exist outside SkiROS2 and elements can be published as coordinate frames. On the skill level the *skiros2_std_lib* provides a skill primitive that can easily turn any ROS *action* into a skill.

The SkiROS2 graphical user interface (GUI) offers a drastically lowered entry hurdle for non-experts. Like many other ROS GUIs, it is written with ROS' *rqt* and can be used alongside them. In the different tabs, users get an overview of the loaded skills. Skill parameters can be changed and skills can be started and stopped. Additionally, the WM view in the GUI provides full access to the content of the WM. The vocabulary including all the properties and relations can be inspected as well as fully modified. New relations and properties can be added through easy-to-use dialogues. An integration with the visualization tool *RViz* allows to modify the pose of a WM element.

5 Use-Cases

In this section, we discuss a selection of different use cases in which *SkiROS2* is used as a robot control system for partially-structured tasks.

5.1 Pick-place with a Mobile Robot

In the pick-and-place scenario, a mobile robot (e.g. *heron* in Fig. 1.1a) is tasked to place an item at a new location that can either be at the same or at another

workstation. In this scenario it is required that the robot can drive between workstations, actuate the arm as well as the gripper, and use the camera to compute the exact pose of the object. The skills can switch the control mode of the arm between compliant control for contact-rich tasks and position control for planned trajectories. The used skills have full integration with planning and it is sufficient to specify a goal in PDDL such as `skiros:contain skiros:locationB skiros:objectA`. As described in Section 4.4, triggering the planning automatically generates the domain description and problem instance. A simplified example skill sequence is:

```
drive(workstationA)
pick(objectA)
drive(workstationB)
place(locationB)
```

Lst. 1.1 shows a simplified description of a *pick* skill. The specified preconditions can be used to infer parameters at planning time, but also at run-time. If this example skill is called manually with *SkiROS2* GUI, it is sufficient to specify the object to pick and the arm to use, and the rest of the parameters are automatically inferred using the WM.

5.2 Dual-arm Piston Insertion

The task in this use case is to perform a tight insertion of a piston into a real engine block. This insertion requires dual-arm manipulation as well as specialized tools since an assistive ring must be held for insertion. The skills used in this task heavily utilize the WM, but are at the same time able to learn parts of the procedure, such as lifting poses, from kinesthetic teaching. Additionally, the pose of objects, such as the engine block, can be estimated using the camera mounted on the robot arm [GRK19]. The motion skills implement a combination of a motion generator with BTs [Rov+18] and have extensive checks on the robot state such as applied forces and torques. The extensive use of those conditions allows aborting the execution if they are exceeded.

While the skills for this use-case do not have the necessary pre- and post-conditions to use them for task planning, the concept of these conditions is used to regularly check if the system is in the desired state. Furthermore, these skills are written so that they are preemptable [Wut+21]. As an example for a preemption procedure we can consider a state in which an object is in the

gripper. Aborting the current action and switching to a different task requires to place the object at an appropriate location first.

5.3 Reinforcement Learning of Industrial Robot Tasks

In a third use-case, *SkiROS2* is used to learn industrial tasks with reinforcement learning (RL) [May+22a; May+22c; May+22b; Ahm+22; AMK23]. Such tasks include pushing an object on a table (shown in Fig. 1.1c) or learning a peg-insertion strategy. The skills used in these scenarios are written by domain experts and can be utilized by the task planner. However, there is no automated reasoning module for some of the skills that can be utilized to fully parameterize them for the task at hand. In this learning approach, the skill parameters in a skill sequence that cannot be parameterized are automatically identified. Then additional information about these learnable parameters, such as their type and upper and lower limits, is obtained from the WM and used to automatically create a learning scenario description. In the next step, the learning procedure starts either on a real robot system or in simulation, and the RL framework calculates the rewards depending on the performance. When leveraging on learning in simulation, up to thousands of executions can be run to identify a robust and well-performing set of parameters. Finally, the best configurations are presented to the operator and a final set can be selected for production [May+22a; May+22c]. If the operator has an educated guess for good parameter values or experiences from similar tasks, this learning approach can also incorporate this information as priors to accelerate learning and increase safety [May+22b]. With [AMK23], an extension of [May+22a; May+22c] is proposed to learn behaviors for a variety of task variations and allow zero-shot execution even for unseen task configurations.

6 Conclusions

Modern autonomous robotic systems require solutions for the fundamental requirements: knowledge organization, control structuring, multi-robot orchestration and integration with external systems. We outlined the requirements and introduced the skill-based robot control platform SkiROS2 and its core components. Its unique combination of features makes it suitable for intelligent autonomous robot control. To show this, we outlined three example use cases that cover a wide variety of relevant challenges in robotics: Integration of diverse robotic systems, task-level planning and integration of vision and learning from

users. As part of these SkiROS2 has also been shown to allow the combination of deductive methods such as reasoning with inductive methods such as learning to improve the performance of executions by interacting with the environment.

The *SkiROS2* platform is fully open source and comes with documentation, tutorials and examples. Currently it integrates with ROS 1, but a ROS 2 version and extensions such as *EzSkiROS* [Riz+23] are in development. The code is available at: `https://github.com/RVMI/skiros2`

Acknowledgement

The authors thank Bjarne Grossmann, David Wuthier, Faseeh Ahmad, Simon Kristoffersson Lind and Momina Rizwan for their contributions.

This work was partially supported by the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation.

References

[Aer+98] Constructions Aeronautiques et al. “Pddl the Planning Domain Definition Language”. In: *Technical Report, Tech. Rep.* (1998).

[AMK23] Faseeh Ahmad, Matthias Mayr and Volker Krueger. “Learning to Adapt the Parameters of Behavior Trees and Motion Generators (BTMGs) to Task Variations”. In: *2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2023.

[Ahm+22] Faseeh Ahmad et al. “Generalizing Behavior Trees and Motion-Generator (BTMG) Policy Representation for Robotic Tasks over Scenario Parameters”. In: *2022 IJCAI Planning and Reinforcement Learning Workshop* (2022).

[Bal+13] Stephen Balakirsky et al. “Knowledge Driven Robotics for Kitting Applications”. In: *Robotics and Autonomous Systems*. Ubiquitous Robotics 61.11 (Nov. 2013), pp. 1205–1214.

[Ben+09] Saddek Bensalem et al. “Designing Autonomous Robots”. In: *IEEE Robotics & Automation Magazine* 16.1 (Mar. 2009), pp. 67–77.

[Bøg+12] Simon Bøgh et al. “Does Your Robot Have Skills?” In: *Proceedings of the 43rd International Symposium on Robotics*. VDE Verlag GmbH, 2012.

[Bru01] H. Bruyninckx. “Open Robot Control Software: The OROCOS Project”. In: *Proceedings 2001 ICRA. IEEE International Conference on Robotics and Automation (Cat. No.01CH37164)*. Vol. 3. May 2001, 2523–2528 vol.3.

[CÖ17a] Michele Colledanchise and Petter Ögren. *Behavior Trees in Robotics and AI: An Introduction*. Chapman \& Hall/CRC Press, Aug. 2017.

[EMR12] Patrick Eyerich, Robert Mattmüller and Gabriele Röger. “Using the Context-Enhanced Additive Heuristic for Temporal and Numeric Planning”. In: *Towards Service Robots for Everyday Environments: Recent Advances in Designing Service Robots for Complex Tasks in Everyday Environments*. Ed. by Erwin Prassler et al. Springer Tracts in Advanced Robotics. Berlin, Heidelberg: Springer, 2012, pp. 49–64.

[GRK19] Bjarne Grossmann, Francesco Rovida and Volker Kruger. “Continuous Close-Range 3D Object Pose Estimation”. In: *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Nov. 2019, pp. 2861–2867.

[Huc14] Jacob O. Huckaby. “Knowledge Transfer in Robot Manipulation Tasks”. PhD thesis. Georgia Institute of Technology, Apr. 2014.

[Iov+22] Matteo Iovino et al. “A Survey of Behavior Trees in Robotics and AI”. In: *Robotics and Autonomous Systems* 154 (Aug. 2022), p. 104096.

[KSB16] David Kortenkamp, Reid Simmons and Davide Brugali. “Robotic Systems Architectures and Programming”. In: *Springer Handbook of Robotics*. Ed. by Bruno Siciliano and Oussama Khatib. Springer Handbooks. Cham: Springer International Publishing, 2016, pp. 283–306.

[Kru+16] V. Krueger et al. “A Vertical and Cyber–Physical Integration of Cognitive Robots in Manufacturing”. In: *Proceedings of the IEEE* 104.5 (May 2016), pp. 1114–1127.

[MJ13] Stenmark Maj and Malec Jacek. “Knowledge-Based Industrial Robotics”. In: *Frontiers in Artificial Intelligence and Applications* (2013), pp. 265–274.

[May+22a] Matthias Mayr et al. “Combining Planning, Reasoning and Reinforcement Learning to Solve Industrial Robot Tasks”. In: *IROS 2022 Workshop on Workshop on Trends and Advances in Machine Learning and Automated Reasoning for Intelligent Robots and Systems* (2022).

[May+22b] Matthias Mayr et al. “Learning Skill-Based Industrial Robot Tasks with User Priors”. In: *2022 IEEE 18th International Conference on Automation Science and Engineering (CASE)*. IEEE, 2022, pp. 1485–1492.

[May+22c] Matthias Mayr et al. “Skill-Based Multi-Objective Reinforcement Learning of Industrial Robot Tasks with Planning and Knowledge Integration”. In: *2022 IEEE International Conference on Robotics and Biomimetics (ROBIO)*. IEEE, 2022, pp. 1995–2002.

[Oli+19] Alberto Olivares-Alarcos et al. “A Review and Comparison of Ontology-Based Approaches to Robot Autonomy”. In: *The Knowledge Engineering Review* 34 (2019/ed).

[Pax+17] Chris Paxton et al. “CoSTAR: Instructing Collaborative Robots with Behavior Trees and Vision”. In: *2017 IEEE International Conference on Robotics and Automation (ICRA)*. May 2017, pp. 564–571.

[Pax+19] Chris Paxton et al. “Visual Robot Task Planning”. In: *2019 International Conference on Robotics and Automation (ICRA)*. May 2019, pp. 8832–8838.

[Qui+09] Morgan Quigley et al. “ROS: An Open-Source Robot Operating System”. In: *2009 IEEE International Conference on Robotics and Automation (ICRA) Workshop on Open Source Robotics*. Kobe, Japan, May 2009.

[Riz+23] Momina Rizwan et al. “EzSkiROS: A Case Study on Embedded Robotics DSLs to Catch Bugs Early”. In: *2023 IEEE/ACM 5th International Workshop on Robotics Software Engineering (RoSE)*. IEEE, 2023, pp. 61–68.

[RGK17] F. Rovida, B. Grossmann and V. Krüger. “Extended Behavior Trees for Quick Definition of Flexible Robotic Tasks”. In: *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Sept. 2017, pp. 6793–6800.

[Rov+18] F. Rovida et al. “Motion Generators Combined with Behavior Trees: A Novel Approach to Skill Modelling”. In: *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Oct. 2018, pp. 5964–5971.

[RK15] Francesco Rovida and Volker Krüger. “Design and Development of a Software Architecture for Autonomous Mobile Manipulators in Industrial Environments”. In: *2015 IEEE International Conference on Industrial Technology (ICIT)*. Mar. 2015, pp. 3288–3295.

[Rov+17] Francesco Rovida et al. “SkiROS—A Skill-Based Robot Control Platform on Top of ROS”. In: *Robot Operating System (ROS): The Complete Reference (Volume 2)*. Ed. by Anis Koubaa. Studies in Computational Intelligence. Cham: Springer International Publishing, 2017, pp. 121–160.

[SKv16] Philipp Schillinger, Stefan Kohlbrecher and Oskar von Stryk. “Human-Robot Collaborative High-Level Control with Application to Rescue Robotics”. In: *2016 IEEE International Conference on Robotics and Automation (ICRA)*. May 2016, pp. 2796–2802.

[Sta+16] Dennis Stampfer et al. “The SmartMDSD Toolchain: An Integrated MDSD Workflow and Integrated Development Environment (IDE) for Robotics Software”. In: (2016).

[SM15] Maj Stenmark and Jacek Malec. “Knowledge-Based Instruction of Manipulation Tasks for Industrial Robotics”. In: *Robotics and Computer-Integrated Manufacturing*. Special Issue on Knowledge Driven Robotics and Manufacturing 33 (June 2015), pp. 56–67.

[TB09] Moritz Tenorth and Michael Beetz. “KNOWROB — Knowledge Processing for Autonomous Personal Robots”. In: *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems*. Oct. 2009, pp. 4261–4266.

[TB13] Moritz Tenorth and Michael Beetz. “KnowRob: A Knowledge Processing Infrastructure for Cognition-Enabled Robots”. In: *The International Journal of Robotics Research* 32.5 (Apr. 2013), pp. 566–590.

[VHF11] David Vernon, Claes von Hofsten and Luciano Fadiga. *A Roadmap for Cognitive Development in Humanoid Robots*. Springer Science & Business Media, Dec. 2011.

[Vol+01] R. Volpe et al. “The CLARAty Architecture for Robotic Autonomy”. In: *2001 IEEE Aerospace Conference Proceedings (Cat. No.01TH8542)*. Vol. 1. Mar. 2001, 1/121–1/132 vol.1.

[Wut+21] D. Wuthier et al. “Productive Multitasking for Industrial Robots”. In: *2021 IEEE International Conference on Robotics and Automation (ICRA)*. May 2021, pp. 12654–12661.

[Wya+10] Jeremy L. Wyatt et al. “Self-Understanding and Self-Extension: A Systems and Representational Approach”. In: *IEEE Transactions on Autonomous Mental Development* 2.4 (Dec. 2010), pp. 282–303.

Paper II

Paper 2

Using Knowledge Representation and Task Planning for Robot-agnostic Skills on the Example of Contact-Rich Wiping Tasks

Matthias Mayr
Lund University
matthias.mayr@cs.lth.se

Faseeh Ahmad
Lund University
faseeh.ahmad@cs.lth.se

Alexander Duerr
Lund University
alexander.duerr@cs.lth.se

Volker Krueger
Lund University
volker.krueger@cs.lth.se

Abstract

The transition to agile manufacturing, Industry 4.0, and high-mix-low-volume tasks require robot programming solutions that are flexible. However, most deployed robot solutions are still statically programmed and use stiff position control, which limit their usefulness.
In this paper, we show how a single robot skill that utilizes knowledge representation, task planning, and automatic selection of skill implementations based on the input parameters can be executed in different contexts. We demonstrate how the skill-based control platform enables this with contact-rich wiping tasks on different robot systems. To achieve that in this case study, our approach needs to address different kinematics, gripper types, vendors, and fundamentally different control interfaces. We conducted the experiments with a mobile platform that has a *Universal Robots UR5e* 6 degree-of-freedom robot arm with position control and a 7 degree-of-freedom *KUKA iiwa* with torque control.

1 Introduction

The need for flexible and skill-based robot control systems in industrial robot applications is becoming increasingly important. With the rise of automation and the growing complexity of tasks, robots must be able to adapt to changing environments and perform a variety of tasks without lengthy configuration times. This requires a control system that can respond quickly and accurately to changes in the environment and the task at hand. However, most of the deployed systems are statically programmed and perform repetitive tasks. This introduces the need for a flexible and skill-based robot control system in industrial robot applications that provides flexibility and supports agile manufacturing.

First platforms for skill-based systems such as *ClaraTy* [Vol+01] or *LAAS* [Ben+09], paved the way. In the area of knowledge integration frameworks, the system built around the *Rosetta ontology* [MJ13; SM15] and *Knowrob* [TB09; TB13] created a solid foundation. The latter addresses the service domain, while the former is placed in the context of industrial robotics. To transfer skills from one robot to another is nontrivial if robots of different types and vendors are involved. For instance, [Top+18] demonstrated skill transfer for contact-free motions within a single family of robots that share the same vendor-supplied high-level interface *ABB RAPID*. In that paper, the simulated experiments were conducted with high-gain position control. This is, of course, still a standard practice within many tasks that have little variation and high certainty. However, the increasing number of contact-rich tasks that manipulators should be able to solve [Wut+21] indicates a clear need for flexible and compliant systems. Achieving compliant behaviors heavily depends on the hardware and vendor, and there are no standard interfaces available for this purpose. In this paper, we explore the question of how skills can be transferred between collaborative robots, such as the *Universal Robot* robots and the *KUKA iiwa* or the *Franka Emika Robot (Panda)*. To do this, we develop a suitable representation of the available robot hardware, a knowledge infrastructure to maintain this hardware knowledge, and a suitable compliant controller that works across different types of robot arms. To validate and verify our claims, we test our architecture on a whiteboard wiping task, as shown in Fig. 2.1, using a *UR5e* and a *KUKA iiwa*. The whiteboard wiping task is an example of a typical contact-rich task that requires a compliant controller. For this, the *UR5e* has a force-torque sensor on the wrist, while robots such as the *iiwa* and the *Panda* have torque sensors in their joints.

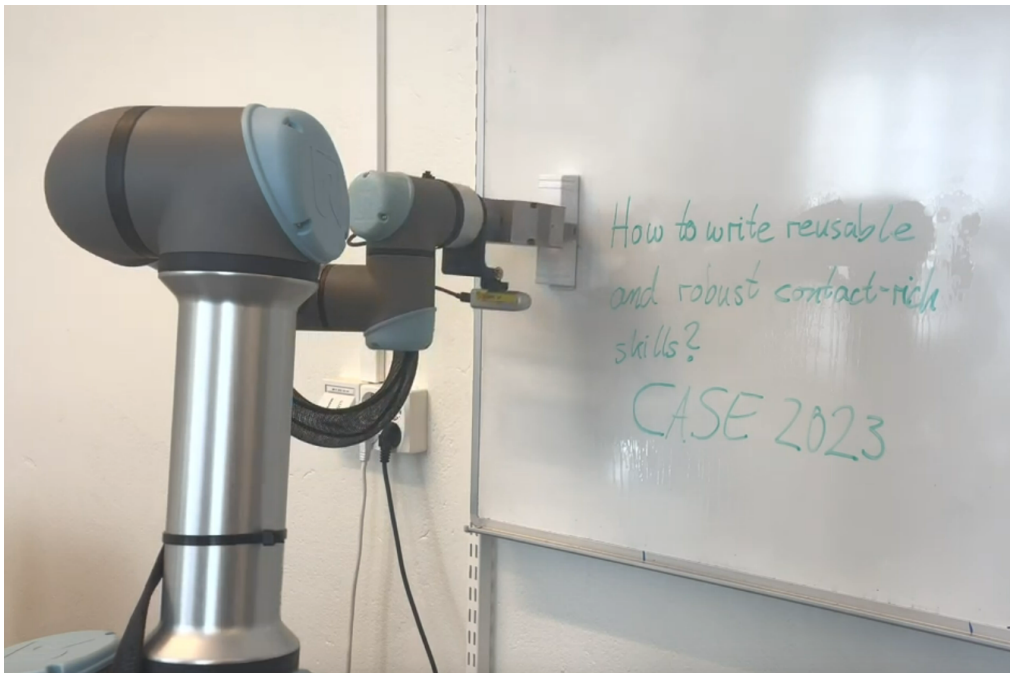


Figure 2.1: With the proposed abstractions and knowledge representation, several different robot systems can execute the same skills while still using their respective hardware such as grippers and controllers.

In summary, our paper makes the following contributions:

1. We present a complete architecture that represents, plans, and executes skills such that they can be transfered between robots.
2. We demonstrate these ideas based on a whiteboard wiping task using a *UR5e* and a *KUKA iiwa* robot.
3. All essential building blocks, including the impedance controller implementations are available as open-source software.
4. The code is written in a robot-agnostic way, making the results and insights scalable to diverse collaborative robotic hardware and tasks.

2 Related Work

2.1 Robot Skill Systems

We can classify skill-based architectures based on their target domain. Projects like *CAST* [Wya+10] and *KnowRob* [TB09; TB13] are for service robotics and general platforms are *ClaraTy* [Vol+01], *LAAS* [Ben+09] and *SmartMDSD* [Sta+16]. However, platforms that address the specific needs of industrial robotics are the most relevant for this work. *CoStar* [Pax+17] has shown to integrate multiple robot systems and provides a GUI for the design and the monitoring of skills. However, it lacks explicit knowledge representation and the possibility to perform task planning. The use of ontologies to describe skills in robotic applications has been explored in various areas, including medical and surgical robotics [Bru+19; Gib+18], perception ontologies [ABR18] for robots, and more [Man+21]. Among these, [MJ13; SM15; Top+18] present a knowledge-based approach for programming synchronized motions between robotic systems and human-robot interactions that is particularly relevant for our work.

In [Top+18], the authors demonstrate the effectiveness of their approach through experiments that involve transformations of dual arm programs and the transfer of skills between robot systems with different kinematics. Their approach uses finite state machines to implement skill behavior and generates *ABB RAPID* code that is specific to *ABB* robot controllers. Our work differs in that we use behavior trees for skill behavior implementation and our implementation is not limited to a particular robot controller. In addition, [Top+18] was validated only in simulation, and the tasks were not contact-rich.

2.2 Robot Motions

In the literature, Movement Primitives (MPs) have been a reliable tool for generating robot motions. Dynamic Movement Primitives (DMPs) [INS01; INS02] and Probabilistic Movement Primitives (ProMPs) [Par+13] have been successfully used to generate arm motions. Both DMPs and a motion representation based on behavior trees and motion generators (BTMG) that will be introduced in Section 4.1 use attractor landscapes to reach a target location. DMPs are non-linear dynamical systems that allow trajectory control, and, like our BTMGs, they have a non-linear forcing term that enables the modification of the original trajectory. However, in contrast to DMPs, the parameters of BTMGs

can be specified explicitly, giving us a better understanding of the robot motion and allowing for a better intregration with knowledge representation.

Wiping Applications: In [Lei+15; Lei+16; Lei+19] the authors discuss research on robotic manipulation of wiping tasks using artificial intelligence reasoning methods. [Lei+15] proposes a classification of compliant manipulation tasks based on symbolic effects, subcategorizes wiping tasks, and demonstrates how to concretize actions based on the example of shards sweeping with a broom. [Lei+16] investigates the reasoning and action execution problems involved in the execution of wiping tasks and proposes a high-level abstraction representation of wiping tasks to develop generalized action execution mechanisms. [Lei+19] combines reasoning methods and compliant robotic manipulation to solve wiping tasks by introducing a qualitative particle distribution model, an approach to generate whole-body wiping motions based on effect-oriented policies, an approach to assess wiping motion quality, and the semantic interpretation of contact situations. Rather than achieving the best wiping skill, the focus of our paper is on presenting a complete architecture that represents, plans, and executes skills, and then using wiping as a challenging example to demonstrate these ideas.

3 Transferable Robot Skills

In this section we introduce the prerequisites that enable to transfer skills between different robot systems. In addition to the skill platform *SkiROS2* and its skill model, the necessary aspects of knowledge presentation and task planning are introduced. The control implementations and motion generation are introduced in Section 4.

3.1 SkiROS2

We utilize the skill-based robot control platform *SkiROS2* [RGK17] to implement transferable robot skills. It is used in several research projects for motion generation [Rov+18], robot coordination [Wut+21] and task-level planning with RL [May+22a; May+22c; May+22b; AMK23]. The architecture is shown in Fig. 2.2 and the main components are the skill manager and the world model. The WM contains the knowledge about the world and represents it in a RDF database. Ontologies such as the *Suggested Upper Merged Ontology (SUMO)* describe the available concepts such as objects or skills and their relations to each other. In addition to that, a *scene* contains the concrete instances of the

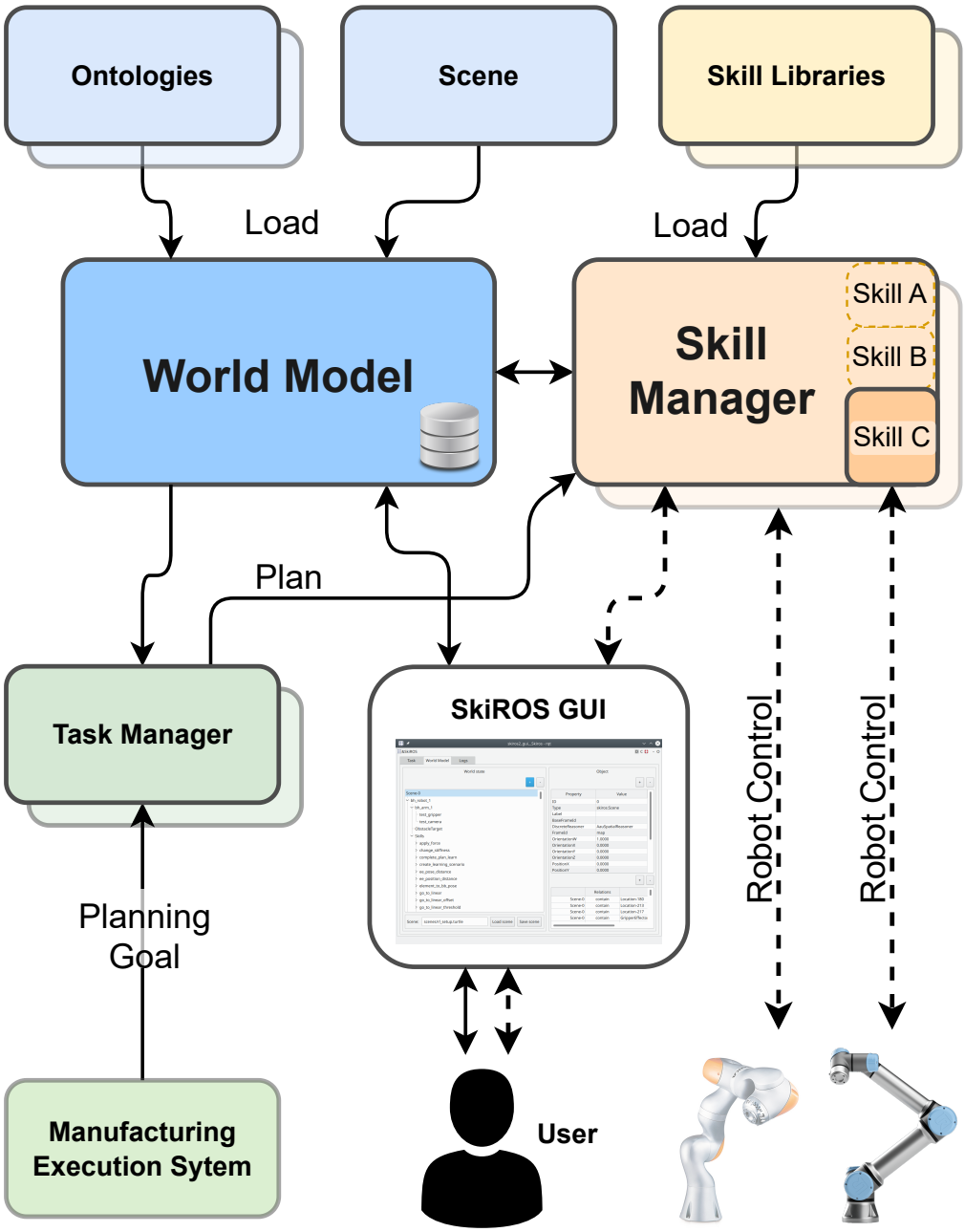


Figure 2.2: An outline of the SkiROS2 architecture. The world model stores the knowledge about the relations, environment and the skills. The skill manager loads and executes the skills. Dashed lines show control flows and solid lines show information flows. Shaded blocks indicate possible multiple instances.

concepts that are defined in the ontologies. Such instances can be the present robot systems, workstations or objects that are currently known.

The skill manager is responsible for loading a set of specified skills from the skill libraries. Each skill that is successfully loaded gets a semantic description in the WM. This includes the parameters and pre-, hold- and post-conditions. In addition to that, the skill manager provides services to start, pause and stop skills and to manage their execution.

The task manager accepts the planning goals in PDDL [Aer+98; RGK17]. Such goals can come from a user or an external entity such as a MES. An example of such a goal is to place an object at a specific location: `(skiros:contain skiros:Location-1 skiros:Product-1)`. The task manager automatically creates a PDDL planning domain from the knowledge in the WM. The currently available skills are added with their preconditions and effects, and the relevant instances in the scene are automatically added to a planning problem. Then a PDDL planner is called to find a sequence of skills that is guaranteed to be optimal [EMR12]. Finally, the plan is converted back into a BT, automatically expanded with the relevant skill implementations and grounded with the instances in the WM [RGK17].

3.2 Skill Model

We define a skill as a parametric procedure that changes the world from some initial state to some new state [Bøg+12]. It is shown in Fig. 2.3. A skill consists of two main components: a skill description and the skill implementation. A skill description contains the skill parameters and the necessary pre-, hold- and post-conditions. These define what prerequisites need to exist to execute a skill and state the expected effects of running it. In the task manager, these conditions can be used for automatic task planning. Furthermore, the parameters state the input and output of a skill. There are three different types of parameters: 1) required, 2) optional and 3) inferred. An example of a required parameter of a ”pick” skill is to specify the robot arm, while an optional parameter could be to slow down the execution for testing. The inferred parameters are reasoned about when starting the skill. An example is a ”gripper” parameter for a pick skill that already has the arm as a required parameter. With a precondition we can define a relation between both of them and we can use the knowledge in the WM to specify the inferred parameter automatically [RGK17].

In addition to the skill description that describes the semantic actions of a skill, the skill implementation is a concrete version for the execution. Each skill im-

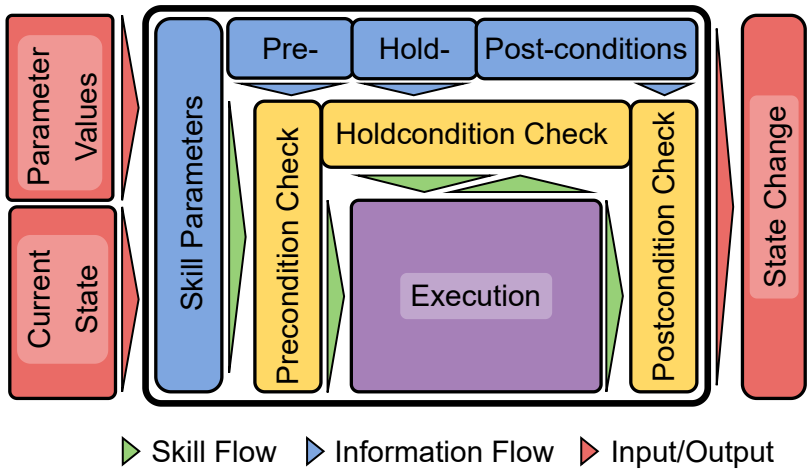


Figure 2.3: The conceptual model of a skill in SkiROS2. Pre-, and hold-conditions ensure that the skill is only executed in the correct world state. Post-conditions check if the desired changes have been achieved.

plementation implements exactly one description with its parameters and pre-, hold- and post-conditions. There are two complexities of skills: Primitive skills that are semantically atomic and compound skills. Primitive skills provide a Python interface for the initialization, start, execution and stop of a skill. In contrast to that, compound skills allow to connect an arbitrary number of primitive skills and compound skills in a BT [CÖ17a; RGK17]. A skill implementation is also allowed to change the skill description, such as adding addition conditions or modifying existing conditions. This is important to support multiple, specific skill implementations. Examples of this are gripper actuation skills. A basic skill description can have a parameter "gripper" of the WM type `Element("rparts:GripperEffector")` and a Boolean parameter "opening state". The implementation of such a skill depends on the actual gripper model and a specific implementation can set the parameter to a specific subtype of `rparts:GripperEffector`, such as `scalable:RobotiqGripper`.

3.3 Knowledge Representation and Task Planning

An important aspect is the separation of skills and tasks. Skills are written in a parametric way to be used in various applications and different tasks. This separation is achieved by explicitly storing knowledge in the WM. Since the conditions and task-specific knowledge is not implicitly represented in the skills, this allows for a modular and extensible design that easily allows to add scen-

```
scalable:Ur5-2 a scalable:Ur5, owl:NamedIndividual ;
  rdfs:label "scalable:ur5"^^xsd:string ;
  skiros:BaseFrameId "cora:Robot-1"^^xsd:string ;
  skiros:CartesianGoalAction
      "/cartesian_trajectory_generator/goal_action"^^xsd:string ;
  skiros:OverlayMotionService
      "/cartesian_trajectory_generator/overlay_motion"^^xsd:string
      ;
  skiros:CartesianStiffnessTopic
      "/cartesian_param_filter/stiffness_goal"^^xsd:string ;
  skiros:CartesianWrenchTopic
      "/cartesian_param_filter/force_goal"^^xsd:string ;
  skiros:CompliantController
      "cartesian_compliance_controller"^^xsd:string ;
  skiros:JointConfigurationController
      "scaled_pos_traj_controller"^^xsd:string ;
  skiros:DiscreteReasoner "AauSpatialReasoner"^^xsd:string ;
  skiros:FrameId "scalable:Ur5-2"^^xsd:string ;
  skiros:LinkedToFrameId "ur5e_base_link"^^xsd:string ;
  skiros:MotionExe
      "/scaled_pos_traj_controller/follow_joint_trajectory"
      ^^xsd:string ;
  skiros:MoveItGroup "manipulator"^^xsd:string ;
  skiros:MoveItReferenceFrame "ur5e_base_link"^^xsd:string ;
  skiros:MoveItTCPLink "ur5e_tcp_link"^^xsd:string ;
  skiros:hasA scalable:WsgGripper-3 .
```

Listing 2.1: An excerpt from the semantic description of one of the robots used in the case study. It includes the necessary knowledge to parameterize motion skills and perform spatial reasoning.

arios or robot systems at a later point. The WM stores this knowledge in a semantic RDF database that utilizes the OWL. SkiROS2 introduces its own ontology, which contains the necessary concepts and relations for skill modeling and reasoning. It is based on the *Core Ontology for Robotics and Automation (CORA)* that is in the *IEEE Standard Ontologies for Robotics and Automation (IEEE Std 1872™-2015)*.

An example for a part of the semantic description of a robot arm is in Listing 2.1. The listing contains the relevant knowledge for the trajectory generation and motion execution. This includes, but is not limited to properties such as ROS interfaces of a robot and the relevant aspects of a motion planning configuration. In addition there are elements needed for reasoning about relations such as the attached gripper.

Tasks are executed by the skill manager and there are two ways to start them: 1) Select and parameterize a skill manually. For example in the GUI or from an API or 2) by providing a planning goal to the task manager and executing the skill sequence. Tasks are concrete skill (sequence) instances with a specific goal. Within a task, the skills can utilize a blackboard to exchange information with their parameters.

4 Control and Motion Generation

4.1 Behavior Trees and Motion Generators

Behavior Trees (BT) are mathematical models used for plan representation and execution. They have been shown to perform reliably in games [CÖ14], artificial intelligence and robotics [Iov+22]. A BT is an acyclic graph defined with nodes and edges defined in a typical parent-child relationship. The nodes are of two types: 1) *control flow* and 2) *execution*. The *control flow* nodes are responsible for directing the flow of the tree and are traditionally classified on the basis of their operation as: 1) sequence, 2) selector, 3) parallel and 4) decorator [CÖ17a; Mar+14]. In essence, a BT works by means of sending a tick signal from the *Root* node. The signal then moves through the nodes controlled by the *control flow* nodes. The return statements from the *execution* nodes are *running*, *success* or *failure*. In order to combine BTs with task-level planning, [RGK17] proposes *extended BTs*.

A motion generator (MG) [Rov+18] generates an arm motion using impedance controllers to control the end effector (EE) in Cartesian space. A MG follows a simple attractor landscape that uses a virtual spring to attract the EE to the desired location, whereas virtual dampers allow safe motion by slowing down the overall motion. In addition, it also allows for a deviation by superimposing a motion on top of the original trajectory. For example, in [May+22c] we used a Cartesian linear trajectory superimposed by an Archimedes spiral to solve the peg-in-hole task.

A Behavior Tree and Movement Generator (BTMG) combines the strengths of both BT and MG to model robotic skills. A BTMG is a parameterized representation that allows us to specify not only the structure of BT but also the properties of the MG. In essence, the BTMG representation not only covers the aspects of plan representation and execution, but also determines the arm motions for the execution strategies [Ahm+22].

4.2 Trajectory Generation

The concept of the MG has been introduced in [Rov+18], which consists of both a trajectory generation method and a corresponding controller to execute the generated trajectory on the manipulator. In the following section, we provide a description of the trajectory generation process adopted for this project.

In many industrial settings, operations are performed in a Cartesian task space rather than in the joint space of the robot. Therefore, we argue that motion references are best generated and provided in the Cartesian end-effector space. In this work, we adopt Cartesian linear paths[1] to generate the corresponding trajectories. This is achieved by selecting appropriate acceleration profiles and setting the maximum Cartesian translational and rotational velocities. Furthermore, we offer the option to synchronize motions that have both translational and rotational components. Specifically, the trajectory generator takes the new goal pose as input and outputs end-effector pose references, which are then sent to the corresponding controller.

Aside from specifying a new goal pose, it is also possible to overlay additional motions on top of the reference pose. This capability has been utilized in various studies such as [May+21; May+22a; May+22c], where an Archimedes spiral is used to find a hole in an insertion procedure. In this work, we apply a sine motion to improve the wiping performance of the robot.

4.3 Compliant Control

Manipulation tasks that have uncertainties need compliant control solutions to provide the necessary flexibility. Such uncertainties can come from either the task itself or inaccurate knowledge about the placement of objects. Another source could be the placement of the robot, which becomes relevant when working with a mobile robot. The robot systems used in the case study provide different interfaces for control commands and therefore also have different solutions for compliant control.

Cartesian Impedance Control

Robot systems such as the *KUKA iiwa* or the *Franka Emika Robot* offer an interface to send commanded torques for each of the joints. This can be used

[1] https://github.com/matthias-mayr/cartesian_trajectory_generator

to perform Cartesian impedance control [MS22] which allows compliant control of the end effector in task space.

Torque-controlled robots are typically gravity compensated. The rigid-body dynamics of such a system in the joint space is described as $q \in \mathbb{R}^n$:

$$M(q)\ddot{q} + C(q,\dot{q})\dot{q} = \tau_\mathrm{c} + \tau^\mathrm{ext} \tag{2.1}$$

where $M(q) \in \mathbb{R}^{n\times n}$ is the generalized inertia matrix, $C(q,\dot{q}) \in \mathbb{R}^{n\times n}$ captures the effects of Coriolis and centripetal forces, $\tau_\mathrm{c} \in \mathbb{R}^n$ represents the input torques, and $\tau^\mathrm{ext} \in \mathbb{R}^n$ represents the external torques, with n being the number of joints of the robot.

Moreover, the torque signal commanded by this controller to the robot, τ_c in Equation (2.1), is composed of the superposition of three joint-torque signals:

$$\tau_\mathrm{c} = \tau_\mathrm{c}^\mathrm{ca} + \tau_\mathrm{c}^\mathrm{ns} + \tau_\mathrm{c}^\mathrm{ext} \tag{2.2}$$

where

$\tau_\mathrm{c}^\mathrm{ca}$ is the torque commanded to achieve a Cartesian impedance behavior with respect to a Cartesian pose reference in the m-dimensional task space, $\xi^\mathrm{D} \in \mathbb{R}^m$, in the frame of the end effector of the robot:

$$\tau_\mathrm{c}^\mathrm{ca} = J^\mathrm{T}(q)\left[-K^\mathrm{ca}\Delta\xi - D^\mathrm{ca}J(q)\dot{q}\right] \tag{2.3}$$

with $J(q) \in \mathbb{R}^{m\times n}$ being the Jacobian relative to the end-effector (task) frame of the robot, and $K^\mathrm{ca} \in \mathbb{R}^{m\times m}$ and $D^\mathrm{ca} \in \mathbb{R}^{m\times m}$ being the virtual Cartesian stiffness and damping matrices, respectively.

$\tau_\mathrm{c}^\mathrm{ns}$ is the torque commanded to achieve a joint impedance behavior with respect to a desired configuration and projected in the null-space of the robot's Jacobian, to not affect the Cartesian motion of the robot's end effector.

$\tau_\mathrm{c}^\mathrm{ext}$ is the torque commanded to achieve the desired external force command in the frame of the end effector of the robot, $F_\mathrm{c}^\mathrm{ext}$:

$$\tau_\mathrm{c}^\mathrm{ext} = J^\mathrm{T}(q)F_\mathrm{c}^\mathrm{ext} \tag{2.4}$$

In this work we utilize $\tau_\mathrm{c}^\mathrm{ext}$ and $\tau_\mathrm{c}^\mathrm{ca}$ in combination with changing stiffnesses by passing configurations through skill primitives.

Cartesian Compliance Control

In contrast to the *KUKA iiwa*, the *Universal Robots UR5e* like many other industrial manipulators does not support direct torque control. This means that

a compliant control solution is needed that can work with joint position or joint velocity commands. Recently, such a method was introduced through forward dynamics compliance control [SRD17]. It combines admittance, impedance and force control into one control strategy. The control loop is closed only through a force-torque sensor, which makes it system independent. Internally it utilizes a forward dynamics simulation of a virtual model of the robot system to map Cartesian inputs to joint commands. For further details, we refer the interested reader to [SRD17] and [SRD19]. Like the impedance controller that was introduced in the previous section, this controller solution also allows for the runtime modification of applied external forces, reference poses and stiffness changes. We utilize these capabilities when modeling behaviors with BTs.

5 Case Study: Wiping Task

We demonstrate the usage of skills in different tasks and on different robot system with the example of a wiping task.

5.1 Challenges

The task at hand and the robot systems selected for this study present a range of intriguing challenges. Firstly, the task instances involve surfaces with different properties, namely a smooth whiteboard and a rough industrial table. The whiteboard is mounted to a wall, whereas the table surface is parallel to the floor. Moreover, we assume that the precise distance between the robot and the surface may not be known in advance. The robot systems are equipped with different grippers, each of which has its own communication interface. Finally, the robots have distinct kinematics, vendor-supplied programming solutions, and control interfaces. In this context, we demonstrate how a skill that utilizes knowledge representation, planning and the automatic skill implementation selection can be executed in different context on different robot systems.

5.2 Implementation

The implementation of such a skill incorporates several key aspects that are discussed in [Rov+18]. Specifically, we utilize a motion generator (MG) that comprises of both trajectory generation (Sec.4.2) and a controller (Sec.4.3).

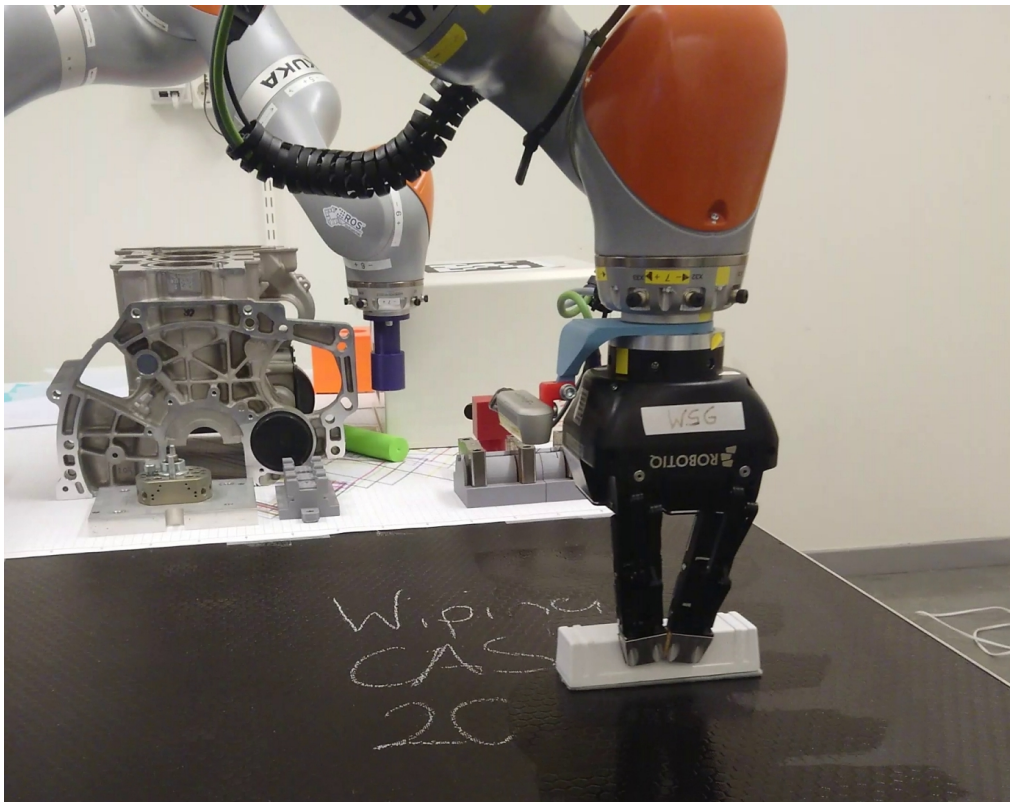


Figure 2.4: The *iiwa* wiping task on a table. The different hardware needs a different controller setup and gripper skills.

As inputs for the skill, only the robot arm and a surface to clean need to be selected among the available instances in the WM. All other necessary data, such as robot-specific properties (e.g., the controller to use) and the surface properties (e.g. the dimensions, force to apply) are automatically fetched from the WM as part of the knowledge integration. Apart from the parameters, this skill also contains the necessary pre- and postconditions for planning. The integrated plannerin SkiROS can receive a goal such as `(skiros:clean scalable:Workstation-1186 scalable:Cell-12)`. For instance, one precondition of the skill is that the tool must already be held in the gripper. This is leveraged in task-level planning, where the robot is required to pick up the tool first. In case of a mobile system, this may entail driving to a different location. The postcondition of the task is that the surface is cleaned.

To begin the wiping process, the end effector is moved to the corner of the

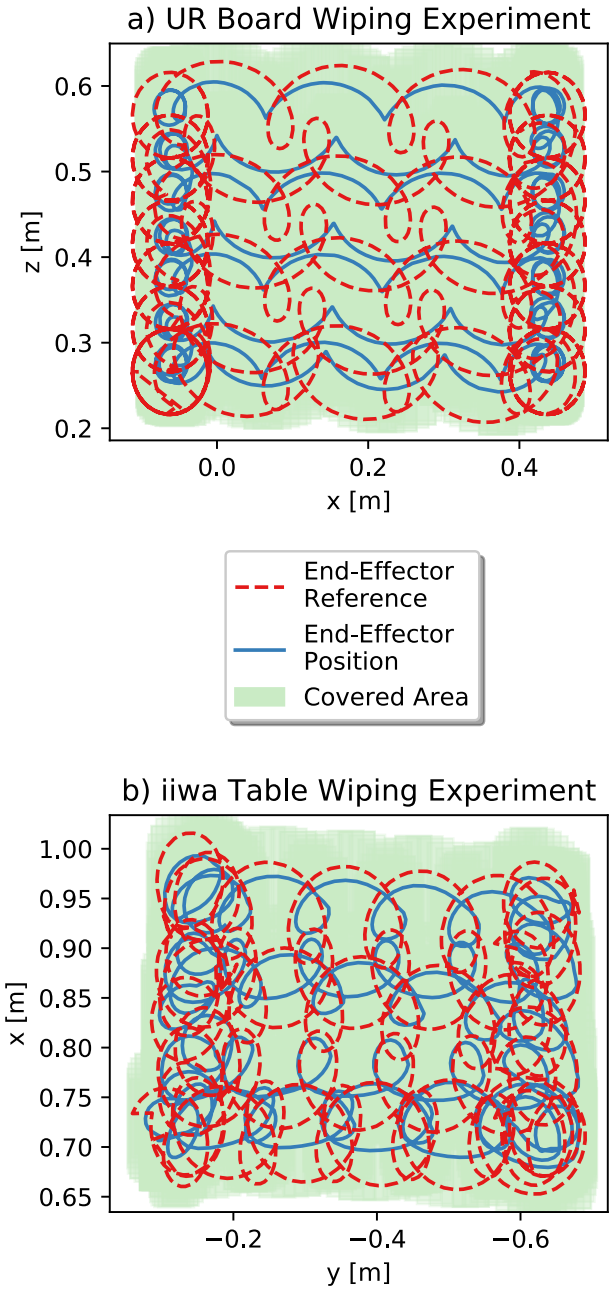


Figure 2.5: The resulting reference positions and actual positions of the end effectors in a) the whiteboard wiping task and with the *UR5e* b) the table wiping task with the *iiwa*. The shaded green shows the surface area that has been covered by the eraser that is used for cleaning.

surface to be cleaned. Once there, it disables the stiffness of the compliant controller along the normal of the surface and applies a predetermined force. In our experiments, 8 N are specified for surfaces that are easier to clean, such as the whiteboard, and 18 N for the table. The robot then applies an overlay motion, in this case a circular motion, while performing the wiping action. From there the skill executes laps by moving the robot's end-effector to the right, then down, and finally left until the entire surface is covered. Finally, upon completion of the task, the robot stops the force application, increases the stiffness along the surface normal, and moves the end-effector away from the surface.

5.3 Experiments and Discussion

We conducted the experiments using two different robot systems. The first system is a mobile platform that has a *UR5e* 6 degree-of-freedom robot arm with a *Schunk* two finger gripper. The second system is a 7 degree-of-freedom *KUKA iiwa* with a *Robotiq* 3-finger gripper. We selected two different surfaces for the cleaning task, each with different orientations and surface properties.

The resulting reference path and the actual path for the *UR5e* are shown in Fig. 2.5a). The implementation spends more time in the left and right extremes since the sub-trajectory ends there and it switches to new lane. This can be adressed by blending between trajectories. During the execution we also noticed subtle vibrations, especially when the arm was stretched out.

In our second experiment we used the *KUKA iiwa* robot system to wipe the surface of a table. As in the previous experiment, the important attributes were represented in the WM, and the skill could be started by simply selecting the surface and the robot arm. The reference path and the actual path for the robot end effector during the task is shown in Figure 2.5b). The controller is very stable and there are no vibrations. We observed that the eraser was occasionally lifted slightly during the wiping process. This could potentially be addressed by increasing the rotational stiffness or applying more pressure to the surface.

In both tasks, the respective robot systems were able to completely clean the surface with the proposed implementation. In principle, the surface can also be cleaned using only the Cartesian linear motion. However, as intuitively expected, the addition of an additional overlay motion improves the cleaning performance in practice.

6 Conclusions and Future Work

The transition towards Industry 4.0 brings challenges such as small batch sizes, constantly changing tasks and environments. Together with the transition towards human-robot colaboration and handling contact-rich tasks, it creates the need for platforms that can address these challenges.

In this paper we presented a complete architecture that allows to transfer skills between different robot systems. We explain the necessary prerequisites and knowledge representation. We demonstrated this by successfully performing contact-rich wiping tasks with a *KUKA iiwa* and a *Universal Robots UR5e*. All essential building blocks including the skill-based system and the controllers are available as open-source software and the results are expected to scale to other hardware and tasks as well.

While the wiping performance was sufficient in our tasks, it can potentially be increased by learning a good combination of path velocities and applied force. A visual inspection with a camera is a next inuitive step. We are also looking into an evaluation with other platforms such as the *Franka Emika Robot (Panda)*.

Acknowledgement

This work was partially supported by the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by Knut and Alice Wallenberg Foundation.

References

[Aer+98] Constructions Aeronautiques et al. “Pddl the Planning Domain Definition Language”. In: *Technical Report, Tech. Rep.* (1998).

[AMK23] Faseeh Ahmad, Matthias Mayr and Volker Krueger. “Learning to Adapt the Parameters of Behavior Trees and Motion Generators (BTMGs) to Task Variations”. In: *2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2023.

[Ahm+22] Faseeh Ahmad et al. “Generalizing Behavior Trees and Motion-Generator (BTMG) Policy Representation for Robotic Tasks over Scenario Parameters”. In: *2022 IJCAI Planning and Reinforcement Learning Workshop* (2022).

[ABR18] Helio Azevedo, José Pedro Ribeiro Belo and Roseli AF Romero. “OntPercept: A Perception Ontology for Robotic Systems”. In: *2018 Latin American Robotic Symposium, 2018 Brazilian Symposium on Robotics (SBR) and 2018 Workshop on Robotics in Education (WRE)*. IEEE, 2018, pp. 469–475.

[Ben+09] Saddek Bensalem et al. “Designing Autonomous Robots”. In: *IEEE Robotics & Automation Magazine* 16.1 (Mar. 2009), pp. 67–77.

[Bøg+12] Simon Bøgh et al. “Does Your Robot Have Skills?” In: *Proceedings of the 43rd International Symposium on Robotics*. VDE Verlag GmbH, 2012.

[Bru+19] Barbara Bruno et al. “The CARESSES EU-Japan Project: Making Assistive Robots Culturally Competent”. In: *Ambient Assisted Living: Italian Forum 2017 8*. Springer, 2019, pp. 151–169.

[CÖ14] Michele Colledanchise and Petter Ögren. “How Behavior Trees Modularize Robustness and Safety in Hybrid Systems”. In: *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*. Sept. 2014, pp. 1482–1488.

[CÖ17a] Michele Colledanchise and Petter Ögren. *Behavior Trees in Robotics and AI: An Introduction*. Chapman \& Hall/CRC Press, Aug. 2017.

[EMR12] Patrick Eyerich, Robert Mattmüller and Gabriele Röger. “Using the Context-Enhanced Additive Heuristic for Temporal and Numeric Planning”. In: *Towards Service Robots for Everyday Environments: Recent Advances in Designing Service Robots for Complex Tasks in Everyday Environments*. Ed. by Erwin Prassler et al. Springer Tracts in Advanced Robotics. Berlin, Heidelberg: Springer, 2012, pp. 49–64.

[Gib+18] Bernard Gibaud et al. “Toward a Standard Ontology of Surgical Process Models”. In: *International Journal of Computer Assisted Radiology and Surgery* 13.9 (Sept. 2018), pp. 1397–1408.

[INS01] A. J. Ijspeert, J. Nakanishi and S. Schaal. “Trajectory Formation for Imitation with Nonlinear Dynamical Systems”. In: *Proceedings 2001 IEEE/RSJ International Conference on Intelligent Robots and Systems. Expanding the Societal Role of Robotics in the the Next Millennium (Cat. No.01CH37180)*. Vol. 2. Oct. 2001, 752–757 vol.2.

[INS02] Auke Jan Ijspeert, Jun Nakanishi and Stefan Schaal. “Learning Rhythmic Movements by Demonstration Using Nonlinear Oscillators”. In: *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS2002)*. CONF. 2002, pp. 958–963.

[Iov+22] Matteo Iovino et al. “A Survey of Behavior Trees in Robotics and AI”. In: *Robotics and Autonomous Systems* 154 (Aug. 2022), p. 104096.

[Lei+15] Daniel Leidner et al. “Classifying Compliant Manipulation Tasks for Automated Planning in Robotics”. In: *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Sept. 2015, pp. 1769–1776.

[Lei+16] Daniel Leidner et al. “Robotic Agents Representing, Reasoning, and Executing Wiping Tasks for Daily Household Chores”. In: *Autonomous Agents and Multi-Agent Systems*. Singapore, 2016.

[Lei+19] Daniel Leidner et al. “Cognition-Enabled Robotic Wiping: Representation, Planning, Execution, and Interpretation”. In: *Robotics and Autonomous Systems* 114 (Apr. 2019), pp. 199–216.

[MJ13] Stenmark Maj and Malec Jacek. “Knowledge-Based Industrial Robotics”. In: *Frontiers in Artificial Intelligence and Applications* (2013), pp. 265–274.

[Man+21] Sumaira Manzoor et al. “Ontology-Based Knowledge Representation in Robotic Systems: A Survey Oriented toward Applications”. In: *Applied Sciences* 11.10 (Jan. 2021), p. 4324.

[Mar+14] A. Marzinotto et al. “Towards a Unified Behavior Trees Framework for Robot Control”. In: *2014 IEEE International Conference on Robotics and Automation (ICRA)*. May 2014, pp. 5420–5427.

[MS22] Matthias Mayr and Julian M Salt-Ducaju. “A C++ Implementation of a Cartesian Impedance Controller for Robotic Manipulators”. In: *arXiv preprint arXiv:2212.11215* (2022). arXiv: 2212.11215.

[May+21] Matthias Mayr et al. “Learning of Parameters in Behavior Trees for Movement Skills”. In: *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2021 IEEE/RSJ International Conference on Intelligent Robots and Systems. IEEE, 2021, pp. 7572–7579.

[May+22a] Matthias Mayr et al. “Combining Planning, Reasoning and Reinforcement Learning to Solve Industrial Robot Tasks”. In: *IROS 2022 Workshop on Workshop on Trends and Advances in Machine Learning and Automated Reasoning for Intelligent Robots and Systems* (2022).

[May+22b] Matthias Mayr et al. “Learning Skill-Based Industrial Robot Tasks with User Priors”. In: *2022 IEEE 18th International Conference on Automation Science and Engineering (CASE)*. IEEE, 2022, pp. 1485–1492.

[May+22c] Matthias Mayr et al. “Skill-Based Multi-Objective Reinforcement Learning of Industrial Robot Tasks with Planning and Knowledge Integration”. In: *2022 IEEE International Conference on Robotics and Biomimetics (ROBIO)*. IEEE, 2022, pp. 1995–2002.

[Par+13] Alexandros Paraschos et al. “Probabilistic Movement Primitives”. In: *Advances in Neural Information Processing Systems*. Vol. 26. Curran Associates, Inc., 2013.

[Pax+17] Chris Paxton et al. “CoSTAR: Instructing Collaborative Robots with Behavior Trees and Vision”. In: *2017 IEEE International Conference on Robotics and Automation (ICRA)*. May 2017, pp. 564–571.

[RGK17] F. Rovida, B. Grossmann and V. Krüger. “Extended Behavior Trees for Quick Definition of Flexible Robotic Tasks”. In: *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Sept. 2017, pp. 6793–6800.

[Rov+18] F. Rovida et al. “Motion Generators Combined with Behavior Trees: A Novel Approach to Skill Modelling”. In: *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Oct. 2018, pp. 5964–5971.

[SRD17] S. Scherzinger, A. Roennau and R. Dillmann. “Forward Dynamics Compliance Control (FDCC): A New Approach to Cartesian Compliance for Robotic Manipulators”. In: *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Sept. 2017, pp. 4568–4575.

[SRD19] Stefan Scherzinger, Arne Roennau and Rüdiger Dillmann. “Inverse Kinematics with Forward Dynamics Solvers for Sampled Motion Tracking”. In: *arXiv:1908.06252 [cs]* (Aug. 2019). arXiv: 1908.06252 [cs].

[Sta+16] Dennis Stampfer et al. “The SmartMDSD Toolchain: An Integrated MDSD Workflow and Integrated Development Environment (IDE) for Robotics Software”. In: (2016).

[SM15] Maj Stenmark and Jacek Malec. “Knowledge-Based Instruction of Manipulation Tasks for Industrial Robotics”. In: *Robotics and Computer-Integrated Manufacturing*. Special Issue on Knowledge Driven Robotics and Manufacturing 33 (June 2015), pp. 56–67.

[TB09] Moritz Tenorth and Michael Beetz. “KNOWROB — Knowledge Processing for Autonomous Personal Robots”. In: *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems*. Oct. 2009, pp. 4261–4266.

[TB13] Moritz Tenorth and Michael Beetz. “KnowRob: A Knowledge Processing Infrastructure for Cognition-Enabled Robots”. In: *The International Journal of Robotics Research* 32.5 (Apr. 2013), pp. 566–590.

[Top+18] Elin A. Topp et al. “Ontology-Based Knowledge Representation for Increased Skill Reusability in Industrial Robots”. In: *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Oct. 2018, pp. 5672–5678.

[Vol+01] R. Volpe et al. “The CLARAty Architecture for Robotic Autonomy”. In: *2001 IEEE Aerospace Conference Proceedings (Cat. No.01TH8542)*. Vol. 1. Mar. 2001, 1/121–1/132 vol.1.

[Wut+21] D. Wuthier et al. “Productive Multitasking for Industrial Robots”. In: *2021 IEEE International Conference on Robotics and Automation (ICRA)*. May 2021, pp. 12654–12661.

[Wya+10] Jeremy L. Wyatt et al. “Self-Understanding and Self-Extension: A Systems and Representational Approach”. In: *IEEE Transactions on Autonomous Mental Development* 2.4 (Dec. 2010), pp. 282–303.

Paper III

Paper 3

Learning of Parameters in Behavior Trees for Movement Skills

Matthias Mayr
Lund University
matthias.mayr@cs.lth.se

Konstantinos Chatzilygeroudis
University of Patras, Greece
costashatz@upatras.gr

Faseeh Ahmad
Lund University
faseeh.ahmad@cs.lth.se

Luigi Nardi
Lund University
Stanford University, USA
lnardi@stanford.edu

Volker Krueger
Lund University
volker.krueger@cs.lth.se

Abstract

Reinforcement Learning (RL) is a powerful mathematical framework that allows robots to learn complex skills by trial-and-error. Despite numerous successes in many applications, RL algorithms still require thousands of trials to converge to high-performing policies, can produce dangerous behaviors while learning, and the optimized policies (usually modeled as neural networks) give almost zero explanation when they fail to perform the task. For these reasons, the adoption of RL in industrial settings is not common. Behavior Trees (BTs), on the other hand, can provide a policy representation that a) supports modular and composable skills, b) allows for easy interpretation of the robot actions, and c) provides an advantageous low-dimensional parameter space. In this paper, we present a novel algorithm that can learn the parameters of a BT policy in simulation and then generalize to the physical robot without any additional training. We leverage a physical simulator with a digital twin of our workstation, and optimize the relevant parameters with a black-box optimizer. We showcase the efficacy of our method with a 7-DOF *KUKA-iiwa* manipulator in a task that includes obstacle avoidance and a contact-rich insertion (peg-in-hole), in which our method outperforms the baselines.

1 Introduction

Even in well-structured environments, like the ones in industrial settings, a desired feature of a robotic setup is to be able to improve over time and easily adapt to novel situations. Reinforcement Learning (RL) [Cha+19; DNP13] provides the theoretical framework that can allow robots to learn complex skills by trial-and-error. State-of-the-art RL methods, however, require at least a few hundred hours of interaction time to find an effective policy. Moreover, in the early stages of learning, those methods can produce unstable behaviors that can damage the robot or the environment (including human operators). Additionally, the optimized controllers, usually modeled as neural networks, cannot easily be interpreted, and when the robot fails at a task it is difficult to devise mitigation plans. In other words, operators have difficulties understanding *when* a robot is doing *what* and *why* and what kind of reliability can be expected. All things considered, controllers learned via pure RL do not satisfy the requirements for safety and production quality.

There exist three main axes for improving the data-efficiency of RL algorithms [Cha+19]: 1) inserting prior knowledge in the policy structure or parameters, 2) learning models of the objective function, and 3) learning or using models of dynamics. Using prior knowledge in the policy can be beneficial as it can a) create an easy to optimize optimization landscape, b) reduce the dimensionality of the search space, or c) provide a strong initial configuration that is close to the optimal value. Learning models using the gathered experience data or using dynamics models (e.g. simulators) allows the agents to perform interactions with the models, and thus reduce the interaction time needed on the physical system [DNP13; Cha+19].

Many types of policy structures have been proposed in the robot learning literature [Cha+19]. Among them, trajectory-based policy structures [KB11; Ude+10; SS13; Ijs+12] have received the most attention. One approach to encode trajectories is to define the policy as a sequence of way-points, while the most widely used approach is using policies based on dynamical systems. The latter type of policies has been used more extensively as they combine the generality of function approximators with the advantages of dynamical systems, such as robustness toward perturbations and stability guarantees [KB11; Krü+12; Ude+10; SS13], which are desirable properties of a robotic system. Parametric movement skills (MS) or motion generators [Rov+18] are another type of trajectory-based policy structure that explicitly operates in the end-effector space and allows for easy external modulation of their parameters. The advantage of this type of policy representation is that it is modular and easily interpretable. To the best

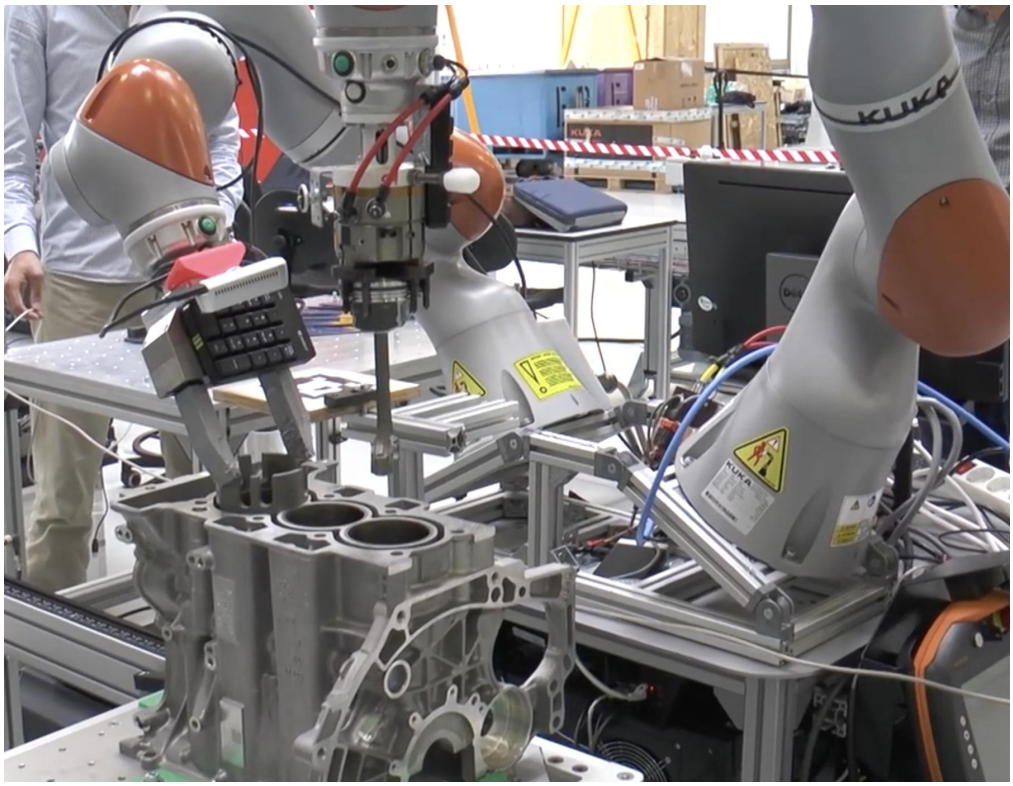


Figure 3.1: The robot setup used for the experiments for engine assembly at PSA: two *KUKA-iiwa* arms mounted on a torso to emulate a configuration similar to a human upper body. Policies are based on BTs and parametric movement skills.

of our knowledge, no prior work is available that combines them with learning.

In this paper, we propose a novel pipeline that is able to learn interpretable policies with no interaction on the physical robot when a digital twin (DT) of the workspace is available. In particular, we propose to use a parametric policy representation that is combining behavior trees (BT) [CÖ14; CÖ17a] with parametric movement skills (MS) [Rov+18], and to learn the parameters of the BT and the MS through reinforcement learning. We call this BT/MS policy representation approach *BTMS*. Since this BTMS policy representation is not differentiable, we solve the policy search using black-box optimization [Cha+17; Cha+19]. Black-box optimizers are able to optimize based solely on the input and the output and they do not make any assumptions on how the policy is modeled. In addition, we utilize domain randomization techniques [Tob+17; Meh+20] in order to generalize the behaviors from the DT to the physical sys-

tem.

Our formulation has several advantages over prior work, namely:

- It can be easily interpreted, and it allows not only to better predict the final robot behavior, but also to efficiently define priors for both structure and parameters for a given task [RGK17];
- It allows to decouple a desired policy into sub-policies that can be optimized separately;
- It provides a low-dimensional search space, and thus learning on the physical system (either from scratch or fine-tuning) is feasible;
- We do *not* require the physical setup to align perfectly with its digital twin.

To validate our proposed pipeline, we create the following scenario: a *KUKA-iiwa* 7-degree-of-freedom (DOF) industrial manipulator needs to insert a piston inside an engine while avoiding the obstacles in the workspace (Fig. 3.1). We emulate this scenario by requiring the manipulator to avoid a big obstacle (the engine block in Fig. 3.1) in the middle of the workspace, and to insert a peg into a hole with a $5\,mm$ clearance (i.e., difference between the hole size and the peg size) (Fig. 3.3).

Our results showcase that the proposed method can learn policies completely in simulation using the DT, and transfer successfully to the physical setup. The results also show that our BTMS policy representation leads to faster learning and better convergence than black-box policies (e.g., neural networks). Finally, using the proposed method we can split the overall policy into sub-policies, learn them separately, and easily combine them to perform the whole task; we show that we can even combine sub-policies where one of them is learned in simulation and the other one on the physical setup.

2 Related Work

2.1 Policy Search and Representation

Model-free policy search approaches have been used in many successful applications in robotics [DNP13]. They can easily be applied to RL problems with

high-dimensional continuous state spaces. However, they are not as sample-efficient and often need hundreds or thousands of episodes [Cha+19; PN17; DNP13].

Model-based methods utilize an internal model of the transition dynamics that allow to optimize a policy without interaction with the physical system. This is generally more efficient [DNP13; PN17], which is an important criterion since many scenarios in robotics need supervision by a human operator.

The commonly used policy representations include radial basis function networks [DNP13], dynamical movement primitives [Ijs+12; Ude+10] and feed-forward neural networks [DNP13; Cha+17]. In recent years deep neural nets seem to become a popular policy. All of them have in common that their final representation can be difficult to interpret. Even if a policy sets a target pose for the robot to reach, it can be problematic to know how it reacts in all parts of the state space.

2.2 Behavior Trees in Manipulation

Behavior trees are a policy representation that can be used for both planning and execution. They have been first described in computer games [CÖ14; Iov+22] and have successful applications in robotics [RGK17; Rov+18; Mar+14; Pax+17; Iov+22].

They are often chosen in skill-based systems [RGK17] because they are modular, interpretable and reactive. However, both their structure and parameters are often hard-coded for a specific task [Rov+18; Mar+14; CÖ14; Pax+17; Iov+22]. Frameworks like *extended BTs* [RGK17] allow to find both through planning. But this is limited to the capabilities of the reasoning system and can be challenging for contact-rich tasks. Furthermore, there is no mechanism to improve the task performance [RGK17].

Learning BTs is mostly exercised in game AI. A recent survey of BTs in robotics and AI [Iov+22] provides a detailed overview of the BT learning methods. In short, existing approaches mostly focus on evolving the structure of a tree [LBC10; Per+11; CPO18] or learning the behavior of *control flow nodes* [FQY16]. However in this work we are assuming that the structure is given *a priori*, e.g., through a framework like [RGK17], to assure better predictability, and learn the parameters of the BT through trial-and-error.

To the best of our knowledge, there is no previous work of applying reinforcement learning with BT policies in the context of manipulation [Iov+22].

Algorithm 1 BT Learning Process

procedure BTMS Policy Learning
  Define policy $\pi : \mathbf{x} \times \theta \rightarrow \mathbf{u}$
  for $i \leq it_{max}$ **do** ▷ $\arg\max_\theta \mathbb{E}[J(\theta)]$
    $\theta_i \sim \mathcal{N}(\mathbf{m}_k, \sigma_k^2 \mathbf{C}_k)$ ▷ CMA-ES
    $R_i = \varnothing$
    for $e \leq evals$ **do parallel** ▷ Episodes
      Set random initial robot state $\mathbf{x}_0$
      Set random initial simulation state
      $D_{i,e} = \varnothing$
      for $j = 0 \rightarrow T - 1$ **do**
        $\mathbf{u}_j = \pi(\mathbf{x}_j | \theta_i)$
        $\mathbf{x}_{j+1} = \text{execute\_on\_robot}(\mathbf{u}_j)$
        $D_{i,e} = D_{i,e} \cup \{(\mathbf{x}_j, \mathbf{u}_j)\}$
      end for
      $R_i = R_i \cup \text{reward}(D_{i,e})$ ▷ Sec. 4.1
    end for
    $r_i = mean(R_i)$
  end for
  $\theta^* = \text{get\_mean\_of\_last\_population}()$
end procedure

3 Approach

Overall, we propose to combine BTs with parametric MS to form a novel policy representation to be learned via reinforcement learning. Most of the learning happens in simulation, and we use domain randomization techniques so that the learned controllers generalize to the physical setup.

In the following subsections, we lay out our choices and approach. In Sec. 3.1 we present the Cartesian controller of the end-effector of our manipulator. In Sec. 3.2 we describe the parametric MS formulation used in our method that computes commands for the low-level Cartesian controller (Sec. 3.1). In Sec. 3.3 we describe the behavior tree formulation and how we use it in our setup to incorporate the parametric MS. Finally, in Sections 3.4, and 3.5 we describe the policy search formulation, and the domain randomization techniques used for our approach. The whole procedure is shown in Algorithm 1.

3.1 Robot Control

For our arm motions, we are controlling the end effector in Cartesian space. The desired behavior is defined in task coordinates $\mathbf{x}_{ee}$. A given joint configuration of the robot $\mathbf{q}$ can be converted into a Cartesian pose using forward kinematics $\mathbf{x}_{ee} = \mathbf{f}_{fk}(\mathbf{q})$. This allows to calculate the pose error $\tilde{\mathbf{x}}$ between $\mathbf{x}_{ee}$ and a time-varying *virtual equilibrium* or *attractor point* $\mathbf{x_d}$.

With inspiration from Cartesian impedance control, we formulate the joint commands given joint positions $\mathbf{q}$ and joint velocities $\dot{\mathbf{q}}$ according to

$$\dot{\mathbf{q}}_c = \mathbf{J}(\mathbf{q})^T \left(\mathbf{K}_d \tilde{\mathbf{x}} + \mathbf{D}_d \mathbf{J}(\mathbf{q}) \dot{\mathbf{q}}\right), \tag{3.1}$$

with $\mathbf{J}(\mathbf{q})$ being the Jacobian for the end effector in the absolute inertial coordinate frame, $\mathbf{K}_d$ the stiffness matrix and $\mathbf{D}_d$ the damping matrix.

Furthermore, we require the real robot to adhere to safety requirements for safe collaboration with humans at the workstation. This includes an adjustable maximum joint velocity and limiting the torques that can be exerted by the robot arm. To achieve that, we use a joint impedance controller on the real robot. This also means that robot motions can not be executed as accurately as with non-collaborative position-controlled robots. Due to the lack of a joint velocity interface, we integrate the joint velocity commands $\dot{\mathbf{q}}_c$ to obtain joint position commands $\mathbf{q}_c$.

In order to have an accurate representation in simulation, the robot model has strict limits on the joint torques.

3.2 Parametric Movement Skills

The reactive adaptation of movement skills in combination with BTs can be used to allow more robust assembly task automation [Rov+18]. Rovida et al. advocate to control the end effector in Cartesian space.

A Cartesian goal point $\mathbf{x}_g$ can be set and a path between the current end effector position $\mathbf{x}_{ee}$ and the goal position $\mathbf{x}_g$ is calculated. The attractor point $\mathbf{x}_d$ is moved along this path with a path velocity v_p. Furthermore, the pose $\mathbf{x}_d$ can be overlayed with additional motions that can be configured by the BT. We use an Archimedes spiral, which is a common method in peg-in-hole assembly tasks. The formulation follows [Par+17] and defines a spiral trajectory in polar coordinates

$$\begin{cases} \delta\alpha = \frac{v_s}{r_i} \\ r_{i+1} = r_i + \delta\alpha \frac{c}{2\pi}, \end{cases} \tag{3.2}$$

with a path velocity v_s and a pitch, i.e. the distance between lines in the spiral, c.

One major advantage of this approach is that the movements can be defined in the task frame [Rov+18]. This allows to easily relocate a task frame through planning or sensing and still have a valid motion configuration. Furthermore, it offers an easier transfer of skills to other robots [Top+18]. We use motion generators as the movement skills in our policy representation, please refer to [Rov+18] for details.

3.3 Behavior Trees

A Behavior Tree (BT) [CÖ17a] is a plan representation and execution tool. A BT can be defined as a directed acyclic graph $G(V, E)$ with $|V|$ nodes and $|E|$ edges. It consists of *control flow nodes* or *processors*, and *execution nodes*. The classical formulation defines four types of *control flow nodes*: 1) *sequence*, 2) *selector*, 3) *parallel* and 4) *decorator* [Mar+14]. A BT has always an initial node with no parents, defined as *Root*, and one or more nodes with no children, called *leaves*.

The execution of a BT is done by periodically injecting a *tick* signal into the *Root*. The signal is routed according to the *control flow nodes* and the return statements of the children. By convention, the signal propagation goes from left to right. This can be exploited to set priorities.

A *sequence* node can be seen as a logical *AND*: it succeeds if all children succeed, and fails if one child fails. The *selector*, also called *fallback* node, represents a logical *OR*: It fails only if all children fail. If one child succeeds, the remaining ones will not be ticked. The *parallel* control flow node forwards ticks to all children and fails if one fails. A *decorator* allows to define custom functions. Leaves of the BT are the *execution nodes* that, when ticked, perform an operation and output one of the three signals: *success*, *failure* or *running*. In particular, execution nodes subdivide into 1) *action* and 2) *condition* nodes. An action performs its operation iteratively at every tick, returning *running* while it is not done, and *success* or *failure* otherwise. A condition never returns running: it performs an instantaneous operation and returns always *success* or *failure*.

In this paper, to assure predictability of the robot behaviors, we assume that a structure is given *a-priori*, only the parameters of the BT and the motion skills need to be determined. We utilize nodes and conditions in the BT to set the configuration of the motion generator in a reactive and easy to understand way.

Fig 3.2 shows the BT that we are using.

3.4 Policy Optimization

In order to optimize for policy parameters, we adopt the policy search formulation [DNP13; Cha+19; Cha+17]. In general, we model the system as a dynamical system of the form:

$$\mathbf{x}_{t+1} = \mathbf{x}_t + M(\mathbf{x}_t, \mathbf{u}_t) + f(\mathbf{x}_t, \mathbf{u}_t), \tag{3.3}$$

with continuous valued states $\mathbf{x} \in \mathbb{R}^E$ and continuous valued actions $\mathbf{u} \in \mathbb{R}^U$. The transition dynamics can generally be modeled as a combination of a simulation of the robot $M(\mathbf{x}_t, \mathbf{u}_t)$ and $f(\mathbf{x}_t, \mathbf{u}_t)$ which models the residual between simulation and reality. Here, we do not model the function f. Instead, we use domain randomization for finding a policy that is robust enough such that we can ignore f.

The goal is to find a policy $\pi, \mathbf{u} = \pi(\mathbf{x}|\theta)$ with policy parameters θ such that $\mathbf{u}$ maximizes the expected long-term reward when executing the policy for T time steps:

$$J = \mathbb{E}\left[\sum_{t=1}^{T} r(\mathbf{x}_t)|\theta\right], \tag{3.4}$$

with $r(\mathbf{x}_t)$ being the immediate reward for being in state $\mathbf{x}$ at time step t.

In our approach the movement skills in our policy π define the movements of the manipulator in Cartesian space in terms of goal points. A BT is used to ensure the configuration depending on the state of the robot [RGK17; Mar+14].

Since our policy representation is not differentiable, we frame the optimization of Eq. (3.4) as a black-box optimization, and seek the maximization of a reward function $J(\theta)$ by only using measurements of the function. Covariance Matrix Adaptation Evolution Strategy (CMA-ES) [Han06] is a stochastic, derivative-free method for numerical optimization of non-linear or non-convex continuous optimization problems. It models a population of points as a multivariate normal distribution. CMA-ES performs three steps at each generation k, we defer to [Han06] for details:

1. Reproduction: sample λ new offspring according to a multi-variate Gaussian distribution of mean $\mathbf{m}_k$ and covariance $\sigma_k^2\mathbf{C}_k$, that is,
   $\theta_i \sim \mathcal{N}(\mathbf{m}_k, \sigma_k^2\mathbf{C}_k)$ for $i \in 1, ..., \lambda$;

2. Truncation selection: rank the λ sampled candidates based on their performance $J(\theta_i)$ and select the fittest μ individuals with $\mu \leq \lambda$;

3. Gaussian update: to reflect the distribution of the μ best candidates, compute $\mathbf{m}_{k+1}$ by averaging the μ individuals: $\mathbf{m}_{k+1} = \frac{1}{\mu} \sum_{i=1}^{\mu} \theta_i$, and $\sigma_{k+1}^2 \mathbf{C}_{k+1}$.

In our experiments, we use BIPOP-CMA-ES with restarts [Han09; Cul+18][1], which is a refinement over canonical CMA-ES. At convergence, it restarts the optimization process with a bigger population λ to increase exploration. In our implementation, we sample a new parameter configuration θ_i at the beginning of an iteration in line 4 of Algorithm 1. When all λ offspring of a population have been evaluated, the algorithm performs the truncation selection and Gaussian update.

Examples of parameters to optimize for are location of goal points in Cartesian space or thresholds in conditions in the BT.

3.5 Domain Randomization

Domain randomization is used to bridge the *reality gap* between the digital twin and the physical robot. The idea of domain randomization is to introduce enough variability into the simulation such that the real physical robot may appear as just another variation of the simulation [Tob+17; Ngu+18; Che+18]. It can also avoid overfitting to a specific solution that might perform very well in simulation, but would fail on the real robot [KMD13]. Furthermore, we are also interested to learn robust policies that can address uncertainties in the environment. In our scenarios, a given policy is evaluated in multiple simulations. See line 6 in Algorithm 1. At the beginning of each evaluation, a varying starting position for the robot arm (line 7) and a random displacement of objects in the workplace (line 8) is applied. We calculate the mean reward of all simulations in line 17 and assign it as the value of the objective function. Furthermore, at the end of a learning procedure in line 19, we get the mean of the last population of CMA-ES rather than the best performing offspring. The reasoning is that it should be a more robust configuration and the chances that it succeeds on the physical system are higher. This allows for our policy to generalize to inevitable differences between the physical setup and its DT.

[1] We use the open source CMA-ES integration in the Limbo package [Cul+18].

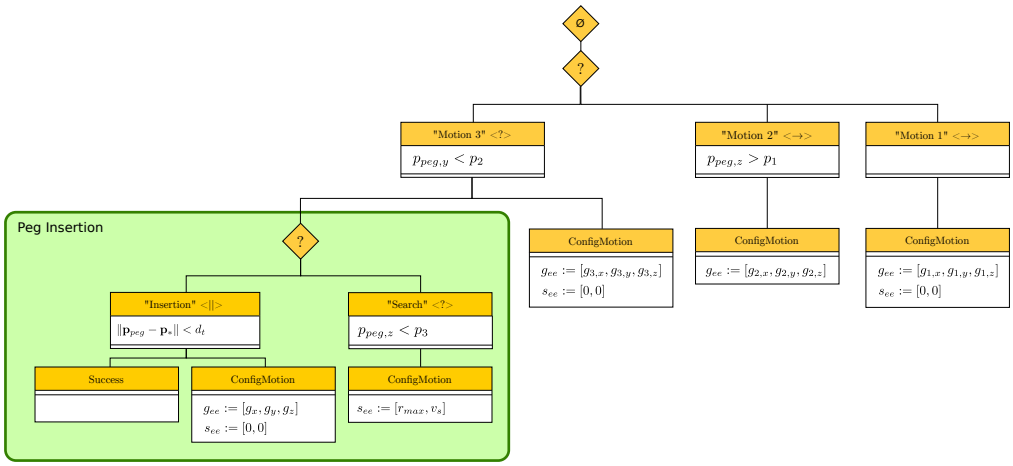


Figure 3.2: The behavior tree used for the combined task in the extended BT format [RGK17]. Each node has conditions shown in the upper half and effects shown in the lower half. The two explicit *control flow nodes* are *selectors* indicated by a question mark. They forward a tick to their children from the left to the right until one returns running or success. Hence "Motion 1" is only executed if the conditions of the other children do not hold. The top right part of the tree handles the obstacle avoidance. The bottom left part of the tree handles the peg-in-hole task.

4 Experimental Results

We evaluate our approach with a *KUKA-iiwa* 7-DOF manipulator (see Fig. 3.1). The robot is controlled by our own re-implementation of the motion generator described in Sec. 3.2. We utilize the *DART* simulator [Lee+18].

For our experiments we use a mock-up of our workstation shown in Fig. 3.3. The goal is to reach a target position (g_3) on the other side of an obstacle (e.g., the engine block in Fig. 3.1) and perform a peg insertion on that side. In order for the task to succeed, two parameters in the BT and several movement skill parameters need to be learned.

The BT is shown in Fig. 3.2. It consists of the bottom left part that models the peg insertion marked in green and the top right part that is parameterized to avoid the obstacle. Learning the whole task with a real robot arm would include many undesired collisions with the environment. Instead, we use the property of BTs that a node in the tree can be replaced with any number of nodes [CÖ14]. When learning the obstacle avoidance task, the bottom-left BT for the peg insertion is replaced with a condition that reports *success* when the end effector is close to the hole. This way, we can separate the two tasks. Therefore, we

can learn the obstacle avoidance task in simulation and minimize the usage of the real system and avoid the risk of damages. We then learn the peg insertion in simulation as well as with the real robot and compare the results. Here, learning the peg insertion does not pose a danger for the environment or the robot. Finally, we re-combine the two separately learned policies two evaluate the whole task.

4.1 Rewards

For our tasks, we use the following intuitive rewards:

1. Finish the task;
2. Closeness to the goal;
3. Negative reward close to obstacles;
4. Closeness to the hole of the workpiece.

In principle, these rewards can be provided by the DT. The DT already provides the information about the goal and if an object is to be avoided.
1. Finish the task: As explained in Section 3.3, a BT can succeed, run or fail. This feedback can be used to easily determine if the task was successfully finished. For that purpose we use a task-dependent fixed reward.
2. Goal position: We use the Euclidean distance between the position of the end effector and the target. We utilize the same function used in [DR11a] and [Cha+17], an exponential function:

$$r_g(\mathbf{x}) = \exp\left(-\frac{1}{2\sigma_c^2}(\|\mathbf{p}_{peg,\mathbf{x}} - \mathbf{p}_*\| + d_g)\right), \tag{3.5}$$

where $\sigma_c = 0.4$, $\mathbf{p}_{peg,x}$ corresponds to the position of the peg in state $\mathbf{x}$ and $\mathbf{p}_*$ is the goal position. We add a minimum distance $d_g = 0.25$ to obtain a steeper slope next to the goal.
3. Collisions: Keeping a safe distance to a certain object in the environment like the table and the engine in Fig. 3.1 is often a basic requirement when parameterizing a policy. This can be easily achieved without additional sensing using the simulation. The objects are known in the DT, and a negative reward close to them can easily be defined:

$$r_a(\mathbf{x}) = -\frac{1}{(d(\mathbf{p}_{peg,\mathbf{x}}, \mathbf{p}_{obj}) + d_a)^2}, \tag{3.6}$$

with $d_a = 0.03$ to avoid a division by zero and calculating the shortest distance $d(\mathbf{p}_{peg,\mathbf{x}}, \mathbf{p}_{obj})$. Furthermore, the same negative reward can be utilized to avoid areas that might be dangerous for the robot or an operator.

4. Closeness to the hole: This quite localized reward acts as an additional attractor towards the desired assembly goal location. We use

$$r_h(\mathbf{x}) = \frac{1}{2\left(d(\mathbf{p}_{peg,\mathbf{x}}, \mathbf{p}_h) + d_h\right)}, \tag{3.7}$$

with $d_h = 0.006$ and the shortest distance function $d(\mathbf{p}_{peg,\mathbf{x}}, \mathbf{p}_h)$. $\mathbf{p}_h$ is a hyper-rectangle in the center of a hole with two sides of 2 mm and stretching from the bottom to 1 mm below the surface.

4.2 Movements with Avoidance of a Static Obstacle

Intuitively, the shape of the desired trajectory suggests to use two intermediate goal points and two conditions for nodes in the BT. The pre-defined structure of the BT can be seen in Fig. 3.2, top-right, and an interpretation is shown in Fig. 3.3. In the configuration shown in Fig. 3.3, the end effector follows the configuration "Motion 1" towards goal (g_1) until the condition $pos_{p,z} > p_1$ of the branch "Motion 2" is fulfilled. This branch is ticked before and executes if the condition is fulfilled.

The parameters that need to be determined include both conditions of nodes p_1 and p_2 as well as the position of the goal points g_1 and g_2 along two axis. The structure of the BT has redundancies since the thresholds and the positions of the goal points interplay with each other.

We learned 10 policies with 5000 iterations each. All policies that successfully executed in simulation also successfully executed on the real system and there were no collisions with the environment.

We compare our approach to a feed-forward neural net that has one hidden layer with ten neurons. The function of the i^{th} layer is $\mathbf{y}_i = \phi_i\left(\mathbf{W}_i\mathbf{y}_{i-1} + \mathbf{b}_i\right)$, with $\mathbf{W}_i$ and $\mathbf{b}_i$ being the weight matrix and bias vector. $\mathbf{y}_{i-1}$ and $\mathbf{y}_i$ are the input and output vectors and ϕ_i is the activation function. We use the hyperbolic tangent as the activation function for all the layers. The output layer sets the target position of the end effector.

The development of the reward in Fig. 3.4 shows that BTMS policies learned significantly faster than neural nets. Neural nets could eventually outperform the given structure of the movement skills. However, even though they control

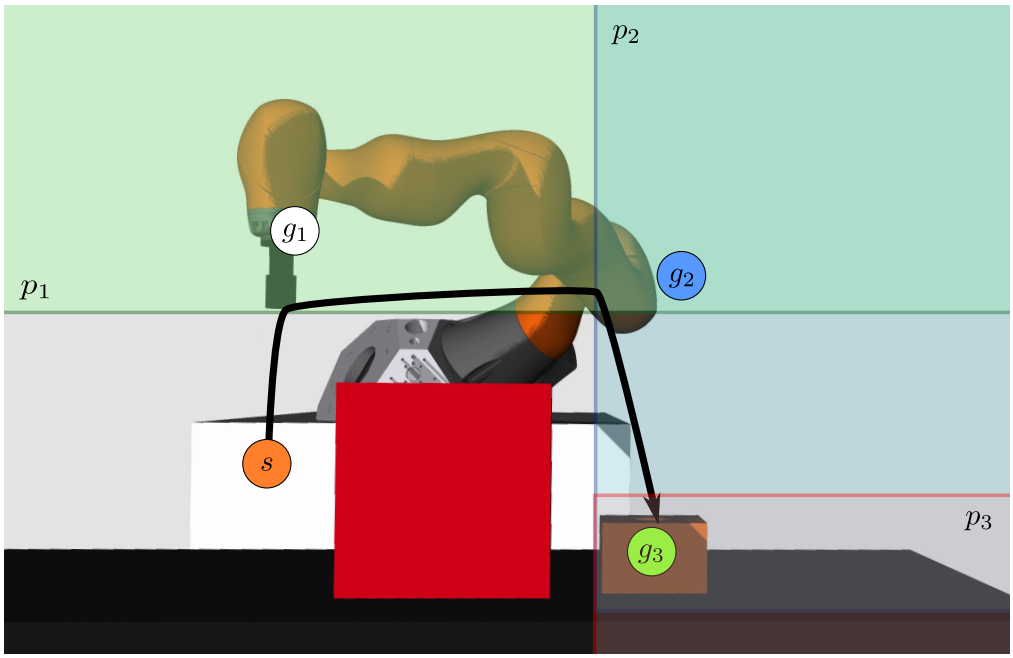


Figure 3.3: The setup of the experiment in simulation in one possible motion configuration. Two thresholds that are set by the parameters p_1 and p_2 are visualized, as well as the parametric goal points (g_1, g_2). The trajectory between the start point s and the target g_3 is outlined.

the end effector goal position, the motions can be much harder to predict. In fact, it is also not clear how this neural net-based policy will handle perturbations. In case of the BTMS policy, the underlying BT of the policy will simply evaluate the end-effector pose making thus the reaction predictable.

4.3 Peg-in-Hole Task

The peg-in-hole task is a common assembly task, e.g., it resembles the piston insertion from [Rov+18]. The setup for the experiment is shown in Fig. 3.3. We know the position of the box within the frame of the workstation with a precision error of less than 10 mm. Based on the idea from [Rov+18] we allow the robot to make a search movement close to the surface of the box in order to find the hole. Intuitively, this is similar to the movement humans would do as well.

Tasks that involve contact forces are much harder to learn than free-space movements. We learn both the z-coordinate of the goal point g_3 that influences the

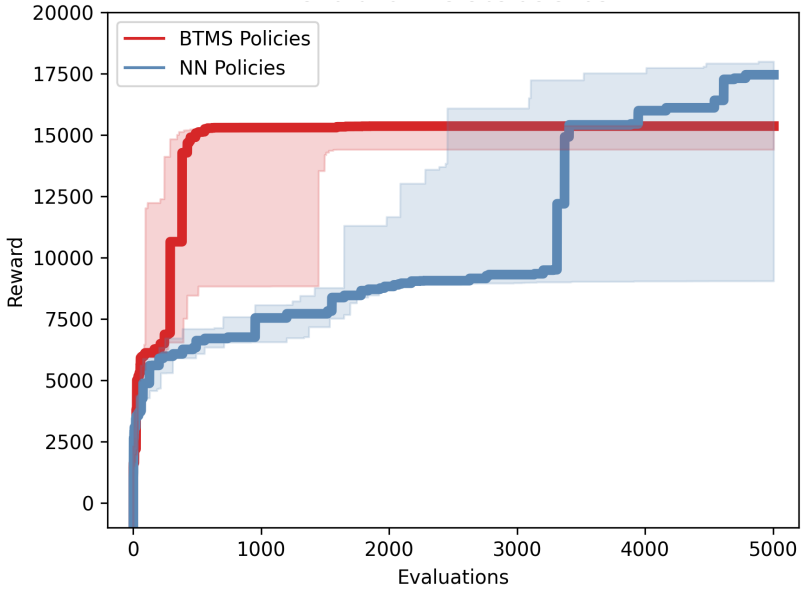


Figure 3.4: The development of the best reward when learning the obstacle task in simulation. The lines are the median values and the shaded region the 25^{th} and 75^{th} percentile (10 replicates). BT policies converge faster and with less variance.

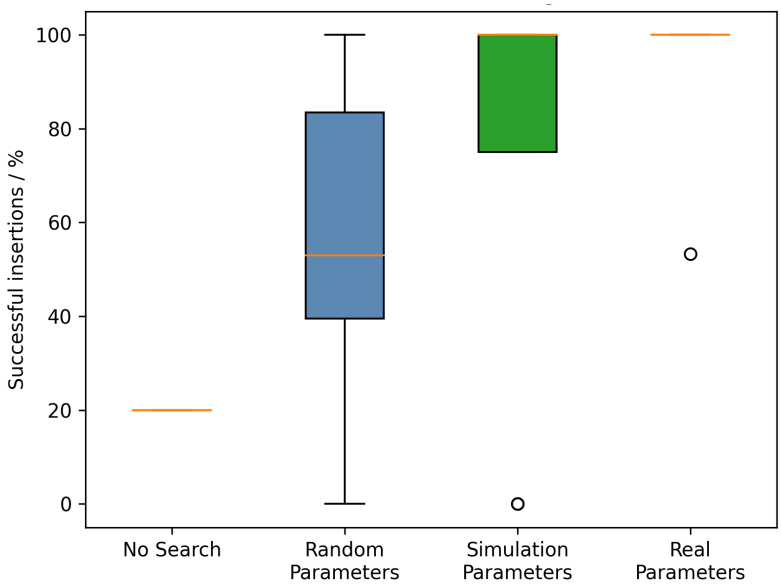


Figure 3.5: The success rates of the peg-in-hole experiment (15 trials per policy). The box plots show the median (red line) and interquartile range (25^{th} and 75^{th} percentile); the whiskers extend to the most extreme data points not considered outliers, and outliers are plotted individually. The learned search parameters clearly improve the insertion rate. Parameters learned on the physical robot have less negative outliers.

pressure that is applied on the surface and the search path velocity v_s. The peg can get stuck or can jump over the hole if there is too little pressure for a given path velocity v_s. The radius of the peg is 2.5 mm smaller than the hole, and an insertion is called successful if the peg is more than 10 mm inside the hole.

In our particular setup, the robot is configured in a low stiffness mode to ensure safety guarantees in human-robot collaborative workspaces. This also means that there can be deviations from the path and that it can be perturbed by a human operator. Therefore, we learn the task with five different start positions and expect to learn a policy that generalizes to other configurations.

We learn policies with 5000 iterations in simulation. The episode length is set to 15 s and each episode starts from one of the five random start positions and a random displacement of the hole. Again, learning in simulation allows to highly parallelize the procedure and does not block or damage expensive hardware.

The learned policies are evaluated on the physical setup with the five start positions used for training and ten previously unknown positions (15 in total). All start positions are 40 cm above the hole and have deviations in x and y direction between 2 cm and 5 cm. One position is exactly above the hole. We allow an episode length of 25 s.

The results are presented in Fig. 3.5. As a baseline we use a peg insertion without a search (Fig. 3.5, left) and randomly sampled search parameters (Fig. 3.5, second from left). The results show that without a search motion a success rate of only 20 % was achieved. Randomly sampled parameters had a median insertion rate of only 53 %. The parameters learned in simulation (Fig. 3.5, third from left) performed with a median success rate of 100 % generally well, but had negative outliers. However, these are easy to detect using the success criterion of the BT since they also perform significantly worse in simulation.

We acknowledge that an accurate enough simulation of a contact-rich task might not always be available. Therefore, we also learned six policies with 200 iterations on the real robot. With the episode length of 15 s and approximately 6 s to reset, the training took about 75 min per policy. The learning is safe in the sense that the robot can not collide with other objects in the workplace. The parameters learned with the real system showed the most consistent performance with a median success rate of 100 % and only one outlier that still achieved 53 %.

4.4 Combining the Tasks

The modular nature of the BTs allows us to combine the policies of the two tasks into one or split a large policy into sub-policies. In this experiment we want to explore 1) if two policies that are learned in simulation and reality, respectively, can be combined, 2) how well the combined policies perform and 3) to which extend it can accomplish the task if the hole is displaced.

We combine the obstacle task policies learned in simulation with the peg insertion policies learned with the real system one-to-one and evaluated them on the BT shown in Fig. 3.2. We assessed if they 1) can be executed on the robot, 2) collide with the environment and 3) successfully insert the peg. In addition to the hole being in the specified location, we also evaluated with deviations of $\pm 5\,\text{mm}$ and $\pm 10\,\text{mm}$ in both horizontal directions x and y. We allow an episode length of 30 s.

None of the policies collide with the environment. In case of a perfect positioning of the hole an insertion rate of 100 % was achieved. A misalignment of $\pm 5\,\text{mm}$ led to a success rate of 91.66 %, while a displacement of 10 mm still allowed an insertion rate of 83.3 %. These results showcase the modularity and efficacy of our BTMS method.

5 Conclusions and Future Work

Motivated by [Rov+18] we introduced a pipeline to learn tasks with the BTMS policy representation that uses BTs and parametric movement skills. It is an interpretable, robust and modular policy for robot manipulators that learns faster than black-box policies, e.g., neural nets. We use a black-box optimizer to learn the paramters of the BTMS policy in simulation. Domain randomization assured that policies are robust enough to cope with differences between the DT and the real robot.

We demonstrated the BTMS policy learning with a free-space movement and a peg-in-hole task. We also showed how these two can be easily combined into a single, more complex policy. Furthermore, we demonstrate that sub-policies learned in simulation can be combined with counterparts learned with the real manipulator. For many tasks this policy representations offers a low dimensional search space that even allows to learn with physical robots.

To the best of our knowledge this is the first application of RL to the para-

meterization of behavior trees with manipulators. This opens up a new way to use and parameterize BTs. Furthermore, this representation allows to easily combine learning with reasoning by providing prior knowledge in terms of the structure or the parameters of the BT. An obvious limitation is that learning cannot be successful if the policy is not flexible enough to allow solutions for a particular task.

We plan to align the simulation more with reality using parameterized prior models [CM18] to further improve the performance for more challenging assembly tasks. We are also planning to look into learning tasks that require dual-arm coordination. Finally, the approach is not limited to movement skills, and we plan to use it with other parameterizable skills such as grasping policies.

APPENDIX

The source code and the submitted video are available at
`https://github.com/matthias-mayr/behavior-tree-policy-learning`

Acknowledgement

This work was partially supported by the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by Knut and Alice Wallenberg Foundation. This research was also supported in part by affiliate members and other supporters of the Stanford DAWN project—Ant Financial, Facebook, Google, InfoSys, Teradata, NEC, and VMware.

We want to thank Francesco Rovida, Bjarne Großmann, Alexander Dürr, Björn Olofsson and Julian Salt for the interesting discussions and the constructive feedback.

References

[CM18] Konstantinos Chatzilygeroudis and Jean-Baptiste Mouret. "Using Parameterized Black-Box Priors to Scale Up Model-Based Policy Search for Robotics". In: *2018 IEEE International Conference on Robotics and Automation (ICRA)*. May 2018, pp. 5121–5128.

[Cha+17] Konstantinos Chatzilygeroudis et al. “Black-Box Data-Efficient Policy Search for Robotics”. In: *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Sept. 2017, pp. 51–58.

[Cha+19] Konstantinos Chatzilygeroudis et al. “A Survey on Policy Search Algorithms for Learning Robot Controllers in a Handful of Trials”. In: *IEEE Transactions on Robotics* (2019), pp. 1–20.

[Che+18] X. Chen et al. “Deep Reinforcement Learning to Acquire Navigation Skills for Wheel-Legged Robots in Complex Environments”. In: *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Oct. 2018, pp. 3110–3116.

[CPO18] M. Colledanchise, R. Nattanmai Parasuraman and P. Ogren. “Learning of Behavior Trees for Autonomous Agents”. In: *IEEE Transactions on Games* (2018), pp. 1–1.

[CÖ14] Michele Colledanchise and Petter Ögren. “How Behavior Trees Modularize Robustness and Safety in Hybrid Systems”. In: *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*. Sept. 2014, pp. 1482–1488.

[CÖ17a] Michele Colledanchise and Petter Ögren. *Behavior Trees in Robotics and AI: An Introduction*. Chapman \& Hall/CRC Press, Aug. 2017.

[Cul+18] Antoine Cully et al. “Limbo: A Flexible High-performance Library for Gaussian Processes Modeling and Data-Efficient Optimization”. In: *Journal of Open Source Software* 3.26 (June 2018), p. 545.

[DNP13] Marc Peter Deisenroth, Gerhard Neumann and Jan Peters. “A Survey on Policy Search for Robotics”. In: *Foundations and Trends® in Robotics* 2.1–2 (Aug. 2013), pp. 1–142.

[DR11a] Marc Peter Deisenroth and Carl Edward Rasmussen. “PILCO: A Model-Based and Data-Efficient Approach to Policy Search”. In: *Proceedings of the 28th International Conference on machine learning (ICML-11)* (2011), pp. 465–472.

[FQY16] Yanchang Fu, Long Qin and Quanjun Yin. “A Reinforcement Learning Behavior Tree Framework for Game AI”. In: *2016 International Conference on Economics, Social Science, Arts, Education and Management Engineering*. Atlantis Press, Aug. 2016.

[Han06] Nikolaus Hansen. “Towards a New Evolutionary Computation”. In: *Studies in Fuzziness and Soft Computing* 192 (2006), pp. 75–102.

[Han09] Nikolaus Hansen. “Benchmarking a BI-population CMA-ES on the BBOB-2009 Function Testbed”. In: *Proceedings of the 11th Annual Conference Companion on Genetic and Evolutionary Computation Conference: Late Breaking Papers*. GECCO ’09. Montreal, Québec, Canada: Association for Computing Machinery, July 2009, pp. 2389–2396.

[Ijs+12] Auke Jan Ijspeert et al. “Dynamical Movement Primitives: Learning Attractor Models for Motor Behaviors”. In: *Neural Computation* 25.2 (Nov. 2012), pp. 328–373.

[Iov+22] Matteo Iovino et al. “A Survey of Behavior Trees in Robotics and AI”. In: *Robotics and Autonomous Systems* 154 (Aug. 2022), p. 104096.

[KB11] S. M. Khansari-Zadeh and A. Billard. “Learning Stable Nonlinear Dynamical Systems With Gaussian Mixture Models”. In: *IEEE Transactions on Robotics* 27.5 (Oct. 2011), pp. 943–957.

[KMD13] S. Koos, J. Mouret and S. Doncieux. “The Transferability Approach: Crossing the Reality Gap in Evolutionary Robotics”. In: *IEEE Transactions on Evolutionary Computation* 17.1 (Feb. 2013), pp. 122–145.

[Krü+12] V. Krüger et al. “Imitation Learning of Non-Linear Point-to-Point Robot Motions Using Dirichlet Processes”. In: *2012 IEEE International Conference on Robotics and Automation*. May 2012, pp. 2029–2034.

[Lee+18] Jeongseok Lee et al. “DART: Dynamic Animation and Robotics Toolkit”. In: *Journal of Open Source Software* 3.22 (Feb. 2018), p. 500.

[LBC10] Chong-U Lim, Robin Baumgarten and Simon Colton. “Evolving Behaviour Trees for the Commercial Game DEFCON”. In: *Applications of Evolutionary Computation*. Ed. by Cecilia Di Chio et al. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2010, pp. 100–110.

[Mar+14] A. Marzinotto et al. “Towards a Unified Behavior Trees Framework for Robot Control”. In: *2014 IEEE International Conference on Robotics and Automation (ICRA)*. May 2014, pp. 5420–5427.

[Meh+20] Bhairav Mehta et al. “Active Domain Randomization”. In: *Conference on Robot Learning*. PMLR, May 2020, pp. 1162–1176.

[Ngu+18] P. D. H. Nguyen et al. “Transferring Visuomotor Learning from Simulation to the Real World for Robotics Manipulation Tasks”. In: *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Oct. 2018, pp. 6667–6674.

[Par+17] H. Park et al. “Compliance-Based Robotic Peg-in-Hole Assembly Strategy Without Force Feedback”. In: *IEEE Transactions on Industrial Electronics* 64.8 (Aug. 2017), pp. 6299–6309.

[Pax+17] Chris Paxton et al. “CoSTAR: Instructing Collaborative Robots with Behavior Trees and Vision”. In: *2017 IEEE International Conference on Robotics and Automation (ICRA)*. May 2017, pp. 564–571.

[Per+11] Diego Perez et al. “Evolving Behaviour Trees for the Mario AI Competition Using Grammatical Evolution”. In: *Applications of Evolutionary Computation*. Ed. by Cecilia Di Chio et al. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2011, pp. 123–132.

[PN17] Athanasios S. Polydoros and Lazaros Nalpantidis. “Survey of Model-Based Reinforcement Learning: Applications on Robotics”. In: *Journal of Intelligent & Robotic Systems* 86.2 (May 2017), pp. 153–173.

[RGK17] F. Rovida, B. Grossmann and V. Krüger. “Extended Behavior Trees for Quick Definition of Flexible Robotic Tasks”. In: *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Sept. 2017, pp. 6793–6800.

[Rov+18] F. Rovida et al. “Motion Generators Combined with Behavior Trees: A Novel Approach to Skill Modelling”. In: *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Oct. 2018, pp. 5964–5971.

[SS13] Freek Stulp and Olivier Sigaud. “Robot Skill Learning: From Reinforcement Learning to Evolution Strategies”. In: *Paladyn, Journal of Behavioral Robotics* 4.1 (Sept. 2013), pp. 49–61.

[Tob+17] J. Tobin et al. “Domain Randomization for Transferring Deep Neural Networks from Simulation to the Real World”. In: *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Sept. 2017, pp. 23–30.

[Top+18] Elin A. Topp et al. “Ontology-Based Knowledge Representation for Increased Skill Reusability in Industrial Robots”. In: *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Oct. 2018, pp. 5672–5678.

[Ude+10] A. Ude et al. “Task-Specific Generalization of Discrete and Periodic Dynamic Movement Primitives”. In: *IEEE Transactions on Robotics* 26.5 (Oct. 2010), pp. 800–815.

Paper IV

Paper 4

Skill-based Multi-objective Reinforcement Learning of Industrial Robot Tasks with Planning and Knowledge Integration

Matthias Mayr
Lund University
matthias.mayr@cs.lth.se

Faseeh Ahmad
Lund University
faseeh.ahmad@cs.lth.se

Konstantinos Chatzilygeroudis
University of Patras, Greece
costashatz@upatras.gr

Luigi Nardi
Lund University
Stanford University, USA
lnardi@stanford.edu

Volker Krueger
Lund University
volker.krueger@cs.lth.se

Abstract

In modern industrial settings with small batch sizes it should be easy to set up a robot system for a new task. Strategies exist, e.g. the use of skills, but when it comes to handling forces and torques, these systems often fall short. We introduce an approach that provides a combination of task-level planning with targeted learning of scenario-specific parameters for skill-based systems. We propose the following pipeline: the user provides a task goal in the planning language *PDDL*, then a plan (i.e., a sequence of skills) is generated and the learnable parameters of the skills are automatically identified, and, finally, an operator chooses reward functions and parameters for the learning process. Two aspects of our methodology are critical: (a) learning is tightly integrated with a knowledge framework to support symbolic planning and to provide priors for learning, (b) using multi-objective optimization. This can help to balance key performance indicators (KPIs) such as safety and task performance since they can often affect each other. We adopt a multi-objective Bayesian optimization approach and learn entirely in simulation. We demonstrate the efficacy and versatility of our approach by learning skill parameters for two different contact-rich tasks. We show their successful execution on a real 7-DOF *KUKA-iiwa* manipulator and outperform the manual parameterization by human robot operators.

1 Introduction

Industrial environments with expensive and fragile equipment, safety regulations and frequently changing tasks often have special requirements for the behaviour policies that control a robot: First, the trend in industrial manufacturing is to move to smaller batch sizes and higher flexibility of work stations. Reconfiguration needs to be fast, easy and should minimize downtime. Second, it is important to be able to guarantee the performance as well as safety for material and workers. Therefore, it is crucial to be able to understand *what* action is performed *when* and *why*. Finally, in industrial environments digital twins provide a lot of task-relevant information such as material properties and approximate part locations that the robot behavior policies have to consider.

One way to fulfill these criteria is to use systems based on parameterized *skills* [Kru+19; Kru+16; Bøg+12]. These encapsulated abilities realize semantically defined actions such as moving the robot arm, opening a gripper or localizing an object with vision. State-of-the-art skill-based software architectures can not only utilize knowledge, but also automatically generate plans (skill-sequences) for a given task [Cro+17; Rov+16]. The skill-based approach is powerful when knowledge can be modeled and formalized *explicitly* [Kru+19; Kru+16]. But it is often limited when it comes to skill parameters of contact-rich tasks that are difficult to reason about. One example are the parameters of a peg insertion search strategy where material properties (e.g. friction) and the robot controller performance need to be considered. While it is possible to create a reasoner that follows a set of rules to determine such skill parameters, it is often challenging to implement and to maintain.

Another way to handle this is to have operators manually specify and try values for these skill parameters. However, this is a manual process and can be cumbersome.

Finally, it is possible to allow the system to learn by interacting with the environment. However, many policy formulations that allow learning (e.g. artificial neural networks) have deficiencies which make their application in an industrial domain with the abovementioned requirements challenging. Primarily during the learning phase, dangerous behaviors can be produced and even state-of-the-art RL methods need hundreds of hours of interaction time [Cha+19]. Learning in simulation can help to reduce downtime and dangers for the real system. But many policy formulations are black boxes for operators and it can be hard to predict their behavior, which could hinder the trust to the system [Edm+19] Moreover, the simulation-to-reality gap [KMD13; MC17] is bigger in lower-level

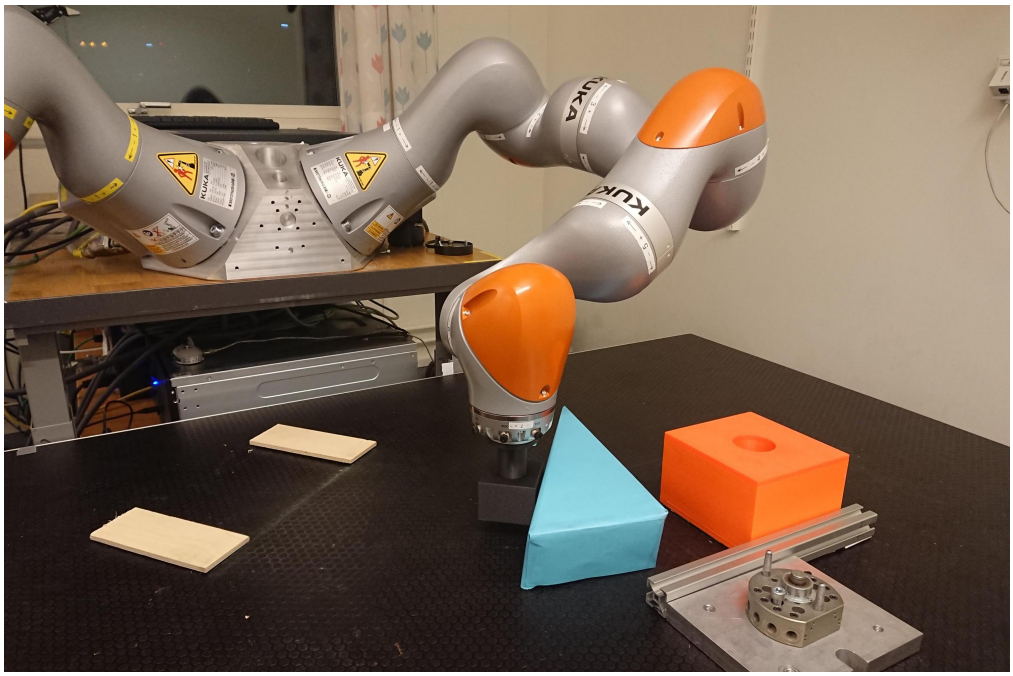


Figure 4.1: The robot setup used for the experiments. Wooden boards indicate the start location for the push task. The goal is the corner between the fixture and the box with the hole for the peg task.

control states (i.e. torques), and policies working directly on raw control states struggle to transfer learned behaviors to the real systems [Cha+19]. Our policy formulation consisting of behavior trees (BT) with a motion generator [Rov+18] has shown to be able to learn interpretable and robust behaviors [May+21].

The formulation of a learning problem for a given task is often not easy and becomes more challenging if factors such as safety or impact on the workstation environment need to be considered. Multi-objective optimization techniques allow to specify multiple objectives and optimize for them concurrently. This allows operators to select from solutions that are optimal for a certain trade-off between the objectives (usually represented as a set of Pareto-optimal solutions). In order to learn sample-efficient and to support the large variety of skill implementations as well as scenarios, we use gradient-free Bayesian optimization as an optimization method.

In this paper we make the following contributions:

1. We introduce a new method which seamlessly integrates symbolic planning and reinforcement learning for skill-based systems to learn interpretable policies for a given task.

2. A Bayesian multi-objective treatment of the task learning problem, which includes the operator through easy specification of problem constraints and task objectives (KPIs); the set of Pareto-optimal solutions is presented to the operator and their behavior can be inspected in simulation and executed on the real system.

3. We demonstrate our approach on two contact-rich tasks, a pushing task and a peg-in-hole task. We compare it to the outcome of the planner without reasoning, randomly sampled parameter sets from the search space and the manual real-world parameterization process of robot operators. In both tasks our approach delivered solutions that even outperform the ones found by the manual search of human robot operators.

2 Related Work

2.1 Skill-based Systems

Skill-based systems are one way to support a quick setup of a robot system for a new task and to allow re-use of capabilities. There are multiple definitions of the term *skills* in the literature. Some define it as a pure *motion skills* [HST91] or "hybrid motions or tool operations" [Tho+03]. Other work has a broader skill definition [Kru+19; Kru+16; Bøg+12; Cro+17; Rov+16; Rov+17; Tho+13]. In this formulation, skills can be arbitrary capabilities that change the state of the world and have pre- and postconditions. Their implementation can include motion skills, but also proficiencies such as vision-based localization of objects. In [Ste+15] skills are "high-level reusable robot capabilities, with the goal to reduce the complexity and time consumption of robot programming". However, compared to [Bøg+12] and [Rov+17] they do not use pre- and postconditions. In [Pax+17], an integrated system for manual creation of *task plans* is presented and shares the usage of BTs with our approach.

Task planners are used in [Kru+19; Kru+16; Cro+17; Rov+16; Rov+17; RGK17; Ste+15; Tho+03] while [Pax+17] lacks such a capability.

In [Ste+15] it is suggested that "Machine learning can be performed on the motion level, in terms of adaptation, or can take the form of structured learning

on a task/error specification level". However, none of the reviewed work offers a combination task-level planning with learning.

2.2 Policy Representation and Learning

An important decision to make when working with manipulators is the type of policy representation and on which level it interfaces with the robot. The latter can strongly influence the learning speed and the quality of the obtained solutions [VGK19; Mar+19]. These choices also influence the form of priors that can be defined and how they are defined [Cha+19]. Not many policies combine the aforementioned properties of being a) interpretable, b) parameterizable for the task at hand and c) allow learning or improvement.

The commonly used policy representations for learning systems include radial basis function networks [DNP13], dynamic movement primitives [Ijs+12; Ude+10] and feed-forward neural networks [DNP13; Cha+17]. In recent years deep artificial neural networks (ANN) seem to become a popular policy. All of them have in common that their final representation can be difficult to interpret. Even if a policy only sets a target pose for the robot to reach, it can be problematic to know how it reacts in all parts of the state space. In contrast to that, [May+21] suggests to learn interpretable policies based on behavior trees [Rov+18]. They work explicitly in end-effector space and allow for an easy formulation of parameter priors to accelerate learning [May+22b].

2.3 Planning and Learning

Symbolic planning is combined with learning in [GK08; GFF19; Yan+18; Sar+21]. In [GK08], the PLANQ-Learning algorithm uses a symbolic planner to shape the reward function based on the conditions defined which are then used by the Q-learner to get an optimal policy with good results on the grid domain. [GFF19] uses the combined symbolic planner with reinforcement learning (RL) in a hierarchical framework to solve complex visual interactive question answering tasks. PEORL [Yan+18] integrates symbolic planning and hierarchical reinforcement learning (HRL) to improve performance by achieving rapid policy search and robust symbolic planning in the taxi domain and grid world. SPOTTER [Sar+21] uses RL to allow the planning agent to discover the new operators required to complete tasks in Grid World. In contrast to all these approaches, our approach aims towards real-life robotic tasks in an Industry 4.0 setting where a digital twin is available.

In [Sty+22], the authors combine symbolic planning with behavior trees (BT) to solve blocks world tasks with a robot manipulator. They use modified Genetic Programming (GP) [KK92] to learn the structure of the BT. In our approach, we focus on learning the parameters of the skills in the BT and utilize a symbolic planner to obtain the structure of the BT.

3 Approach

Our approach consists of two main components that interact in different stages of the learning pipeline: First, *SkiROS* [Rov+17], a skill-based framework for ROS, which represents the implemented skills with BTs, hosts the world model (digital twin), and interacts with the planner. *SkiROS* is also used to execute BTs while learning and to perform tasks on the real system. Second, the learning framework that provides the simulation, the integration with the policy optimizer as well as the reward function definition and calculation. The architecture of the system and the workflow is shown in Figure 4.2: (1) an operator enters the task goal into a GUI; (2) a plan with the respective learning scenario configuration is generated; (3) an operator complements the scenario with objectives and reward functions; (4) learning is conducted in simulation using the skills and information from the world model; (5) in the multi-objective optimization case, a set of Pareto-optimal solutions is generated and presented to the operator; finally, (6) the operator can select a good solution from this set given the desired trade-off between KPIs and execute it on the real system.

3.1 Behavior Trees

A Behavior Tree (BT) [CÖ17a]is a formalism for plan representation and execution. Like [RGK17; Mar+14], we define it as a directed acyclic graph $G(V, W)$ with $|V|$ nodes and $|W|$ edges. It consists of *control flow nodes* (*processors*), and *execution nodes*. The four basic types of *control flow nodes* are 1) *sequence*, 2) *selector*, 3) *parallel* and 4) *decorator* [Mar+14]. A BT always has one initial node with no parents, defined as *Root*, and one or more nodes with no children, called *leaves*. When executing a BT, the *Root* node periodically injects a *tick* signal into the tree. The signal is routed through the branches according to the implementation of the *control flow nodes* and the return statements of their children. By convention, the signal propagation goes from left to right.

The *sequence* node corresponds to a logical *AND*: it succeeds if all children succeed and fails if one child fails. The *selector*, also called *fallback* node, represents

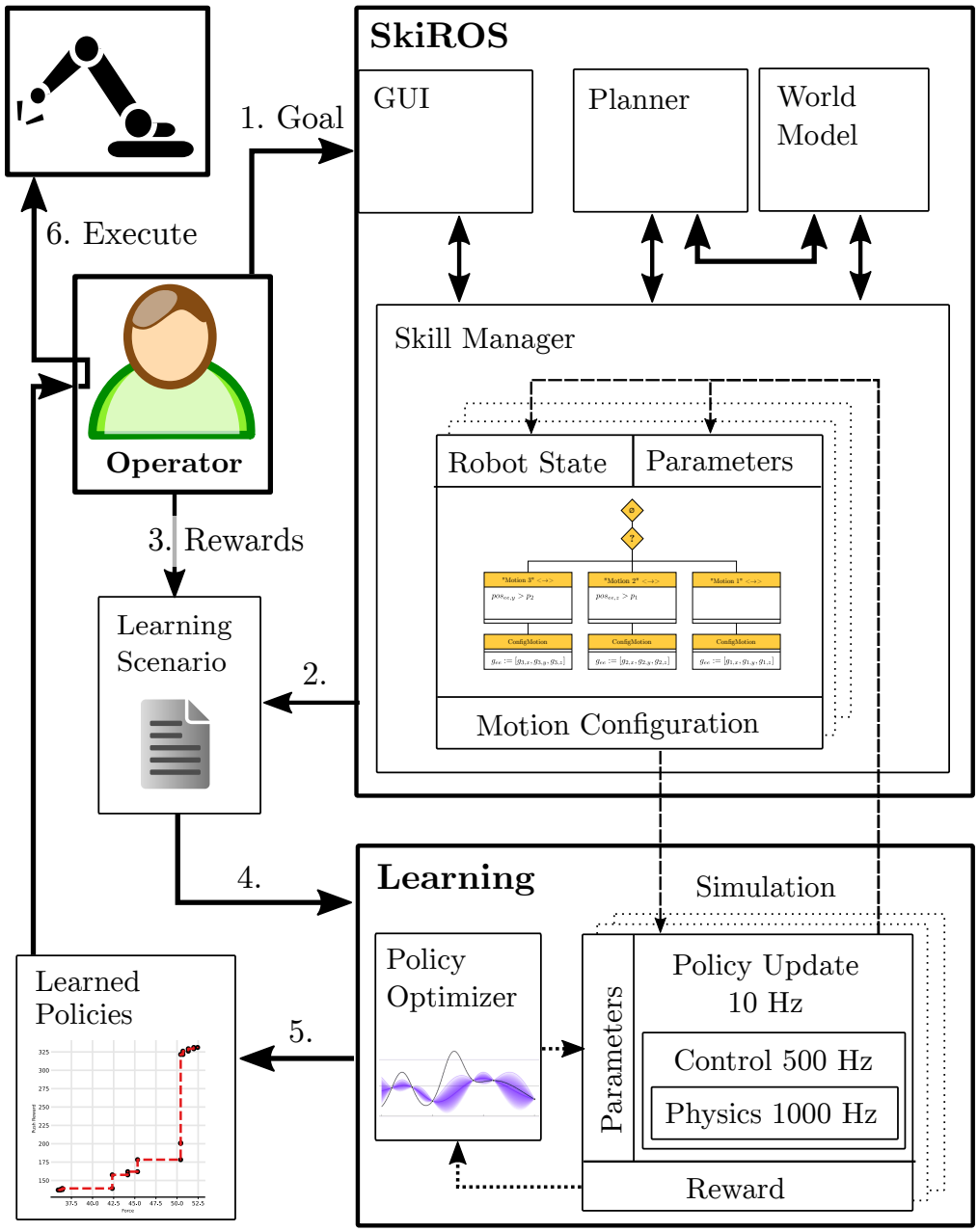


Figure 4.2: The architecture of the system that depicts the pipeline: (1) The operator enters the goal state; (2) a learning scenario for the plan is created; (3) rewards and hyperparameters are specified; (4) learning is conducted using the skills and the information in the world model; (5) after policy learning, the operator can choose which policies to execute on the real system (6).

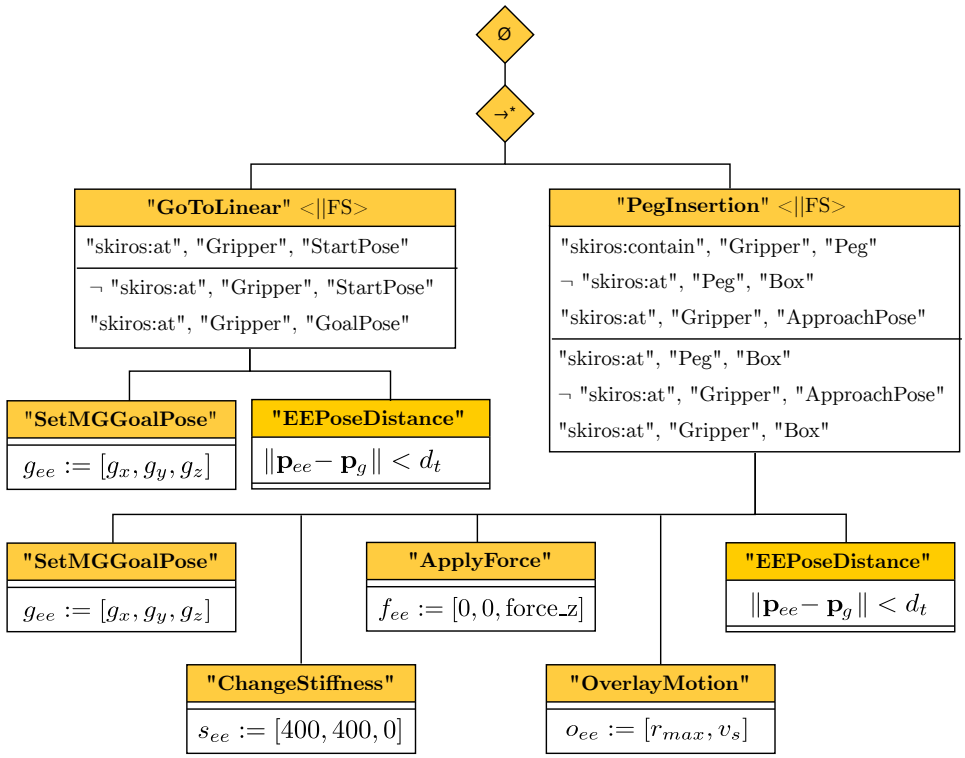


Figure 4.3: The BT of the generated plan for the peg insertion task in *eBT format* [RGK17]. Each node has conditions or pre-conditions shown in the upper half and effects or post-conditions shown in the lower half. The *serial start control flow node* ($\rightarrow^*$) executes in a sequence and remembers the successes. The skills have a *parallel-first-success* processor ($< ||FS >$).

a logical *OR*: If one child succeeds, the remaining ones will not be ticked. It fails only if all children fail. The *parallel* control flow node forwards ticks to all children and fails if one fails. A *decorator* allows to define custom functions. Implementations like *extended Behavior Trees* (eBT) in *SkiROS* [RGK17] add custom processors such as *parallel-first-success* that succeeds if one of the parallel running children succeeds. Leaves of the BT are the *execution nodes* that, when ticked, execute one cycle and output one of the three signals: *success*, *failure* or *running*. In particular, execution nodes subdivide into 1) *action* and 2) *condition* nodes. An action performs its operation iteratively at every tick, returning *running* while it is not done, and *success* or *failure* otherwise. A condition performs an instantaneous operation and returns always *success* or *failure* and never *running*. An example of the BT for the peg insertion task is in Fig. 4.3.

3.2 Planning and Knowledge Integration

The Planning Domain Definition Language (PDDL) [FL03; Cro+17] is used to formulate the planning problem. We use the *SkiROS* [Rov+17] framework that automatically translates a task into a PDDL planning problem by generating domain description and problem instance using the world model. We then use the semantic world model (WM) from *SkiROS* [Rov+17] as the knowledge integration framework.

Actions and fluents are obtained by utilizing the predicates that have pre- or post-conditions in the world model. For the problem instance, the objects (robots, arms, grippers, boxes, poses, etc.) in the scene and their initial states (as far as they are known) are used. After getting the necessary domain description and the problem instance, *SkiROS* calls the planner. The goal of the planner is to return a sequence of skills that can achieve the goal conditions of the task. The individual skills are partially parameterized with explicit data from the WM. The WM is aware of the skill parameters that need to be learned for the task at hand and they are automatically identified in the skill sequence.

3.3 Policy Optimization

In order to optimize for policy parameters, we adopt the policy search formulation [DNP13; Cha+19; Cha+17]. We formulate a dynamical system in the form:

$$\mathbf{x}_{t+1} = \mathbf{x}_t + M(\mathbf{x}_t, \mathbf{u}_t, \boldsymbol{\phi}_R), \tag{4.1}$$

with continuous-valued states $\mathbf{x} \in \mathbb{R}^E$ and actions $\mathbf{u} \in \mathbb{R}^U$. The transition dynamics are modeled by a simulation of the robot and the environment $M(\mathbf{x}_t, \mathbf{u}_t, \boldsymbol{\phi}_R)$. They are influenced by the domain randomization parameters $\boldsymbol{\phi}_R$.

The goal is to find a policy $\pi, \mathbf{u} = \pi(\mathbf{x}|\boldsymbol{\theta})$ with policy parameters $\boldsymbol{\theta}$ such that we maximize the expected long-term reward when executing the policy for T time steps:

$$J(\boldsymbol{\theta}) = \mathbb{E}\left[\sum_{t=1}^{T} r(\mathbf{x}_t, \mathbf{u}_t)|\boldsymbol{\theta}\right], \tag{4.2}$$

where $r(\mathbf{x}_t, \mathbf{u}_t)$ is the immediate reward for being in state $\mathbf{x}$ and executing action $\mathbf{u}$ at time step t. The discrete switching of branches in the BT and most skills are not differentiable. Therefore, we frame the optimization in Eq. (4.2) as a black-box optimization and pursue the maximization of the reward function $J(\theta)$ only by using measurements of the function. The optimal reward function

to solve the task is generally unknown, and a combination of reward functions is usually used. In the RL literature, this is usually done with a weighted average, that is, $r(\mathbf{x}_t, \mathbf{u}_t) = \sum_i w_i r_i(\mathbf{x}_t, \mathbf{u}_t)$. In this paper, we chose not to use a weighted average of reward functions that represent different objectives (as the optimal combination of weights cannot always be found [KCM18]), but optimize for all objectives concurrently (Sec. 3.5) using Bayesian Optimization.

3.4 Bayesian Optimization

We consider the problem of finding a global minimizer (or maximizer) of an unknown (black-box) objective function f: $\mathbf{s}^* \in \arg\min_{\mathbf{s}\in\mathbb{S}} f(\mathbf{s})$, where $\mathbb{S}$ is some input design space of interest in D dimensions. The problem addressed in this paper is the optimization of a (possibly noisy) function $f : \mathbb{S} \to \mathbb{R}$ with lower and upper bounds on the problem variables. The variables defining $\mathbb{S}$ can be real (continuous), integer, ordinal, and categorical as in [NKO19]. We assume that the function f is in general expensive to evaluate and that the derivatives of f are in general not available. The function f is called black box because we cannot access other information than the output y given an input value $\mathbf{s}$.

This problem can be tackled using Bayesian Optimization (BO) [Fra18]. BO approximates $\mathbf{s}^*$ with a sequence of evaluations, $y_1, y_2, \ldots, y_t$ at $\mathbf{s}_1, \mathbf{s}_2, \ldots, \mathbf{s}_t \in \mathbb{S}$, which maximizes an utility metric, with each new $\mathbf{s}_{t+1}$ depending on the previous function values. BO achieves this by building a probabilistic surrogate model on f based on the set of evaluated points $\{(\mathbf{s}_i, y_i)\}_{i=1}^t$. At each iteration, a new point is selected and evaluated based on the surrogate model which is then updated to include the new point $(\mathbf{s}_{t+1}, y_{t+1})$.

BO defines an utility metric called the acquisition function, which gives a score to each $\mathbf{s} \in \mathbb{S}$ by balancing the predicted value and the uncertainty of the prediction for $\mathbf{s}$. The maximization of the acquisition function guides the sequential decision making process and the exploration versus exploitation trade-off: the highest score identifies the next point $\mathbf{s}_{t+1}$ to evaluate.

BO is a statistically efficient black-box optimization approach when considering the number of necessary function evaluations [BCd10]. It is, thus, especially well-suited to solve problems where we can only perform a limited number of function evaluations, such as the ones found in robotics.

We use the implementation of BO found in *HyperMapper* [NKO19; Nar+17; Bod+16; Sou+21]. Our implementation selects the Expected Improvement (EI) acquisition function [MTZ78] and we use uniform random samples as a warm-up

strategy before starting the optimization.

3.5 Multi-objective Optimization

Let us consider a multiple objectives minimization (or maximization) over $\mathbb{S}$ in D dimensions. We define $f : \mathbb{S} \to \mathbb{R}^p$ as our vector of objective functions $f = (f_1, \ldots, f_p)$, taking $\mathbf{s}$ as input, and evaluating $y = f(\mathbf{s}) + \epsilon$, where ϵ is a Gaussian noise term. Our goal is to identify the Pareto frontier of f, that is, the set $\Gamma \subseteq \mathbb{S}$ of points which are not dominated by any other point, *i.e.,* the maximally desirable $\mathbf{s}$ which cannot be optimized further for any single objective without making a trade-off. Formally, we consider the partial order in $\mathbb{R}^p$: $y \prec y'$ iff $\forall i \in [p], y_i \leqslant y'_i$ and $\exists j,\, y_j < y'_j$, and define the induced order on $\mathbb{S}$: $\mathbf{s} \prec \mathbf{s}'$ iff $f(\mathbf{s}) \prec f(\mathbf{s}')$. The set of minimal points in this order is the Pareto-optimal set $\Gamma = \{\mathbf{s} \in \mathbb{S} : \nexists \mathbf{s}' \text{ such that } \mathbf{s}' \prec \mathbf{s}\}$. We aim to identify Γ with the fewest possible function evaluations using BO. For this purpose we use the *HyperMapper* multi-objective Bayesian optimization which is based on random scalarizations [PKP20].

3.6 Motion Generator and Robot Control

The arm motions are controlled in end-effector space by a Cartesian impedance controller. The time varying *reference* or *attractor point* of the end effector $\mathbf{x}_d$ is governed by a motion generator (MG). Given the joint configuration $\mathbf{q}$, we can calculate the end-effector pose $\mathbf{x}_{ee}$ using forward kinematics and obtain an error term $\mathbf{x}_e = \mathbf{x}_{ee} - \mathbf{x}_d$. Together with the joint velocities $\dot{\mathbf{q}}$, the Jacobian $\mathbf{J}(\mathbf{q})$, the configurable stiffness and damping matrices K_d and D_d, the task control is formulated as $\tau_c = \mathbf{J}^T(\mathbf{q})\left(-\mathbf{K}_d\mathbf{x}_e - \mathbf{D}_d\mathbf{J}(\mathbf{q})\dot{\mathbf{q}}\right)$. Additionally, the task control can be overlayed with commanded generalized forces and torques $\mathbf{F}_{ext} = (f_x\ f_y\ f_z\ \tau_x\ \tau_y\ \tau_z)$: $\tau_{ext} = \mathbf{J}^T(\mathbf{q})\mathbf{F}_{ext}$. We utilize the integration introduced in [Rov+18] and used in [May+21], which proposes to parameterize the MG with movement skills from the BT. The reference pose is shaped by 1) a linear trajectory to a goal point and 2) overlay motions that can be added to the reference pose as discussed in [Rov+18; May+21]. E.g. an Archimedes spiral for search.

To make it compliant with the dynamical system in Eq. (4.1), a new reference configuration of the controller is only generated at every time step t. It includes the reference pose, stiffnesses, applied wrench and forms the action $\mathbf{u}$ with a dimension of $U = 19$. The stiffness and applied force are changed gradually at

every time step t to ensure a smooth motion. The state space consists of joint positions and joint velocities and is $E = 14$ dimensional. Direct control of the torques of a robot arm requires high update rates and we control the robot arm at 500 Hz based on the current action $\mathbf{u}$, but continuously updated values for $\mathbf{q}$ and $\dot{\mathbf{q}}$. Therefore, from the perspective of Eq. (4.1), the controller is to be seen as part of the model $M(\mathbf{x}_t, \mathbf{u}_t)$.

We assume a human-robot collaborative workspace with fragile objects. Therefore, the stiffnesses and applied forces are to be kept to a minimum and less accuracy than e.g. high-gain position-controlled solutions is to be expected.

4 Experiments

In our experiments we use a set of pre-defined skills that are part of a skill library. In order to solve a task, the planner determines a sequence that can achieve the goal condition of the task. This skill sequence is also automatically parameterized to the extend possible, e.g. the goal pose of a movement. We evaluate our system in two contact-rich scenarios that are shown in Fig. 4.1: A) pushing an object with uneven weight distribution to a goal pose and B) inserting a peg in a hole with a 1.5 mm larger radius. Pure planning-based solutions for both these tasks have a poor performance in reality (Fig. 4.5).

As a baseline we invited six robot operators to manually parameterize the skills for the tasks. Their main objective is to find a parameter set that robustly solves the task. As an additional objective they were asked to minimize the impact of the robot arm and its tool on the environment as long as it does not affect the first objective.

The robot arm used for the physical evaluation is a 7-degree-of-freedom (DOF) *KUKA iiwa* arm controlled by a Cartesian impedance controller (Sec. 3.6).

4.1 Reward Functions

For each task, we utilize a set of reward functions parameterized for the learning scenario configuration. Each configured reward has an assigned objective and can be weighted against other rewards. Each experiment uses a subset of the following reward functions:

Task completion

A fixed reward is assigned when the BT returns success upon task completion.

End-effector distance to a box

We use a localized reward to attract the end effector towards the goal location $r_h(\mathbf{x}) = \left(2\left(d(\mathbf{p}_{ee,\mathbf{x}}, \mathbf{p}_h) + d_o\right)\right)^{-1}$, where d_o is the distance offset and $d(\mathbf{p}_{ee,\mathbf{x}}, \mathbf{p}_h)$ is the shortest distance function between the end effector and the box.

Applied wrench

This reward calculates the cumulative forces applied by the end effector on the environment.

Reward functions 4-6 share a common operation of computing an exponential function of the calculated metric to obtain the reward as used in ([DR11b; Cha+17]) $r(d_m) = \exp\left(-\frac{1}{2\sigma_w^2}(d_m + d_o)\right)$, where σ_w is a configurable width, d_o is a distance offset and d_m is the input metric.

End-effector distance to a goal

This reward uses distance between the end effectors current pose and goal pose to calculate the input metric $d_{ee,g} = \|\mathbf{p}_{ee,\mathbf{x}} - \mathbf{p}_g\|$

End-effector-reference-position distance

This reward uses the distance between the end effectors reference pose (Sec. 3.6) and its current pose to calculate the input metric $d_{ee,d} = \|\mathbf{p}_{ee,\mathbf{x}} - \mathbf{x_d}\|$

Object-pose divergence

This reward uses the translational and angular distance between the object's goal pose and its current pose.

4.2 Push Task

The push task starts by specifying the goal in the *SkiROS* Graphical User Interface (GUI) as: `(skiros:at skiros:ObjectToBePushed-1 skiros:Object-GoalPose-1)`. *SkiROS* calls the planner to generate a plan given all the available skills. The plan consists of two skills: 1) *GoToLinear* skill and 2) *Push* skill. The first skill moves the end effector from its current location to the *approach pose* of the object. This *approach pose* is defined in the WM and needs to be reached before interacting with the object.

The push skill then moves the end effector to the object's geometric centre with an optional offset in the horizontal (x) and (y) directions. Once the end effector reaches it, the motion generator executes a straight line to the (modified) target location.

The push task is formulated as a multi-objective task. It also has two objectives, 1) success and 2) applied force. The first objective has three associated rewards: 1) object position difference from goal position, 2) object orientation difference from goal orientation, and 3) end-effector distance to the goal location. The second objective accumulates the Cartesian distance between the end-effector reference pose and the actual end-effector pose as a measure of the force applied by the controller. The learnable parameters in this task are offsets in the horizontal (x) and (y) direction of both the push skill's start and goal locations. An offset of the start location allows the robot to push from a particular point from the side of the object. Together with the offsets on the goal position, these learnable parameters collectively define the trajectory of the push.

The object to be pushed has a height of 0.07 m and is an orthogonal triangle in the horizontal dimensions (x) and (y). It has a length of 0.15 m and 0.3 m and it weights 2.5 kg. For this task we use a square-shaped peg for pushing with a side length of 0.07 m and a height of 0.05 m. Start and goal locations are $\approx$0.43 m apart and are rotated by 26 *deg*. We define success if the translational and rotational difference of the object w.r.t the goal is less than 0.01 m and 5 *deg*, respectively.

We learn for 400 iterations and repeat the experiment ten times. In order to obtain solutions that are robust enough to translate to the real system, we apply domain randomization. Each parameter set is evaluated in seven worlds. Each execution uniformly samples one out of the four start positions for the robot arm. Furthermore, we vary the location of the object and the goal in the horizontal (x) and (y) directions by sampling from a Gaussian distribution with a standard deviation of 7 mm.

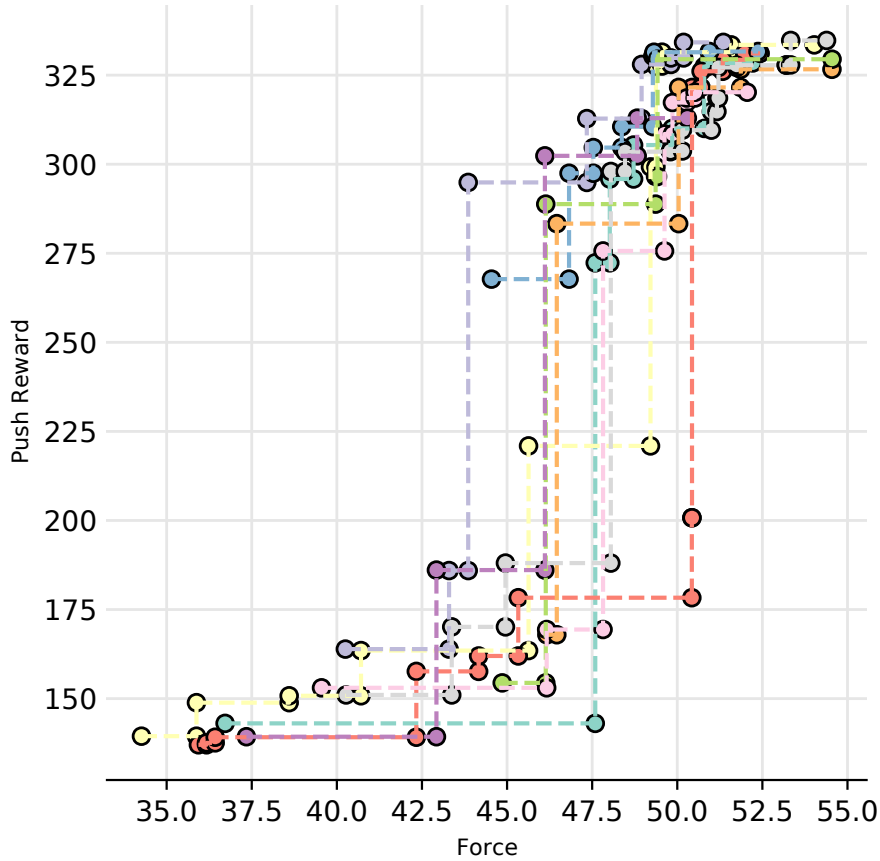


Figure 4.4: Pareto front of the push task. Each experiment has a different color and each point represents a Pareto-optimal solution. It shows that higher rewards for pushing require higher interaction forces with the environment.

We compare the learned solutions with (a) the outcome of a direct planner solution without any offset on the start and goal pose while pushing, (b) ten sets of random parameters from the search space and (c) the policies that are parameterized by the robot operators. We evaluated on the four start configurations used for learning as well as on two additional unknown ones. The results are shown in Fig. 4.5a.

The results of a multi-objective optimization are parameters found along a Pareto front (Sec. 3.5, see Fig. 4.4). It contained 8.3 points on average, of which some minimize the impact on the environment to an extent that the push is not successful. An operator can choose a solution that is a good compromise

between the success of the task on the real system and the force applied on the environment. The performance of one of the solutions that existed on the Pareto front is shown in Fig. 4.5.

Furthermore, we asked six robot operators to find values for the learnable parameters of the skill sequences. They were given the same start positions used for learning and were given a script to reset the arm to a start position of their choice. They could experiment with the system until they decided that their parameter set fulfills the criteria. Their final parameter set that was also evaluated on the known and unknown start configurations. On average the operators spent (16.3 ± 6.4) min and executed 11.1 ± 3.0 trials on the system to configure this task. Four out of the six operators found solutions that achieved the task from every start state. However, two of the operators' final parameters only achieved success rates of 50 % and 16.66 %.

4.3 Peg-in-Hole Task

The PDDL goal of the peg insertion task is `(skiros:at skiros:Peg-1 skiros:BoxWithHole-1)`. The BT that is generated by the planner is shown in Fig. 4.3 and consists of two skills: 1) *GoToLinear* skill and 2) *PegInsertion* skill. The first skill moves the end effector from its current location to the *approach pose* of the hole. Once it is reached, the peg insertion procedure starts.

The *PegInsertion* skill starts when the end effector reaches the approach pose of the box. It uses four separate *SkiROS* primitive skills to 1) set the stiffness of the end effector to zero in (z) direction, 2) apply a downward force in (z) direction, 3) configure the center of the box as a goal and 4) additionally apply an overlaying circular search motion on top of the reference pose of the end effector as described in [May+21]. The BT returns success only if the peg is inserted into the box hole by more than 0.01 m.

We model the peg insertion as a multi-objective and multi-reward task. There are two objectives of the task, 1) successful insertion and 2) applied force. To assess the efficacy of the first objective, we use three rewards, 1) success of the BT, 2) peg distance to the hole, and 3) peg distance to the box. For the second objective, we use a single reward that measures the total force applied by the peg. There are three learnable parameters in this task, 1) downward force applied by the robot arm, 2) radius of the overlay search motion and 3) path velocity of the overlay search motion.

We learn for 400 iterations in the simulation and repeat this experiment ten times. To increase the robustness of the solutions we use domain randomization and evaluate each parameter configuration in seven worlds. We vary the location of the box by sampling from a Gaussian distribution with a standard deviation of 7 mm and uniformly sample one out of five start configurations of the robot arm. We compare the performance of the learned policies with (1) the outcome of the planner without a parameterized search motion, (2) randomly chosen parameter configurations from the parameter search space used for learning and (3) policies that are parameterized by human operators (see Fig. 4.5b).

The learned Pareto-optimal configurations consist of 6.1 points on average. We evaluated the insertion success using the 5 known and additional 10 unknown start configurations of the robot (Fig. 4.5b).

To find policies for this task, the human operators took (31.8 ± 10.9) min and executed 39 ± 14 trials on the system. However, compared to the randomly sampled policies the average insertion rate only increased from 41 % to 52.2 %. This is much lower than the average insertion rate of 96 % of the best learned policies as shown in box four, Fig. 4.5b. Furthermore, the average force that was chosen by the operators compared to the learned policies was 16.6 % higher. Finally, the successful insertions by the learned policies were also 18.1 % faster. Therefore, the learned policies outperformed the human operators in both objectives while also producing more reliable results.

5 Conclusion

In this paper we proposed a method for effectively combining task-level planning with learning to solve industrial contact-rich tasks. Our method leverages prior information and planning to acquire *explicit* knowledge about the task, whereas it utilizes learning to capture the *tacit* knowledge, i.e., the knowledge that is hard to formalize and which can only be captured through actual interaction.We utilize behavior trees as an interpretable policy representation that is suitable for learning and leverage domain randomization for learning in simulation. Finally, we formulate a multi-objective optimization scheme so that (1) we handle conflicting rewards adequately, and (2) an operator can choose a policy from the Pareto front and thus actively participate in the learning process.

We evaluated our method on two scenarios using a real *KUKA* 7-DOF manipulator: (a) a pushing task, and (b) a peg insertion task. Both tasks are contact-rich and naïve planning fails to solve them. The approach was able

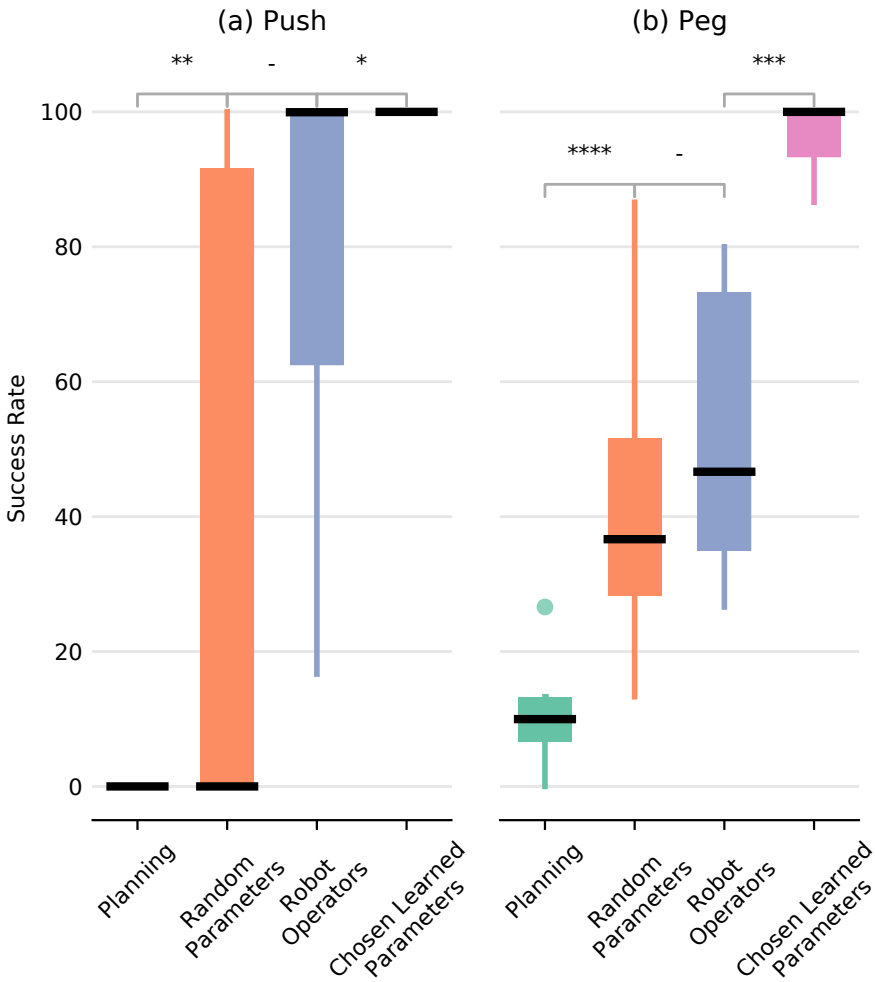


Figure 4.5: The success rates of both experiments. The box plots show the median (black line) and interquartile range (25^{th} and 75^{th} percentile); the lines extend to the most extreme data points not considered outliers, and outliers are plotted individually. The number of stars indicates that the p-value of the Mann-Whitney U test is less than 0.1, 0.05, 0.01 and 0.001 respectively.

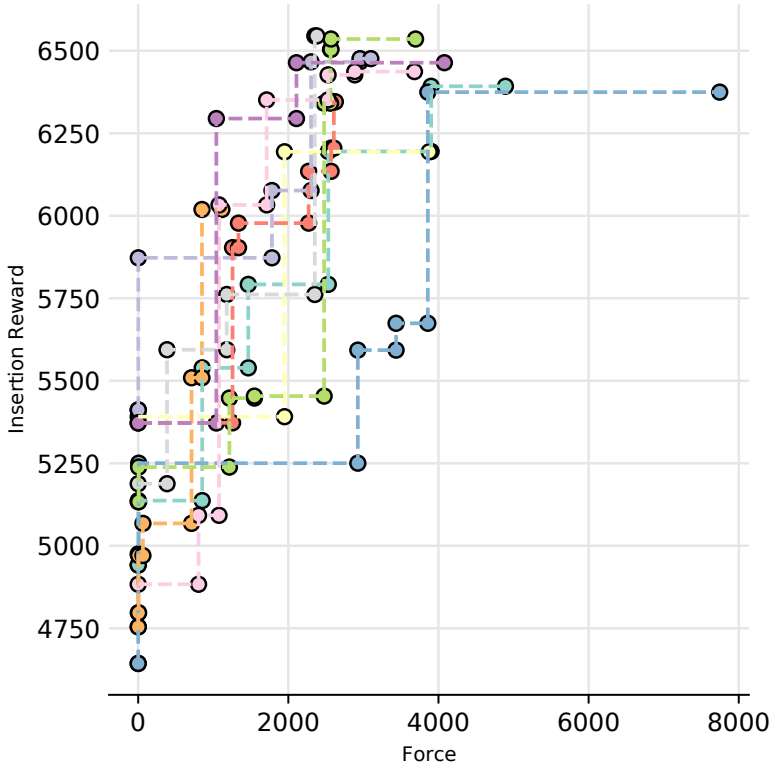


Figure 4.6: Pareto front of the peg task. Each experiment has a different color. The goal is to maximize insertion reward while minimizing the interaction forces.

to outperform the baselines including the manual parameterization by robot operators.

For future work we are looking into multi-fidelity learning that can leverage a small amount of executions on the real system to complement the learning in simulation. Furthermore, the use of parameter priors for the optimum seems a promising direction to guide the policy search and make it more efficient.

Appendix

The implementation and the supplemental video are available at:
`https://sites.google.com/ulund.org/SkiREIL`
`https://github.com/matthias-mayr/SkiREIL`

Acknowledgement

We thank Alexander Durr, Elin Anna Topp, Francesco Rovida and Jacek Malec for the interesting discussions and the constructive feedback.

This work was partially supported by the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by Knut and Alice Wallenberg Foundation. This research was also supported in part by affiliate members and other supporters of the Stanford DAWN project—Ant Financial, Facebook, Google, InfoSys, Teradata, NEC, and VMware.

References

[Bod+16] Bruno Bodin et al. "Integrating Algorithmic Parameters into Benchmarking and Design Space Exploration in 3D Scene Understanding". In: *International Conference on Parallel Architectures and Compilation*. 2016.

[Bøg+12] Simon Bøgh et al. "Does Your Robot Have Skills?" In: *Proceedings of the 43rd International Symposium on Robotics*. VDE Verlag GmbH, 2012.

[BCd10] Eric Brochu, Vlad M. Cora and Nando de Freitas. *A Tutorial on Bayesian Optimization of Expensive Cost Functions, with Application to Active User Modeling and Hierarchical Reinforcement Learning*. Dec. 2010. arXiv: `1012.2599 [cs]`.

[Cha+17] Konstantinos Chatzilygeroudis et al. "Black-Box Data-Efficient Policy Search for Robotics". In: *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Sept. 2017, pp. 51–58.

[Cha+19] Konstantinos Chatzilygeroudis et al. "A Survey on Policy Search Algorithms for Learning Robot Controllers in a Handful of Trials". In: *IEEE Transactions on Robotics* (2019), pp. 1–20.

[CÖ17a] Michele Colledanchise and Petter Ögren. *Behavior Trees in Robotics and AI: An Introduction*. Chapman \& Hall/CRC Press, Aug. 2017.

[Cro+17] Matthew Crosby et al. “Integrating Mission and Task Planning in an Industrial Robotics Framework”. In: *Twenty-Seventh International Conference on Automated Planning and Scheduling*. June 2017.

[DNP13] Marc Peter Deisenroth, Gerhard Neumann and Jan Peters. “A Survey on Policy Search for Robotics”. In: *Foundations and Trends® in Robotics* 2.1–2 (Aug. 2013), pp. 1–142.

[DR11b] Marc Peter Deisenroth and Carl Edward Rasmussen. “PILCO: A Model-Based and Data-Efficient Approach to Policy Search”. In: *Proceedings of the 28th International Conference on International Conference on Machine Learning*. ICML’11. Bellevue, Washington, USA: Omnipress, 2011, pp. 465–472.

[Edm+19] Mark Edmonds et al. “A Tale of Two Explanations: Enhancing Human Trust by Explaining Robot Behavior”. In: *Science Robotics* 4.37 (2019), eaay4663.

[FL03] M. Fox and D. Long. “PDDL2.1: An Extension to PDDL for Expressing Temporal Planning Domains”. In: *Journal of Artificial Intelligence Research* 20 (Dec. 2003), pp. 61–124. arXiv: 1106.4561.

[Fra18] Peter I. Frazier. *A Tutorial on Bayesian Optimization*. July 2018. arXiv: 1807.02811 [cs, math, stat].

[GFF19] Daniel Gordon, Dieter Fox and Ali Farhadi. “What Should I Do Now? Marrying Reinforcement Learning and Symbolic Planning”. In: *arXiv:1901.01492 [cs]* (Jan. 2019). arXiv: 1901.01492 [cs].

[GK08] Matthew Grounds and Daniel Kudenko. “Combining Reinforcement Learning with Symbolic Planning”. In: *Adaptive Agents and Multi-Agent Systems III. Adaptation and Multi-Agent Learning*. Ed. by Karl Tuyls et al. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2008, pp. 75–86.

[HST91] Tsutomu Hasegawa, T Suehiro and Kunikatsu Takase. “A Model-Based Manipulation System with Skill-Based Execution in Unstructured Environment”. In: *Fifth International Conference on Advanced Robotics’ Robots in Unstructured Environments*. IEEE, 1991, pp. 970–975.

[Ijs+12] Auke Jan Ijspeert et al. “Dynamical Movement Primitives: Learning Attractor Models for Motor Behaviors”. In: *Neural Computation* 25.2 (Nov. 2012), pp. 328–373.

[KCM18] Rituraj Kaushik, Konstantinos Chatzilygeroudis and Jean-Baptiste Mouret. “Multi-Objective Model-based Policy Search for Data-efficient Learning with Sparse Rewards”. In: *Proceedings of The 2nd Conference on Robot Learning*. PMLR, Oct. 2018, pp. 839–855.

[KMD13] S. Koos, J. Mouret and S. Doncieux. “The Transferability Approach: Crossing the Reality Gap in Evolutionary Robotics”. In: *IEEE Transactions on Evolutionary Computation* 17.1 (Feb. 2013), pp. 122–145.

[KK92] John R Koza and John R Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. Vol. 1. MIT press, 1992.

[Kru+16] V. Krueger et al. “A Vertical and Cyber–Physical Integration of Cognitive Robots in Manufacturing”. In: *Proceedings of the IEEE* 104.5 (May 2016), pp. 1114–1127.

[Kru+19] Volker Krueger et al. “Testing the Vertical and Cyber-Physical Integration of Cognitive Robots in Manufacturing”. In: *Robotics and Computer-Integrated Manufacturing* 57 (June 2019), pp. 213–229.

[Mar+19] Roberto Martín-Martín et al. “Variable Impedance Control in End-Effector Space: An Action Space for Reinforcement Learning in Contact-Rich Tasks”. In: *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Nov. 2019, pp. 1010–1017.

[Mar+14] A. Marzinotto et al. “Towards a Unified Behavior Trees Framework for Robot Control”. In: *2014 IEEE International Conference on Robotics and Automation (ICRA)*. May 2014, pp. 5420–5427.

[May+21] Matthias Mayr et al. “Learning of Parameters in Behavior Trees for Movement Skills”. In: *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2021 IEEE/RSJ International Conference on Intelligent Robots and Systems. IEEE, 2021, pp. 7572–7579.

[May+22b] Matthias Mayr et al. “Learning Skill-Based Industrial Robot Tasks with User Priors”. In: *2022 IEEE 18th International Conference on Automation Science and Engineering (CASE)*. IEEE, 2022, pp. 1485–1492.

[MTZ78] Jonas Mockus, Vytautas Tiesis and Antanas Zilinskas. “The Application of Bayesian Methods for Seeking the Extremum”. In: *Towards global optimization* 2.117-129 (1978), p. 2.

[MC17] Jean-Baptiste Mouret and Konstantinos Chatzilygeroudis. “20 Years of Reality Gap: A Few Thoughts about Simulators in Evolutionary Robotics”. In: *Proceedings of the Genetic and Evolutionary Computation Conference Companion on - GECCO '17*. Berlin, Germany: ACM Press, 2017, pp. 1121–1124.

[NKO19] Luigi Nardi, David Koeplinger and Kunle Olukotun. “Practical Design Space Exploration”. In: *2019 IEEE 27th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*. Oct. 2019, pp. 347–358.

[Nar+17] Luigi Nardi et al. “Algorithmic Performance-Accuracy Trade-off in 3D Vision Applications Using HyperMapper”. In: *2017 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*. May 2017, pp. 1434–1443.

[PKP20] Biswajit Paria, Kirthevasan Kandasamy and Barnabás Póczos. “A Flexible Framework for Multi-Objective Bayesian Optimization Using Random Scalarizations”. In: *Proceedings of The 35th Uncertainty in Artificial Intelligence Conference*. PMLR, Aug. 2020, pp. 766–776.

[Pax+17] Chris Paxton et al. “CoSTAR: Instructing Collaborative Robots with Behavior Trees and Vision”. In: *2017 IEEE International Conference on Robotics and Automation (ICRA)*. May 2017, pp. 564–571.

[RGK17] F. Rovida, B. Grossmann and V. Krüger. “Extended Behavior Trees for Quick Definition of Flexible Robotic Tasks”. In: *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Sept. 2017, pp. 6793–6800.

[Rov+18] F. Rovida et al. “Motion Generators Combined with Behavior Trees: A Novel Approach to Skill Modelling”. In: *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Oct. 2018, pp. 5964–5971.

[Rov+16] Francesco Rovida et al. “Planning for Sustainable and Reliable Robotic Part Handling in Manufacturing Automation”. In: *Proceedings of the Twenty-Sixth International Conference on Automated Planning and Scheduling (ICAPS 2016)*. 2016.

[Rov+17] Francesco Rovida et al. “SkiROS—A Skill-Based Robot Control Platform on Top of ROS”. In: *Robot Operating System (ROS): The Complete Reference (Volume 2)*. Ed. by Anis Koubaa. Studies in Computational Intelligence. Cham: Springer International Publishing, 2017, pp. 121–160.

[Sar+21] Vasanth Sarathy et al. “SPOTTER: Extending Symbolic Planning Operators through Targeted Reinforcement Learning”. In: *Proceedings of the 20th International Conference on Autonomous Agents and MultiAgent Systems*. AAMAS ’21. Richland, SC: International Foundation for Autonomous Agents and Multiagent Systems, May 2021, pp. 1118–1126.

[Sou+21] Artur Souza et al. “Bayesian Optimization with a Prior for the Optimum”. In: *Machine Learning and Knowledge Discovery in Databases. Research Track*. Ed. by Nuria Oliver et al. Cham: Springer International Publishing, 2021, pp. 265–296.

[Ste+15] M. Stenmark et al. “On Distributed Knowledge Bases for Robotized Small-Batch Assembly”. In: *IEEE Transactions on Automation Science and Engineering* 12.2 (Apr. 2015), pp. 519–528.

[Sty+22] Jonathan Styrud et al. “Combining Planning and Learning of Behavior Trees for Robotic Assembly”. In: *2022 International Conference on Robotics and Automation (ICRA)*. May 2022, pp. 11511–11517.

[Tho+03] U. Thomas et al. “Error-Tolerant Execution of Complex Robot Tasks Based on Skill Primitives”. In: *2003 IEEE International Conference on Robotics and Automation (Cat. No.03CH37422)*. Vol. 3. Taipei, Taiwan: IEEE, 2003, pp. 3069–3075.

[Tho+13] Ulrike Thomas et al. “A New Skill Based Robot Programming Language Using UML/P Statecharts”. In: *2013 IEEE International Conference on Robotics and Automation*. May 2013, pp. 461–466.

[Ude+10] A. Ude et al. “Task-Specific Generalization of Discrete and Periodic Dynamic Movement Primitives”. In: *IEEE Transactions on Robotics* 26.5 (Oct. 2010), pp. 800–815.

[VGK19] Patrick Varin, Lev Grossman and Scott Kuindersma. “A Comparison of Action Spaces for Learning Manipulation Tasks”. In: *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2019, pp. 6015–6021.

[Yan+18] Fangkai Yang et al. “PEORL: Integrating Symbolic Planning and Hierarchical Reinforcement Learning for Robust Decision-Making”. In: *Proceedings of the 27th International Joint Conference on Artificial Intelligence*. IJCAI’18. Stockholm, Sweden: AAAI Press, July 2018, pp. 4860–4866.

Paper V

Paper 5

Learning Skill-based Industrial Robot Tasks with User Priors

Matthias Mayr
Lund University
matthias.mayr@cs.lth.se

Carl Hvarfner
Lund University
carl.hvarfner@cs.lth.se

Konstantinos Chatzilygeroudis
University of Patras, Greece
costashatz@upatras.gr

Luigi Nardi
Lund University
Stanford University, USA
lnardi@stanford.edu

Volker Krueger
Lund University
volker.krueger@cs.lth.se

Abstract

Robot skills systems are meant to reduce robot setup time for new manufacturing tasks. Yet, for dexterous, contact-rich tasks, it is often difficult to find the right skill parameters. One strategy is to learn these parameters by allowing the robot system to learn directly on the task. For a learning problem, a robot operator can typically specify the type and range of values of the parameters. Nevertheless, given their prior experience, robot operators should be able to help the learning process further by providing educated guesses about where in the parameter space potential optimal solutions could be found. Interestingly, such prior knowledge is not exploited in current robot learning frameworks. We introduce an approach that combines user priors and Bayesian optimization to allow fast optimization of robot industrial tasks at robot deployment time. We evaluate our method on three tasks that are learned in simulation as well as on two tasks that are learned directly on a real robot system. Additionally, we transfer knowledge from the corresponding simulation tasks by automatically constructing priors from well-performing configurations for learning on the real system. To handle potentially contradicting task objectives, the tasks are modeled as multi-objective problems. Our results show that operator priors, both user-specified and transferred, vastly accelerate the discovery of rich Pareto fronts, and typically produce final performance far superior to proposed baselines.

1 Introduction

In modern manufacturing settings, the setup of a robot system for a new task should be fast and easy. At the same time, to assure safety of equipment and workers it is important that robot behavior is always predictable and explainable.

One way to combine all these requirements is a system based on modular and explainable *robot skills* [Kru+16]. Robot skills (or just skills) are semantically defined parametric actions where parameters have to be chosen based on the task at hand through planning, sensing and knowledge integration. For contact-rich tasks, however, it can still be very challenging to find well-functioning skill parameter values, as even human operators may encounter difficulties identifying a successful and robust parameter set [May+22c]. One solution is to allow a robot system to find these parameters through reinforcement learning (RL) directly on the task. A recent approach [May+21] to RL in the industrial context suggests to model explainable policies with behavior trees (BTs) and a motion generator (MG) [Rov+18], and to optimize these through efficient policy learning and domain randomization within a digital twin. However, beyond the straight learning problem, there are two important aspects to consider:

(1) Learning often needs to balance various key performance indicators (KPIs) such as robot speed, safety or the need to minimize interaction forces with manufacturing parts. While [May+21] is able to handle only single-objective learning, we argue that many tasks are best described as multi-objective learning problems where the outcome is a variety of policies for different trade-offs between the objectives [May+22c]. An operator can then choose a solution with the desired properties.

(2) The learning problem can be reduced by constraining the parameter space within which the RL approach is searching for suitable skill parameters. Given some prior experience, a robot operator can typically not only help to constrain the search space, but further accelerate the learning process by providing guesses on where in the parameter space optimal solutions may be found, and have these regions be emphasized throughout the learning process. These guesses could be based on the operator's intuition or experience, or be generated from previously utilized policies on similar tasks through transfer learning. Fig. 5.1 visualizes how guesses regarding good configurations can be represented through a *probability density over optimal parameter settings*. In fact, by utilizing knowledge in the form of a probability density over the optimum, the search can focus on promising areas of the search space without explicitly restricting the search

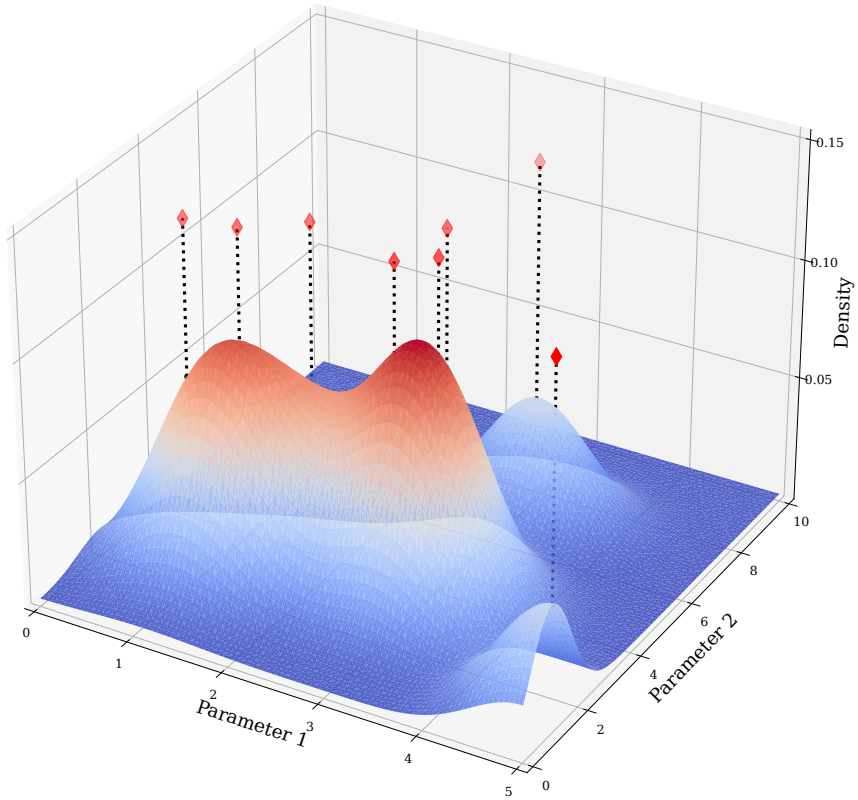


Figure 5.1: The visualization of two dimensions of a multimodal prior formed by results from learning in simulation that can be used as a prior for learning on the real robot. Well-performing configurations in simulation (red diamonds) are used to construct a probability density over optimal parameter settings, enabling accelerated learning on the real task through Bayesian optimization.

space to these regions.

With this paper we make the following contributions:

1. We introduce an approach to incorporate parameter priors in the form of probability densities for the optimal configuration, in conjunction with multi-objective Bayesian optimization, into the learning process of industrial robot tasks.

2. We assess the performance of our method and evaluate the influence of well-placed and misleading priors on various tasks.

3. We show priors learned in simulation can enable accelerated optimization

on the real system, without the requirement of explicit operator knowledge.

2 Related Work

2.1 Reinforcement Learning with Robot Systems

Reinforcement Learning (RL) [SB18], and especially Direct Policy Search (PS) methods [Cha+19; DNP13], have been successful in robotics applications as they can be applied in high-dimensional continuous state-action problems. In order to apply PS methods successfully in robotics applications, one must do one (or a combination) of the following: (a) provide prior structure in the policy, (b) learn models of the dynamics or the expected return, and/or (c) use prior information about the search space [Cha+19].

The type of the policy structure plays an important role for the effectiveness of learning in practical robotics applications. However, there is always a tradeoff between having a representation that is expressive enough (e.g. large neural networks), and one that provides a space that is efficiently searchable [Cha+19]. Another important property is choosing the level on which the policy interacts with the robot (e.g. task-space vs joint-space); it has been shown that it can strongly influence the learning speed and the quality of the obtained solutions [VGK19; Mar+19].

Traditionally, the robotic controllers (or policies) are hand-designed; either via an analytic model-based approach (e.g. inverse dynamics controllers) [PS08] or as more general finite state machines (FSMs) [Cal+16]. These hand-designed policies usually come with a small amount of parameters (thus efficiently searchable), but might need to be re-designed when changing task or robot. In principle, most of the hand-designed policies are easily intepretable and we can infer why the robot is choosing a specific action.

The most popular way of defining a policy in the RL literature is as a function approximator (e.g. a neural network) [SS13; Cha+19; DNP13]. In this case, policies can be very expressive and task-agnostic, which means that the same policies can be re-used on different tasks or robots without substantial changes. The commonly used policy representations for learning systems include radial basis function networks [DNP13], dynamical movement primitives [Ijs+12; Ude+10] and feed-forward neural networks [DNP13; Cha+17]. In recent years, in RL settings, deep artificial neural networks (ANNs) have become the default policy type [Aru+17; Cha+19]. ANN policies enable us to easily increase the

expressiveness and generality of the policy, but can make optimization difficult due of the large number of parameters. In contrast to the previous policy types, this type of policy is harder to interpret, and it is generally a difficult task to know why the robot is choosing a specific reaction to an environmental change. Therefore, [May+21] and [May+22c] suggest to learn interpretable policies based on BTs and a MG [Rov+18] that are well suited for the requirements of an industrial environment.

2.2 Meta-learning for Bayesian Optimization

For the optimization of black-box functions, Bayesian Optimization (BO) constitutes a sample-efficient [Fra18] choice across multiple fields, including machine learning [SLA12], robotics [Cal+14], and hardware design [NKO19]. In BO, there are several means of injecting prior knowledge, the most common of which is through the choice of the Gaussian Process kernel. However, several approaches have been proposed to explicitly bias or direct the optimization, based on accumulated data or knowledge from previous tasks.

Transfer learning approaches make use of data obtained from previous experiments to guide current ones. Feurer et. al. [Feu+22] propose to combine surrogate models from previous experiments, and use the combined surrogate when performing a new task. Perrone et. al. [Per+18], restrict the search space of the new task to some convex region based on optima found on previous tasks, excluding suboptimal regions in the outer edges of previous search spaces.

Injecting explicit prior distributions over the location of an optimum is an emerging topic in BO. In these cases, the user explicitly defines a prior probability distribution $\pi(\boldsymbol{x})$ that encodes their belief on where the optimum is likely to be located. Souza et. al. propose *BOPrO* [Sou+21], which combines $\pi(\boldsymbol{x})$ with a data-driven model into a pseudo-posterior. From the pseudo-posterior, configurations are selected using the Expected Improvement (EI) acquisition function. Hvarfner et. al. [Car+22] propose πBO, which weights the acquisition function by $\pi(\boldsymbol{x})$, and decays the prior's influence over time. Consequently, it retains conventional convergence rates [Bul11] for any choice of $\pi(\boldsymbol{x})$ when used in conjunction with EI.

3 Approach

In order to learn robot tasks, we utilize two main components: 1) *SkiROS2* [Rov+17; Kru+16] is a skill-based system for *ROS*. It provides a world model (digital twin) and a skill representation based on behavior trees (BT), and has an integrated task planner. 2) An RL framework that integrates optimizers and provides a simulation as well as reward calculation [May+21; May+22c].

When setting up the system for a new task, the operator typically specifies a high-level goal using the *Planning Domain Definition Language* (PDDL). Once the planner finds a valid sequence of skills, the learnable parameters in the skills are automatically identified. The operator can state lower and upper bounds for the parameters to be learned. A more detailed description can be found in [May+22c]. In this work, we additionally allow for specification of a unimodal or multimodal prior for the optimum. Therefore, expert knowledge and previous experiences can be actively integrated into the learning process.

3.1 Skill Representation

We adopt the skill definition from [Kru+16; Rov+17] that defines a skill as an ability to change the world state. To support task planning, a skill has a set of pre-conditions that must be satisfied before the execution is started and post-conditions that state and verify the effects. Skills usually model instructions from standard operation procedures (SOPs), such as *pick¡object¿*, *insert¡object¿*, *press¡object¿*, etc.

Our parametric skills can utilize the world model to retrieve knowledge and are implemented with BTs. A BT [CÖ17a], is a plan representation and execution tool that is used in many areas including computer games and robotics [CÖ17a; Iov+22]. As in [RGK17; Mar+14], we define it as a directed acyclic graph with nodes and edges. It consists of *control flow nodes* or *processors* that link *execution nodes*. Three common types of *control flow nodes* are 1) *sequence* (logical *AND*), 2) *selector* (logical *OR*) and 3) *parallel*. A BT always has one initial node with no parents, called *Root node*. During the execution of a BT, a periodic *tick signal* is injected into the *Root node*. The signal is routed according to the *control flow nodes* and the return statements of the children. The leaves of the BT are the *execution nodes* that execute one cycle and output one of the three signals when being ticked: *success*, *failure* or *running*. Execution nodes subdivide into 1) *action* and 2) *condition* nodes. An action node performs its operation iteratively at every tick and returns *running* while it is not done, and *success* or

failure otherwise. A condition node performs an atomic operation and can only return *success* or *failure*, but never *running*. One significant difference between BTs and FSMs is that BTs implement a two-way control flow like function calls in programming languages. In contrast, classical FSMs implement a one-way control flow similar to *GOTO* statements, which often becomes challenging to scale and maintain.

For modeling parametric movements, our movement skills use a MG in combinations with BTs [Rov+18]. This formulation is a type of trajectory-based policy structure that explicitly operates in end-effector space. The advantages of such movement skills are that they are modular, interpretable and allow for an easy adaption to environmental changes, e.g. if objects are relocated. In line with [Rov+18], we require a compliant robot controller that operates in end-effector space and utilize the same Cartesian impedance controller as in [May+22c].

3.2 Policy Optimization

In order to optimize for policy parameters, we adopt the policy search formulation in [DNP13; Cha+19; Cha+17]. We formulate a dynamical system of the form:

$$\mathbf{x}_{t+1} = \mathbf{x}_t + M(\mathbf{x}_t, \mathbf{u}_t, \boldsymbol{\phi}_R), \tag{5.1}$$

with continuous-valued states $\mathbf{x} \in \mathbb{R}^E$ and actions $\mathbf{u} \in \mathbb{R}^U$. The transition dynamics are modeled by a simulation of the robot and the environment $M(\mathbf{x}_t, \mathbf{u}_t, \boldsymbol{\phi}_R)$. They are influenced by the domain randomization parameters $\boldsymbol{\phi}_R$.

The goal is to find a policy $\pi, \mathbf{u} = \pi(\mathbf{x}|\boldsymbol{\theta})$ with policy parameters $\boldsymbol{\theta}$ such that we maximize the expected long-term reward when executing the policy for T time steps:

$$J(\boldsymbol{\theta}) = \mathbb{E}\left[\sum_{t=1}^{T} r(\mathbf{x}_t, \mathbf{u}_t)|\boldsymbol{\theta}\right], \tag{5.2}$$

where $r(\mathbf{x}_t, \mathbf{u}_t)$ is the immediate reward for being in state $\mathbf{x}$ and executing action $\mathbf{u}$ at time step t. The discrete switching of branches in the BT and most skills are not differentiable. Therefore, we frame the optimization in Eq. (5.2) as a black-box function and pursue the maximization of the reward function $J(\theta)$ only by using measurements of the function. The optimal reward function to solve the task is generally unknown, and a combination of reward functions is usually used. In the RL literature, this is usually done with a weighted average, that is, $r(\mathbf{x}_t, \mathbf{u}_t) = \sum_i w_i r_i(\mathbf{x}_t, \mathbf{u}_t)$. In this paper, we choose not to use a weighted average of reward functions that represent different objectives (as the optimal

combination of weights cannot always be found [KCM18]), but optimize for all objectives concurrently (Sec. 3.4) using Bayesian Optimization.

3.3 Bayesian Optimization

As mentioned in Section 3.2. we view the of optimization of our policy as an unknown black-box optimization problem. In this setting, information about the objective function f can only be extracted through the potentially noisy output y yielded by an given input $\boldsymbol{x}$. We wish to find $\boldsymbol{x}^* \in \arg\max_{\boldsymbol{x}\in\mathcal{X}} f(\boldsymbol{x})$ for some bounded, D-dimensional input space $\mathcal{X}$. As the function f is typically expensive in some resource of interest, one wishes to optimize f with a low total number of evaluations.

To solve the aforementioned black-box optimization problem, we employ BO. It aims to find $\boldsymbol{x}^*$ by sequentially selecting new design points $\{\boldsymbol{x}_i\}_{i=1}^N$ through some measure of utility, then receiving their corresponding output $\{y_i\}_{i=1}^N$, for some maximal number of iterations N. This is achieved through the use of a probabilistic surrogate model $p(f|\mathcal{D}_n)$, which uses all available observations $\mathcal{D}_n = \{\boldsymbol{x}_i, y_i\}_{i=1}^n$ at a given iteration n to emulate the objective f. After obtaining an initial number of observations through some space-filling design (Design of Experiments, DoE). BO uses the aforementioned utility measure, commonly called an acquisition function, to decide on subsequent queries. A query is selected $\boldsymbol{x}_{n+1}$ by considering a trade-off between uncertain regions (exploration) and regions of high predicted value (exploitation) under $p(f)$. After evaluation, the observation y_{n+1} is obtained, and the surrogate model is updated. The most commonly used acquisition function is Expected Improvement (EI) [JSW98; Bul11], which is defined as

$$\boldsymbol{x}_{n+1} \in \arg\max_{\boldsymbol{x}\in\mathcal{X}} \mathbb{E}_y \left[[(y_n^* - y(\boldsymbol{x})]^+ \right] \tag{5.3}$$

where y_n^* is the best obtained (noisy) output at iteration n. EI is simple to implement, and can be computed closed-form.

For tasks with a substantial level of noise, such as robot learning, the consideration of noise in the objective f is of particular importance [Cal+16]. EI can potentially struggle in such noisy settings [Vaz+08; Let+19; GL10] due to its consideration of the improvement of a noisy observation. As such, we utilize a noisy-robust version, called Noisy EI (NEI) [Let+19], defined as

$$\boldsymbol{x}_{n+1} \in \arg\max_{\boldsymbol{x}\in\mathcal{X}} \mathbb{E}_f \left[[(f_n^* - f(\boldsymbol{x})]^+ \right] \tag{5.4}$$

which, despite similarities to EI, requires approximation through Monte Carlo by sampling latent function values at each prior observed location. Through its consideration of the noiseless optimum f^* as opposed to y^*, NEI, yields desired robustness to noise and converges to EI in a noiseless setting. For our experiments, we use the *HyperMapper* [NKO19; Nar+17] framework, and the NEI acquisition function introduced in Eq.(5.4), modified for a multi-objective setting. The modification is covered exhaustively in Section 3.4.

3.4 Multi-objective Optimization

In the multi-objective optimization setting, we consider a set of K objectives $\boldsymbol{f} = (f_1, \ldots, f_K)$, all defined over the same D-dimensional input space $\mathcal{X}$ and observed through a noisy output $\boldsymbol{y} = y_1, \ldots, y_K$. Our goal is to find the set of points that are not dominated by any other point in $\mathcal{X}$. For a pair of inputs $\boldsymbol{x}$ and $\boldsymbol{x'}$ with corresponding multi-objetive outputs $\boldsymbol{y}$ and $\boldsymbol{y'}$, $\boldsymbol{x'}$ is said to dominate $\boldsymbol{x}$ if $y'_k \geq y_k, \forall k \in \{1, \ldots K\}$, i.e. $\boldsymbol{y'}$ is superior in every objective. The set of non-dominated points, known as the Pareto frontier, is in turn expressed as $\Gamma = \{\boldsymbol{x} \in \mathcal{X} : \nexists \boldsymbol{x'} \text{ s.t. } \boldsymbol{x'} \prec \boldsymbol{x}\}$, where $\prec$ is the domination relation. Γ thus contains the set of maximally desired points, with various trade-offs in the objectives.

For our experiments, we use the random scalarizations approach proposed by Paria et. al. [PKP20], which computes the acquisition function across objectives as

$$\alpha(\boldsymbol{x}, \boldsymbol{\lambda}) = \sum_{k=1}^{K} \lambda_k \alpha_k(\boldsymbol{x}), \quad \sum_{k=1}^{K} \lambda_k = 1 \tag{5.5}$$

on the K objective-wise acquisition functions $\{\alpha_k\}_{k=1}^{K}$ and the scalarization $\boldsymbol{\lambda}$, sampled from a Dirichlet distribution. To quantify the quality of the obtained Pareto front, we use the Hypervolume Indicator (HV) [CSR15] metric. HV computes the volume spanned by the Pareto-optimal observations $\{\boldsymbol{y}_p\}_{p=1}^{|\Gamma|}$ from some reference point $\boldsymbol{r}$ as $\mathcal{HV}(\Gamma, \boldsymbol{r}) = \lambda_K \left(\cup_p [\boldsymbol{y}_p, \boldsymbol{r}]\right)$ where $[\boldsymbol{y}_p, \boldsymbol{r}]$ denotes the hyperrectangle bounded by vertices $\boldsymbol{y}_p$ and $\boldsymbol{r}$, and λ_K is the K-dimensional Lebesgue measure.

3.5 Priors for the Optimum

For our experiments, we use the *HyperMapper* implementation of πBO [Car+22], combined with the NEI acquisition function. Moreover, we utilize the prior for sampling during DoE. Critically, $\pi(\boldsymbol{x})$ is defined on the input space $\mathcal{X}$. As such,

the user defines one prior jointly over all objectives, so that the prior emphasizes regions that are believed to contribute to the Pareto front, with no emphasis towards any particular objective.

The tasks use two types of priors over the optimum:
Simulation: We consider Gaussian densities for each parameter. These are set once, prior to conducting the experiments by an expert operator. The priors are left untouched for the whole duration of the experiments to avoid bias.
Real system: In addition to the operator priors, we use πBO in a transfer learning setting, as we form the prior based on previous data. The Pareto front designs obtained from simulation are used to form a Gaussian kernel density estimator (KDE) [Par62]. KDE places a Gaussian density on each point on Γ, which enables the automatic construction of multimodal distributions, exemplified by Figure 5.1. The obtained Pareto front from simulation then serves as a starting point for learning in the real system, without involving the operator.

4 Experiments

We evaluate the influence of priors for the optimum on the learning process for three different tasks. Their setup on the workstation is shown in Fig. 5.2; Fig. 5.3 shows the learnable parameters. One of the tasks is a contact-free movement from one side of an object (e.g., the engine block in Fig. 5.2) to the other side. The other two tasks are contact-rich manipulations where a peg needs to be inserted into a hole and an object with an uneven weight distribution needs to be pushed to another location, respectively. All tasks have in common that they are solved with existing skills that use BTs and a MG to actuate the robot and where some skill parameters need to be found via learning. For each task, a learning scenario configuration file describes attributes such as the robot system to use, the configured reward functions and the learnable skill parameters with their bounds and optionally their priors.

All learning problems are defined as multi-objective problems where one objective assesses the performance and speed of solving the task and the other one is either a safety metric, defined by distance between the robot and a fragile item when passing it, or an impact metric, which considers the interactions forces of the robot and the work pieces. None of the tasks have a single best policy, and since they balance trade-offs between the competing objectives (KPIs), it is up to an operator to decide which one to use as a final policy. We utilize the HV defined by the Pareto-optimal points to measure how much of the solution space is covered. We locate the reference point $\boldsymbol{r}$ for the HV calculation based on the

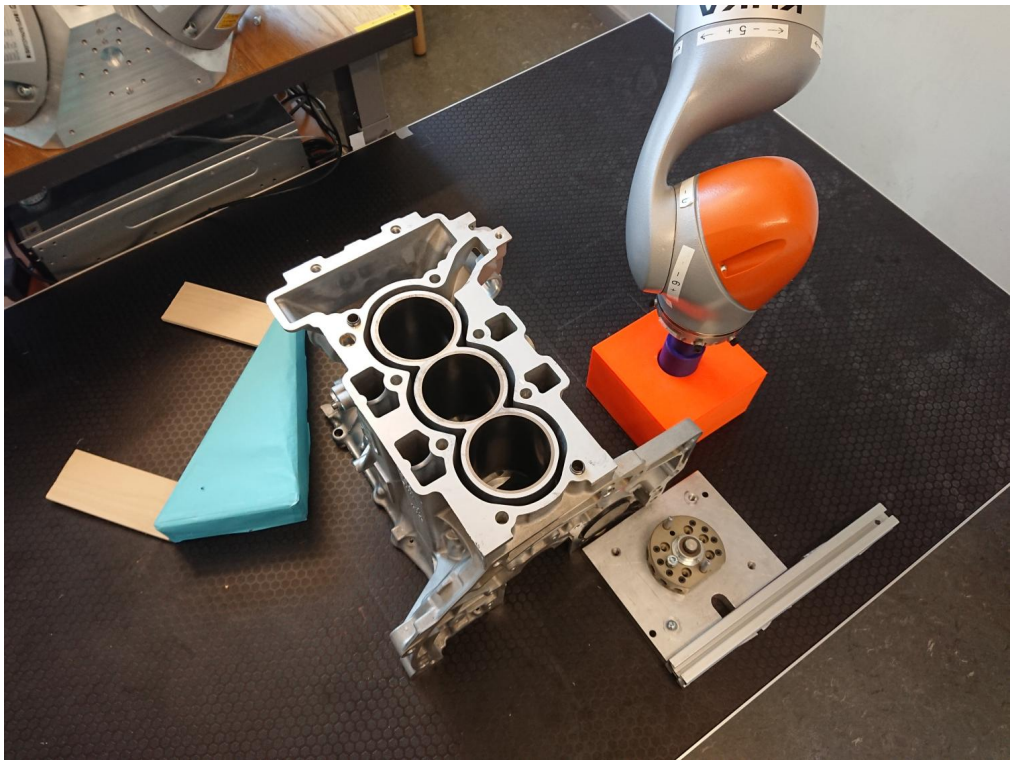


Figure 5.2: The experimental setup for the tasks. The orange block is used for the peg insertion. The engine is the obstacle that must be avoided with the use of movement skills when transitioning from one side to the other. The push task requires the blue object to be pushed from its current pose to the corner between the box and the fixture when the engine is not at the workstation.

worst value of the respective objective that is typically seen on a Pareto front for that task.

We evaluate the influence of priors that are defined by a domain expert and represent a typically chosen trade-off between task performance and safety or impact. Furthermore, we evaluate the impact of misleading priors, which are purposely designed to not adequately solve the task. In practice, the operator prior puts high density on regions of the search space that are believed to yield Pareto-optimal policies, whereas the misleading prior puts very high density on an outer edge of the search space - a choice a reasonable operator would likely not make.

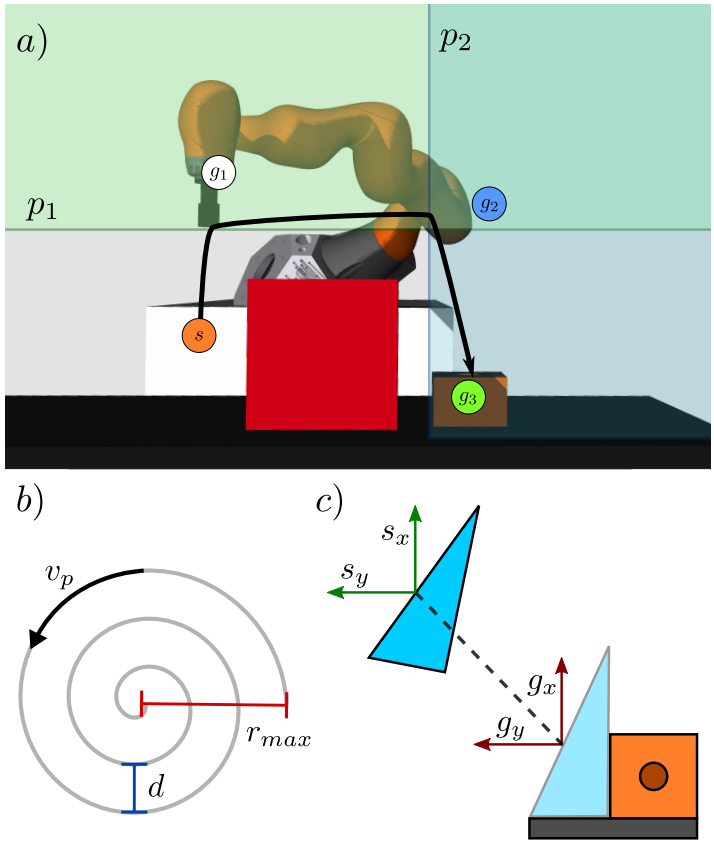


Figure 5.3: A depiction of the learnable parameters of the different tasks. a) The setup of the obstacle avoidance task with the parametric goal points g_1 and g_2 and the adjustable thresholds p_1 and p_2 in one possible motion configuration. b) The spiral of the search motion for the peg insertion is defined by the pitch d, the maximal r_{max} and the path velocity v_p. In addition, a downward force is set. c) The learnable offsets for the start and goal location of the push task are shown.

4.1 Learning in Simulation

As suggested in [May+21] and [May+22c], we learn the tasks in simulation based on a digital twin of the experimental setup. To have a performance reference, we compute two performance baselines for each of the three tasks where we learn the skill parameters through (1) random search and (2) BO with no priors. We repeat every experiment configuration 20 times to account for noise.

Peg Insertion Task

The goal of the peg insertion task is to insert a peg into a hole with a 1.5 mm larger radius. The setup imitates a piston insertion into the engine shown in Fig. 5.2. The configuration of such an insertion does not allow to tilt the piston. Therefore, the insertion strategy is to hold the object upright and to perform an Archimedes spiral as a search motion. The realization of a spiral is defined by the path velocity of the reference point, the pitch (i.e. the distance between lines in the spiral) and the maximal radius. Furthermore, the insertion skill sets a downward force that is applied by the arm while searching.

As in [May+22c], this task has two objectives: 1) the performance of the insertion which is assessed with the distance of the peg to the hole as well as a success reward if the peg is inserted by more than 0.01 m and 2) an integral over the commanded force while searching.

In order to learn a robust solution, each candidate parameter set is evaluated 7 times: (1) each execution randomly selects one out of the five start positions for the robot arm, (2) in simulation the hole is translated horizontally by a Gaussian offset with a standard deviation of 7 mm. See [May+22c] for more details about the experiment.

Object Pushing Task

This task requires to push an object with an uneven weight distribution from a start location (shown in Fig. 5.2) to the corner between the block and the metal fixture. The pushing is done with a square peg that is 0.07 m wide. The parametric push skill first moves the end effector to a location above the object, before it lowers it and performs a Cartesian linear motion towards the goal. The start and goal location of the push movement can be altered in both horizontal directions. This allows for learning of a push motion that starts from a location at the side of the object, and which implicitly takes the center of

mass into consideration. Every parameter set is evaluated 7 times where (1) the start position is randomly selected from a set of 4 initial positions and (2) the position of the object and the target are slightly perturbed by Gaussian offsets to avoid overfitting.

The performance of the task is assessed by the distance of the end effector to the target pose and by the position and orientation error of the object with respect to the target pose. The other objective assesses the total amount of force that the robot arm applies on the environment by integrating over the error between the actual pose of the end effector and the reference pose. See [May+22c] for a more detailed description.

Obstacle Avoidance Task

The goal of the task is to find a policy that uses parametric movement skills to avoid a static obstacle in the workspace. The structure of the skill is predefined and set up so that an obstacle can be passed from above. As shown in Fig. 5.3, the end effector starts at the point s and moves towards the goal (g_1) until it is above the parametric threshold p_1 in z-direction. When above p_1, the reference point will move towards the goal (g_2) until the threshold p_2 in y-direction is reached. Then, the motion towards the point g_3 is started. The learnable parameters include the thresholds p_1 and p_2 as well as the y and z coordinates of the parametric goal points (g_1) and (g_2). See [May+21] for additional details.

This task uses a positive reward that evaluates the distance between the end effector and the goal position. Furthermore, there is a fixed reward when reaching the goal. The safety objective evaluates the distance between the end effector and the object and the table.

Results

The experimental results are summarized in Fig. 5.4. In the peg insertion task BO without priors (blue) performed equally well than with the operator priors (green). This task also allowed for a quick recovery from misleading priors (red), indicating that it is easier to learn than the other tasks. In the other two tasks operator priors (green) greatly improved the learning speed and learning results. Operator priors vastly outperformed the baselines, as it yielded an increase in final HV of about 40% over BO with no priors. At the same time, less than 40% of the iterations were needed to achieve the final performance of the baselines.

The tasks also showed that the usage of deliberately misleading priors can generally hamper the learning performance. To provide an additional indication of the performance of the operator priors for a specific task, we performed random sampling in the space that is defined by these priors (brown). While priors sampling performs equally well in the peg insertion task it shows significantly worse performance in the other two tasks.

4.2 Learning with the Real system

While learning in simulation has several advantages, it also has limitations: Especially contact-rich tasks require an accurate model of the robot system and the workstation to allow the learning policies to transfer to the real system. This can be difficult to achieve and to maintain. Therefore, we learn the peg insertion task and the obstacle avoidance task directly on the real system.

We learn using the same operator priors as in simulation. As a baseline we use BO without priors. In addition, we also use the Pareto-optimal points that were obtained by learning in simulation as a multimodal prior when learning on the real system as an application of transfer learning. This can be particularly interesting, because it does not necessarily require operators to be able to specify priors. Moreover, the approach allows further refinement on the real system in case the task was not accurately modeled in simulation.

For each of the tasks, we do four repetitions of each configuration to reduce measurement noise. When using the Pareto-optimal points from simulation for a learning process on the real system, we use the points of a single run in simulation that applied BO without priors. This means that the operator never needed to explicitly state any priors.

Peg Insertion Task

When learning this task on the real system, we utilize the same five start positions as above when learning in simulation. Every parameter set is evaluated three times, randomly selecting one of the start positions for each run.

Obstacle Avoidance Task

Since learning this task on the real robot system can result in collisions with the object, the engine in Fig. 5.2 is replaced by an object that avoids damages

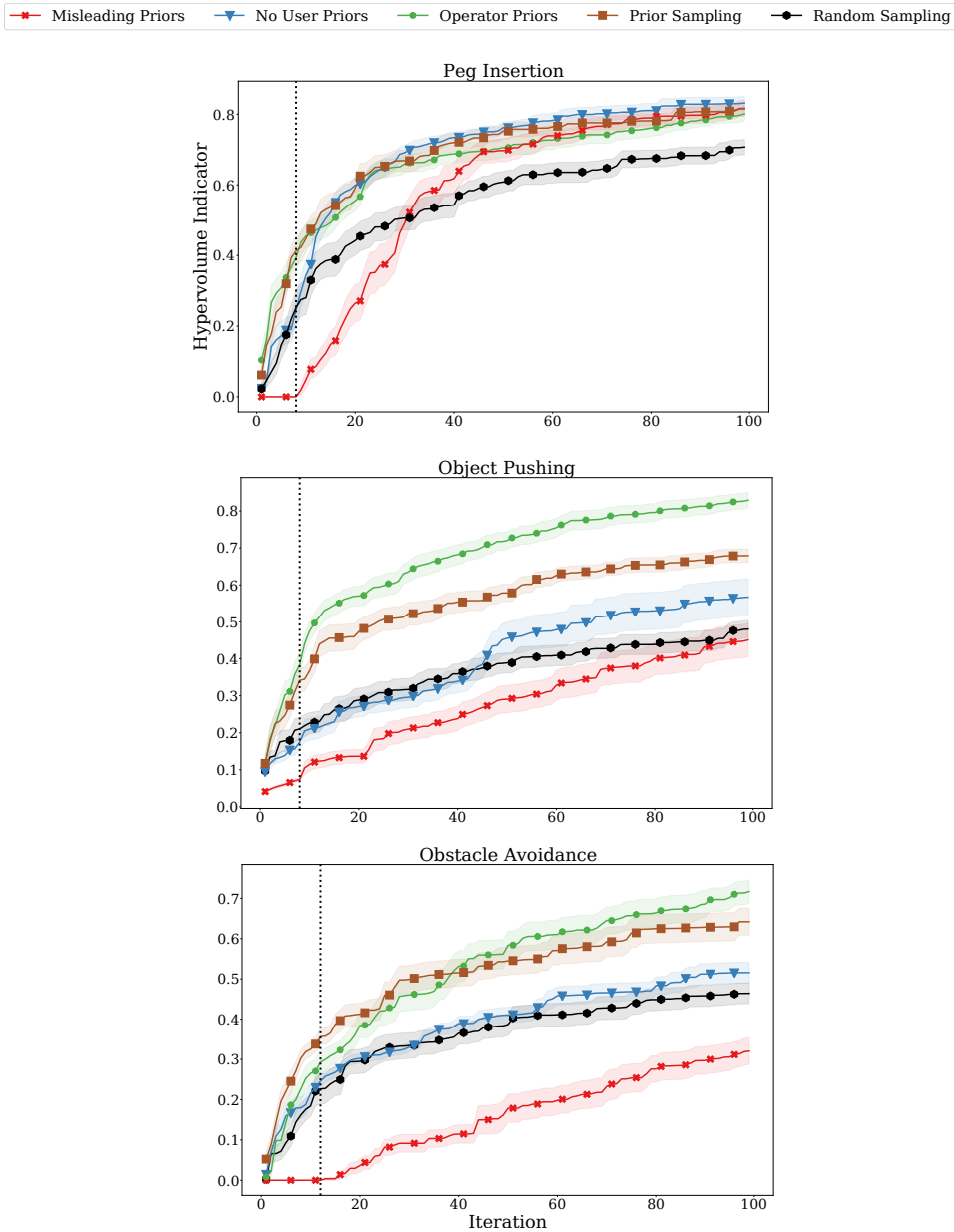


Figure 5.4: The learning progress of the peg insertion, the object pushing and the obstacle avoidance tasks in simulation. The dashed line denotes the end of the DoE phase and the shaded regions are the standard error of the mean. BO with operator priors improves substantially on both BO without priors and prior sampling for two tasks. For the less difficult peg insertion tasks, all approaches but random sampling achieve comparable performance.

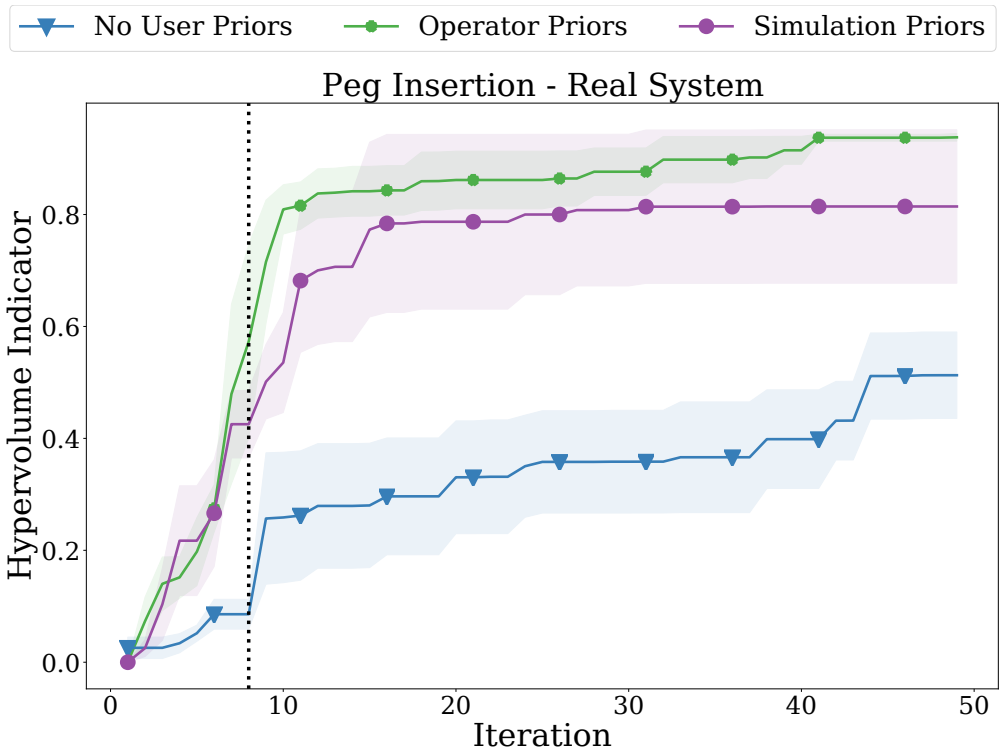


Figure 5.5: The learning progress of the peg insertion task on the real robot system. The dashed line denotes the end of the DoE phase and the shaded regions are the standard error of the mean. Both operator priors and simulation priors yield substantial performance gains over BO without priors.

to the robot. Furthermore, since successful policies do not interact with the environment in this task, every parameter set is evaluated only once.

Results

The experimental results are summarized in Fig. 5.5 and 5.6. They demonstrate again that well-placed operator priors could accelerate the learning and yielded to better learning results. In both tasks the priors derived from simulation results perform equally well initially, but can eventually be outperformed by the operator priors. However, since the simulation priors do not need to be defined explicitly, they are particularly interesting for new tasks or less experienced operators. In both tasks it takes less than 30% of the iterations to achieve the same performance as BO without priors. When learning the peg insertion task,

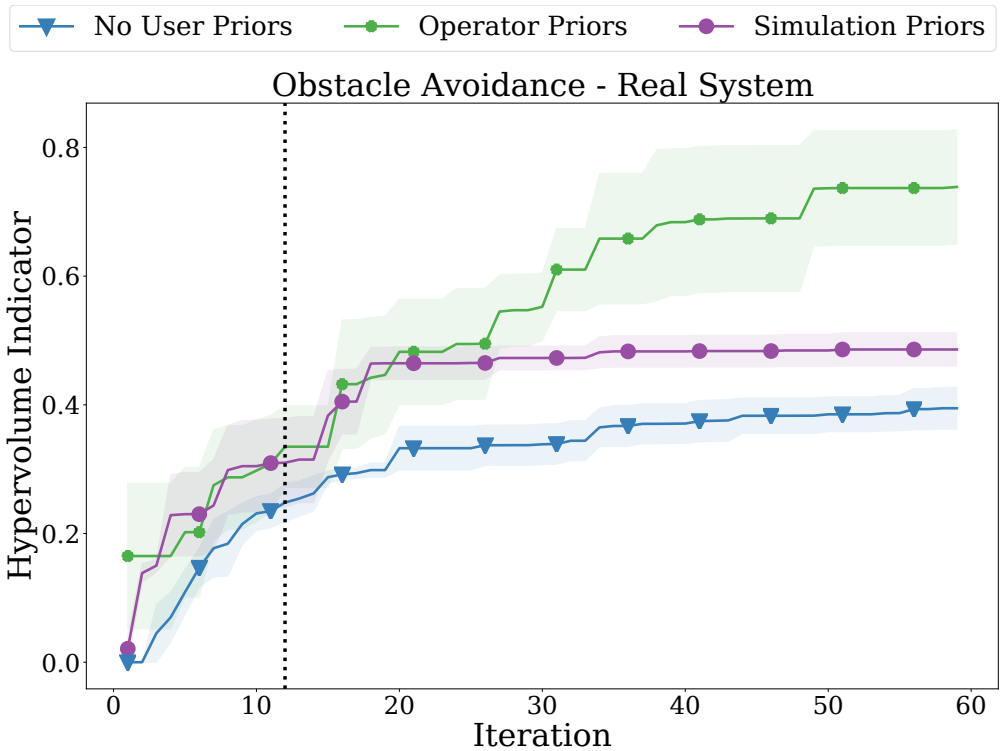


Figure 5.6: The learning progress of the obstacle avoidance task on the real robot system. The dashed line denotes the end of the DoE phase and the shaded regions are the standard error of the mean. Both operator priors and simulation priors yield substantial performance gains over BO without priors.

the average force applied by the robot was 32 % lower when using priors from simulation and 47 % lower with operator priors compared to a policy search without priors. In the obstacle avoidance task, the average amount of required interferences by robot operators due to forceful collisions with the object was much lower when learning with operator (10.75 ± 4.43) and simulation priors (4.5 ± 0.8) than with BO without priors (24.25 ± 4.43). This indicates that even the safety of a policy search can be increased by incorporating well-chosen priors.

5 Conclusions

We evaluated the influence of prior beliefs about the location of good candidate solutions when learning several industrial robot tasks. Since the parameters of interpretable robot skills often have a concrete meaning, they offer a natural opportunity for robot operators to incorporate their knowledge and experiences into the learning process. We have shown that expert operator priors can substantially speed up the search and yield higher performing policies, and seldom harm the performance. We have also demonstrated how using results from learning in simulation as priors can automate the prior design, and how this choice accelerates the learning of the same task on the real robot system. Lastly, we have highlighted the risk and potential performance loss associated with specifying a drastically incorrect prior.

We believe that the usage of priors for robot tasks learning in a combination with skills and the knowledge integration is a promising direction to achieve intelligent robot systems that can quickly learn to adapt. Moreover, the usage of priors can ease the adaption of RL in industrial robot tasks by providing operators an intuitive tool to guide a learning process. We are planning to look more into the transfer of knowledge between different tasks and robot configurations. Furthermore, multi-fidelity learning could combine a small amount of executions on the real system with learning in simulation to allow for a quicker and safer adjustment to new tasks.

Appendix

The implementation as well as additional information are available at: `https://github.com/matthias-mayr/SkiREIL`

Acknowledgement

This work was partially supported by the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by Knut and Alice Wallenberg Foundation. This research was also supported in part by affiliate members and other supporters of the Stanford DAWN project — Ant Financial, Facebook, Google, InfoSys, Teradata, NEC, and VMware.

References

[Aru+17] Kai Arulkumaran et al. “Deep Reinforcement Learning: A Brief Survey”. In: *IEEE Signal Processing Magazine* 34.6 (Nov. 2017), pp. 26–38.

[Bul11] A. D. Bull. “Convergence Rates of Efficient Global Optimization Algorithms”. In: *Journal of Machine Learning Research* 12.88 (2011), pp. 2879–2904.

[Cal+14] Roberto Calandra et al. “Bayesian Gait Optimization for Bipedal Locomotion”. In: *Learning and Intelligent Optimization*. Ed. by Panos M. Pardalos et al. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2014, pp. 274–290.

[Cal+16] Roberto Calandra et al. “Bayesian Optimization for Learning Gaits under Uncertainty: An Experimental Comparison on a Dynamic Bipedal Walker”. In: *Annals of Mathematics and Artificial Intelligence* 76.1-2 (Feb. 2016), pp. 5–23.

[CSR15] Yongtao Cao, Byran J. Smucker and Timothy J. Robinson. “On Using the Hypervolume Indicator to Compare Pareto Fronts: Applications to Multi-Criteria Optimal Experimental Design”. In: *Journal of Statistical Planning and Inference* 160 (May 2015), pp. 60–74.

[Car+22] Carl Hvarfner et al. “piBO: Augmenting Acquisition Functions with User Beliefs for Bayesian Optimization”. In: *International Conference on Learning Representations*. 2022.

[Cha+17] Konstantinos Chatzilygeroudis et al. “Black-Box Data-Efficient Policy Search for Robotics”. In: *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Sept. 2017, pp. 51–58.

[Cha+19] Konstantinos Chatzilygeroudis et al. “A Survey on Policy Search Algorithms for Learning Robot Controllers in a Handful of Trials”. In: *IEEE Transactions on Robotics* (2019), pp. 1–20.

[CÖ17a] Michele Colledanchise and Petter Ögren. *Behavior Trees in Robotics and AI: An Introduction*. Chapman \& Hall/CRC Press, Aug. 2017.

[DNP13] Marc Peter Deisenroth, Gerhard Neumann and Jan Peters. “A Survey on Policy Search for Robotics”. In: *Foundations and Trends® in Robotics* 2.1–2 (Aug. 2013), pp. 1–142.

[Feu+22] Matthias Feurer et al. *Practical Transfer Learning for Bayesian Optimization*. Oct. 2022. arXiv: 1802.02219 [cs, stat].

[Fra18] Peter I. Frazier. *A Tutorial on Bayesian Optimization*. July 2018. arXiv: 1807.02811 [cs, math, stat].

[GL10] Robert Gramacy and Herbert Lee. "Optimization under Unknown Constraints". In: *Bayesian Statistics 9* (Apr. 2010).

[Ijs+12] Auke Jan Ijspeert et al. "Dynamical Movement Primitives: Learning Attractor Models for Motor Behaviors". In: *Neural Computation* 25.2 (Nov. 2012), pp. 328–373.

[Iov+22] Matteo Iovino et al. "A Survey of Behavior Trees in Robotics and AI". In: *Robotics and Autonomous Systems* 154 (Aug. 2022), p. 104096.

[JSW98] Donald R. Jones, Matthias Schonlau and William J. Welch. "Efficient Global Optimization of Expensive Black-Box Functions". In: *Journal of Global Optimization* 13.4 (Dec. 1998), pp. 455–492.

[KCM18] Rituraj Kaushik, Konstantinos Chatzilygeroudis and Jean-Baptiste Mouret. "Multi-Objective Model-based Policy Search for Data-efficient Learning with Sparse Rewards". In: *Proceedings of The 2nd Conference on Robot Learning*. PMLR, Oct. 2018, pp. 839–855.

[Kru+16] V. Krueger et al. "A Vertical and Cyber–Physical Integration of Cognitive Robots in Manufacturing". In: *Proceedings of the IEEE* 104.5 (May 2016), pp. 1114–1127.

[Let+19] Benjamin Letham et al. "Constrained Bayesian Optimization with Noisy Experiments". In: *Bayesian Analysis* 14.2 (June 2019), pp. 495–519.

[Mar+19] Roberto Martín-Martín et al. "Variable Impedance Control in End-Effector Space: An Action Space for Reinforcement Learning in Contact-Rich Tasks". In: *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Nov. 2019, pp. 1010–1017.

[Mar+14] A. Marzinotto et al. "Towards a Unified Behavior Trees Framework for Robot Control". In: *2014 IEEE International Conference on Robotics and Automation (ICRA)*. May 2014, pp. 5420–5427.

[May+21] Matthias Mayr et al. “Learning of Parameters in Behavior Trees for Movement Skills”. In: *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2021 IEEE/RSJ International Conference on Intelligent Robots and Systems. IEEE, 2021, pp. 7572–7579.

[May+22c] Matthias Mayr et al. “Skill-Based Multi-Objective Reinforcement Learning of Industrial Robot Tasks with Planning and Knowledge Integration”. In: *2022 IEEE International Conference on Robotics and Biomimetics (ROBIO)*. IEEE, 2022, pp. 1995–2002.

[NKO19] Luigi Nardi, David Koeplinger and Kunle Olukotun. “Practical Design Space Exploration”. In: *2019 IEEE 27th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*. Oct. 2019, pp. 347–358.

[Nar+17] Luigi Nardi et al. “Algorithmic Performance-Accuracy Trade-off in 3D Vision Applications Using HyperMapper”. In: *2017 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*. May 2017, pp. 1434–1443.

[PKP20] Biswajit Paria, Kirthevasan Kandasamy and Barnabás Póczos. “A Flexible Framework for Multi-Objective Bayesian Optimization Using Random Scalarizations”. In: *Proceedings of The 35th Uncertainty in Artificial Intelligence Conference*. PMLR, Aug. 2020, pp. 766–776.

[Par62] Emanuel Parzen. “On Estimation of a Probability Density Function and Mode”. In: *The annals of mathematical statistics* 33.3 (1962), pp. 1065–1076.

[Per+18] Valerio Perrone et al. “Scalable Hyperparameter Transfer Learning”. In: *Advances in Neural Information Processing Systems*. Vol. 31. Curran Associates, Inc., 2018.

[PS08] Jan Peters and Stefan Schaal. “Learning to Control in Operational Space”. In: *The International Journal of Robotics Research* 27.2 (2008), pp. 197–212.

[RGK17] F. Rovida, B. Grossmann and V. Krüger. “Extended Behavior Trees for Quick Definition of Flexible Robotic Tasks”. In: *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Sept. 2017, pp. 6793–6800.

[Rov+18] F. Rovida et al. “Motion Generators Combined with Behavior Trees: A Novel Approach to Skill Modelling”. In: *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Oct. 2018, pp. 5964–5971.

[Rov+17] Francesco Rovida et al. “SkiROS—A Skill-Based Robot Control Platform on Top of ROS”. In: *Robot Operating System (ROS): The Complete Reference (Volume 2)*. Ed. by Anis Koubaa. Studies in Computational Intelligence. Cham: Springer International Publishing, 2017, pp. 121–160.

[SLA12] Jasper Snoek, Hugo Larochelle and Ryan P Adams. “Practical Bayesian Optimization of Machine Learning Algorithms”. In: *Advances in Neural Information Processing Systems*. Vol. 25. Curran Associates, Inc., 2012.

[Sou+21] Artur Souza et al. “Bayesian Optimization with a Prior for the Optimum”. In: *Machine Learning and Knowledge Discovery in Databases. Research Track*. Ed. by Nuria Oliver et al. Cham: Springer International Publishing, 2021, pp. 265–296.

[SS13] Freek Stulp and Olivier Sigaud. “Robot Skill Learning: From Reinforcement Learning to Evolution Strategies”. In: *Paladyn, Journal of Behavioral Robotics* 4.1 (Sept. 2013), pp. 49–61.

[SB18] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. Second edition. Adaptive Computation and Machine Learning Series. Cambridge, Massachusetts: The MIT Press, 2018.

[Ude+10] A. Ude et al. “Task-Specific Generalization of Discrete and Periodic Dynamic Movement Primitives”. In: *IEEE Transactions on Robotics* 26.5 (Oct. 2010), pp. 800–815.

[VGK19] Patrick Varin, Lev Grossman and Scott Kuindersma. “A Comparison of Action Spaces for Learning Manipulation Tasks”. In: *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2019, pp. 6015–6021.

[Vaz+08] Emmanuel Vazquez et al. “Global Optimization Based on Noisy Evaluations: An Empirical Study of Two Statistical Approaches”. In: *Journal of Physics: Conference Series* 135.1 (Nov. 2008), p. 012100.

Paper VI

Paper 6

Learning to Adapt the Parameters of Behavior Trees and Motion Generators (BTMGs) to Task Variations

Faseeh Ahmad
Lund University
faseeh.ahmad@cs.lth.se

Matthias Mayr
Lund University
matthias.mayr@cs.lth.se

Volker Krueger
Lund University
volker.krueger@cs.lth.se

Abstract

The ability to learn new tasks and quickly adapt to different variations or dimensions is an important attribute in agile robotics. In our previous work, we have explored Behavior Trees and Motion Generators (BTMGs) as a robot arm policy representation to facilitate the learning and execution of assembly tasks. The current implementation of the BTMGs for a specific task may not be robust to the changes in the environment and may not generalize well to different variations of tasks. We propose to extend the BTMG policy representation with a module that predicts BTMG parameters for a new task variation. To achieve this, we propose a model that combines a Gaussian process and a weighted support vector machine classifier. This model predicts the performance measure and the feasibility of the predicted policy with BTMG parameters and task variations as inputs. Using the outputs of the model, we then construct a surrogate reward function that is utilized within an optimizer to maximize the performance of a task over BTMG parameters for a fixed task variation. To demonstrate the effectiveness of our proposed approach, we conducted experimental evaluations on push and obstacle avoidance tasks in simulation and with a real *KUKA iiwa* robot. Furthermore, we compared the performance of our approach with four baseline methods.

1 Introduction

Robots have been utilized effectively for many years in repetitive and automated industrial processes. However, despite the shift towards smaller batch sizes and increased demand for customization, many robot systems still require a lengthy and expensive reconfiguration process. To keep up with the demands of society and modern industrial production, robots should have the ability to adapt quickly to different situations. In these situations, the task formulations should be robust to failures, interpretable, and possibly reactive to failures. Additionally, the task formulations should also be adaptable to different variations or dimensions of the same task, such as pushing an object to different locations, picking an object from any location in the space, and avoiding an obstacle with different shapes and positions.

To overcome the challenges, Rovida F. et al. [Rov+18] have suggested a representation that combines behavior trees (BT) [CÖ14; CÖ17a] and motion generators (MG), (BTMG). In our previous work, we used BTMGs to model skills for contact-rich tasks such as inserting a peg into the hole to mimic engine assembly [Rov+18; May+21] and pushing an object to a target location [May+22a; May+22c].

A BTMG is a parameterized policy representation that combines the strengths of both behavior trees and motion generators. Behavior trees provide a clear and intuitive way to describe the high-level logic of the robot's behavior, while motion generators generate the low-level motion commands by controlling the end-effector in Cartesian space. For a more concrete definition of motion generators, refer to [Rov+18]. The parameters of a BTMG can be used to specify the structure of the behavior tree as well as values such as controller stiffness.

BTMGs are easy to interpret and can be designed to be **robust** to faults and failures that can occur during execution [Rov+18]. Furthermore, they have the ability to be **reactive** [CÖ14], allowing the robot to adapt and respond to current circumstances. Simple BTs can also be systematically combined with more complex ones to solve complex tasks [Rov+18; May+21; RGK17].

BTMGs are a promising technique for motion modeling because of their explicitness, robustness, and reactiveness. There are mainly three ways to set the parameters of BTMGs. One way is to specify them manually or fine-tune them by experts [Rov+18]. Another way is to determine those parameters through reasoning. However this requires the existence of such a reasoner for the task at hand, which can not always be assumed. Finally, BTMG parameters can be learned through reinforcement learning (RL) [May+22a; May+22c; May+22b].

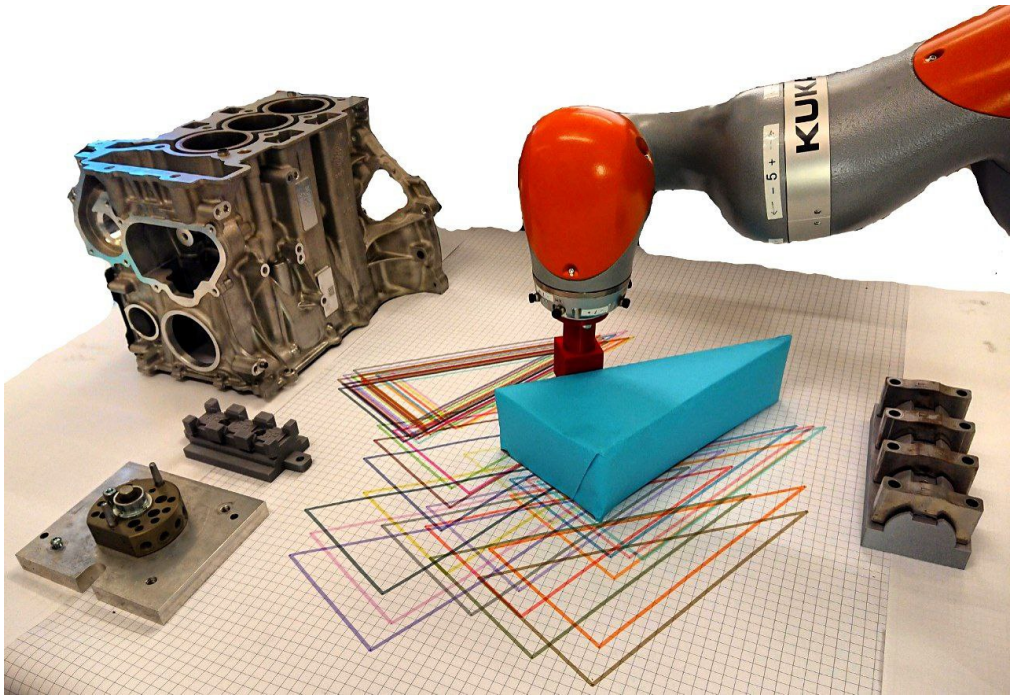


Figure 6.1: The experimental setup. It shows the object with the skewed weight distribution that is pushed with a $45\,\mathrm{mm}$ wide peg. On the table the different start and goal positions for the object can be seen in different colours. On the sides, some example sizes for obstacles are shown.

However, learned BTMG parameters are in many cases scenario-specific and changes in the setup may require relearning them.

Setting BTMG parameters using these methods can limit the usage of BTMGs in scenarios that require quick adaptability. For example, tasks such as pushing an object to different locations, picking an object from various locations, or even picking objects with various shapes would require updating the parameters of the respective BTMGs. This problem is also present in the original formalization of dynamic motion primitives (DMPs) [INS01; INS02] and was later addressed in [Ijs+12].

In this paper, we propose an extension to the BTMG formulation that enables quick adaptation to different task variations by incorporating a model that combines a Gaussian process (GP) and a weighted support vector machine (SVM) classifier. Our model uses a GP to learn a function that predicts the performance measure of a policy using task variations and BTMG parameters as inputs. Furthermore, the model also trains a weighted SVM classifier that predicts the

feasibility of a policy. For example, in a push task, the performance measure of a policy can be given by its overall reward, which depends on the error between the actual and target position of the pushed object. In this task, a policy can be feasible when this error is below a user-defined threshold. Once the model is trained, we optimize the BTMG parameters over the resulting surrogate reward function for a given new task variation.

The following are our main contributions:

- We extend BTMG policy representation that enables it to quickly adapt to task variations.
- We propose a model that combines a GP and a weighted SVM classifier to predict the performance measure and feasibility of a BTMG policy for a new task variation, and subsequently optimize the output of the model to obtain resulting BTMG parameters.
- We evaluate the performance of the proposed method in simulation and on a real *KUKA iiwa* robot for two tasks and compare its performance with four baselines.

2 Related Work

Movement primitives, based on motor primitives theory [Mus99; FH05], are mathematical formulations of dynamic systems that generate motions. Two well-known movement primitives used in robotics are Dynamic Movement Primitives (DMPs) [INS01; INS02] and Probabilistic Movement Primitives (ProMPs) [Par+13]. Movement primitives can be generalized and have proven successful in various robotics applications, such as dynamic motion primitives [INS01; INS02]. Similar to our BTMGs, DMPs intially lacked the capacity to generalize to different task parameters. This was resolved later by introducing a small change in the transformation system [Ijs+12].

While both DMPs and BTMGs are capable of generating motions through attractor landscapes, the parameters for DMPs are learned implicitly from a set of demonstrations, whereas parameters for BTMGs can be explicitly specified manually, inferred through a reasoner, or learned using RL. Nevertheless, a comprehensive comparison of the two approaches would require further investigation and is outside the scope of this paper.

DMPs have been extended with intermediate via points [Nin+11; Nin+12; WA18; ZGA19], and can generalize to new goals by interpolating weights of neighboring DMPs [Wei+13] or by using Gaussian Process Regression (GPR) to generate new parameters [FUG11]. Furthermore, GPs [RW06] have been used to generalize DMPs to external task variations, arbitrary movements, and adapting trajectories to new situations online in [AMB16; FUH16; FUG11], respectively. In [Lee+18], Gaussian mixture models are used to learn the mapping of task parameters and the forcing term of DMPs.

The mixture of movement primitives (MoMP) algorithm introduced in [MKP10; Mül+13], can also be used to generalize the basis movements stored in the library. The MoMP algorithm captures the robot's position and velocity as parameters for the expected hitting position and velocity. A new motion is generated by a weighted sum of DMPs, assigning a probability to a DMP based on the sensed state. MoMPs and ProMPs have been applied successfully in various applications, including learning striking movements for table tennis robots [MKP10; Gom+16] and solving Human-Robot collaborative tasks [Mae+17] using ProMPs.

We draw inspiration from prior work on DMPs to extend BTMG's formulation by incorporating generalization to different task variations using GP, as seen in [FUG11; AMB16; FUH16]. These studies employed GPs to directly map task variations to DMP parameters, which we refer to as the *direct* model in this paper. However, our approach differs significantly in how we use GPs. Instead of using the *direct* model, we propose a model that combines GP with a weighted SVM classifier to predict the performance of tasks and the feasibility of a policy, using task variations and BTMG parameters as inputs. Since our model predicts both performance measure and feasibility, we refer to it as the *PerF* model, short for performance and feasibility.

3 BTMG and Task Variations

We define BTMG as a parametric policy representation, BTMG($\boldsymbol{\theta}$) where $\boldsymbol{\theta} \in \mathbb{R}^N$. The parameters $\boldsymbol{\theta}$ can range from determining the structure of the behavior tree (BT) to specifying the controller stiffness values of the motion generator (MG). These parameters are further subdivided into **intrinsic** parameters $\boldsymbol{\theta}_i$ and **extrinsic** parameters $\boldsymbol{\theta}_e$ [Ahm+22].

Intrinsic parameters $\boldsymbol{\theta}_i$ determine the structure of the behavior tree, the number of control nodes, the type of motion generator, etc. For example, consider a

policy T_p for a push task, which has intrinsic parameters $\boldsymbol{\theta}_i$. These parameters are fixed and independent of the task instance, meaning that T_p uses the same $\boldsymbol{\theta}_i$ values regardless of the starting position, or the target position of the object. In other words, $\boldsymbol{\theta}_i$ is situation-invariant. Within the scope of this paper, these parameters are assumed to be known a priori.

Extrinsic parameters $\boldsymbol{\theta}_e$ are situation dependent e.g. to determine the applied force, offsets, and the velocity of the end effector. Again, $\boldsymbol{\theta}_e$ can be specified manually [Rov+18; Rov+16], inferred through a reasoning framework, or learned using RL. We have already demonstrated how RL can be used to obtain BTMG parameters [May+21] and used it in simulation and on a real robot to solve multi-objective tasks [May+22c; May+22b].

In addition to $\boldsymbol{\theta}$, we also consider task variations $\boldsymbol{v} \in \mathbb{R}^M$. Task variations refer to different possible alterations of a given task, such as different start and goal positions of an object. For example, a task variation $\boldsymbol{v}$ in the case of a push task would be a 4D vector consisting of the values of the start and goal positions of the object along the horizontal and vertical axes.

Note that the task variation parameters are different from the extrinsic BTMG parameters (Figure 6.2). We take two task variations $\boldsymbol{v}_1 = (v_{s_x}, v_{s_y}, v_{g1_x}, v_{g1_y})$ and $\boldsymbol{v}_2 = (v_{s_x}, v_{s_y}, v_{g2_x}, v_{g2_y})$ that define the start and goal positions of the object.
For variations $\boldsymbol{v}_1$ and $\boldsymbol{v}_2$, we have corresponding $\boldsymbol{\theta}_{e1} = (\theta_{e1_{s_x}}, \theta_{e1_{s_y}}, \theta_{e1_{g_x}}, \theta_{e1_{g_y}})$ and $\boldsymbol{\theta}_{e2} = (\theta_{e2_{s_x}}, \theta_{e2_{s_y}}, \theta_{e2_{g_x}}, \theta_{e2_{g_y}})$ that collectively define the start and the goal locations for the pushing action.

As $\boldsymbol{\theta}_i$ has no impact on adapting BTMGs to different variations, our objective in this paper is to establish a relationship between $\boldsymbol{\theta}_e$ and $\boldsymbol{v}$ that would enable the adaptation of BTMGs to new variations.

4 Approach

In this section, we explain how we adapt BTMG parameters for a new task variation by using the *PerF* model. Figure 6.3 shows how the *PerF* model works in comparison with a *direct* model. The overall approach is divided into the training (Sec. 4.1) and query phase (Sec. 4.2). In the training phase, we pass each task variation $\boldsymbol{v}_k \in \mathbb{V}_{train}$, into an extended RL pipeline similar to [May+22c]. For each learning process for different task variations, we utilize three sets of outputs from the RL pipeline to train the *direct* and the *PerF* models:

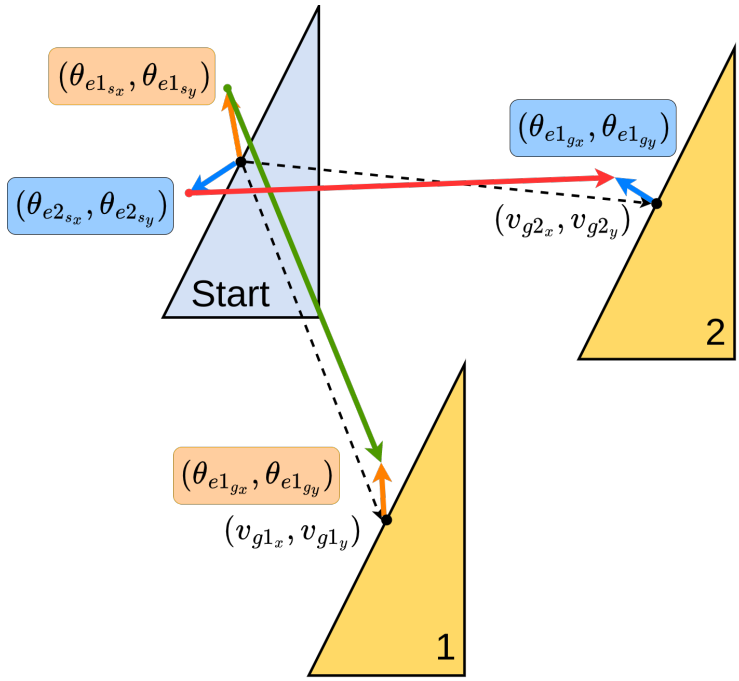


Figure 6.2: An illustration of two simplified task variations $\boldsymbol{v}_1$ and $\boldsymbol{v}_2$ in the pushing task that only vary the goal location. The orange and blue vectors are set by the respective learned extrinsic parameters $\boldsymbol{\theta}_{e1}$ and $\boldsymbol{\theta}_{e2}$, so that they define the resulting green and red push vectors that should successfully push the object.

1. *Best policies*: For every task variation we get the best performing policy: $\mathbb{T} = \{(\boldsymbol{v}_k, \boldsymbol{\theta}^*_{e,\boldsymbol{v}_k}) | k = 1, \ldots, n\}$
2. *All evaluated configurations and their rewards:* $\mathbb{K} = \{(\boldsymbol{v}_k, \boldsymbol{\theta}_{ei,\boldsymbol{v}_k}, r_{\boldsymbol{\theta}_{ei,\boldsymbol{v}_k}}) | k = 1, \ldots, n \text{ and } i = 1, \ldots, t \leq t_{\max}\}$
3. *All evaluated configurations and their feasibility:* $\mathbb{E} = \{(\boldsymbol{v}_k, \boldsymbol{\theta}_{ei,\boldsymbol{v}_k}, f_{\boldsymbol{\theta}_{ei,\boldsymbol{v}_k}}) | k = 1, \ldots, n \text{ and } i = 1, \ldots, t \leq t_{\max}\}$

The *direct* model M is trained with the set $\mathbb{T}$ and, as a result, learns to predict $\hat{\boldsymbol{\theta}}_e$ given $\boldsymbol{v}$. On the other hand, the *PerF* model is trained with the sets $\mathbb{K}$ and $\mathbb{E}$ and as a result it learns to predict the reward $\hat{r}$ and feasibility $\hat{f}$ of a policy with parameters $\boldsymbol{\theta}_e$. The model further uses $\hat{r}$ and $\hat{f}$ to generate a surrogate reward function that obtains $\hat{\boldsymbol{\theta}}_e$ given $\boldsymbol{v}$. For more details on how we obtain set $\mathbb{T}$, we direct the reader to [May+22c]. To obtain sets $\mathbb{K}$ and $\mathbb{E}$, we follow the same procedure as in [May+22c], retaining all configurations along with their respective rewards and feasibilities for a given task variation.

The intuition behind using the *PerF* model together with an optimizer is to guide the combination of GP and weighted SVM towards predicting policy parameters $\boldsymbol{\theta}_e$ that prioritize performance measure and feasibility. In contrast, the *direct* model does not take into account the performance measure and feasibility. In the following subsections, we explain our approach in more depth.

4.1 Training Phase

We frame the mapping of the task variations $\boldsymbol{v}$ to the extrinsic BTMG parameters $\boldsymbol{\theta}_e$ as a supervised learning problem. The training phase aims to learn two functions: $\hat{J}$ that predicts the reward achieved by a policy and $\hat{F}$ that predicts if a policy is feasible, see Figure 6.3. We propose to use GP and weighted SVM to learn $\hat{J} : (\boldsymbol{\theta}_e, \boldsymbol{v}) \mapsto \hat{r} \in \mathbb{R}$ and $\hat{F} : (\boldsymbol{\theta_e}, \boldsymbol{v}) \mapsto \hat{f} \in \{0, 1\}$. $\hat{J}$ and $\hat{F}$ are trained by data points in sets $\mathbb{K}$ and $\mathbb{E}$, provided by the RL pipeline introduced in [May+21].

For each task variation, $\boldsymbol{v}_k \in \mathbb{V}_{train}$, similar to [May+21; May+22c], we define $J_{\boldsymbol{v}_k}(\boldsymbol{\theta}_e)$ as the expected sum of individual rewards over time, given a sequence of extrinsic parameters $\boldsymbol{\theta}_{e1}, \boldsymbol{\theta}_{e2}, \ldots, \boldsymbol{\theta}_{et} \in \boldsymbol{\theta}_e$.

In [May+21; May+22c], we use Bayesian optimization (BO) as a black-box optimization method to obtain the optimal policy parameters $\boldsymbol{\theta}_e^*$ and the best reward $J_{\boldsymbol{v}_k}(\boldsymbol{\theta}_e^*)$. In this paper, however, we use BO to obtain $J_{\boldsymbol{v}_k}(\boldsymbol{\theta}_e)$ by computing $J_{\boldsymbol{v}_k}(\boldsymbol{\theta}_{e1}), J_{\boldsymbol{v}_k}(\boldsymbol{\theta}_{e2}), \ldots, J_{\boldsymbol{v}_k}(\boldsymbol{\theta}_{et})$ over the sequence $\boldsymbol{\theta}_{e1}, \boldsymbol{\theta}_{e2}, \ldots, \boldsymbol{\theta}_{et}$. This allows us to not only have the optimal policy parameters $\boldsymbol{\theta}_e^*$ and the corresponding best reward $J_{\boldsymbol{v}_k}(\boldsymbol{\theta}_e^*)$ but it also provides us with intermediate $\boldsymbol{\theta}_{et}$ and $J_{\boldsymbol{v}_k}(\boldsymbol{\theta}_{et})$. Overall, this provides us with large amount of data to train the $\hat{J}$ function and allows us to capture the overall reward landscape better.

In addition to learning the reward function $\hat{J}$, we also learn the feasibility function $\hat{F}$. The motivation behind learning $\hat{F}$ is twofolds: First, it provides a user-defined metric to evaluate the feasibility of a policy and second, it complements the reward formulation of a task by addressing the potential shortcomings of inaccurate reward formulations. In principle, we do not need to optimize feasibility if the reward formulation covers all aspects of the task. However, in practice, reward formulation is challenging, so feasibility addresses these shortcomings effectively. It ensures learned policies align with the task's requirements, despite imperfect reward formulations.

For a given task variation $\boldsymbol{v}_k$, we define the feasibility function $F_{\boldsymbol{v}_k}(\boldsymbol{\theta}_e)$ as a binary function that maps to 1 or 0 depending on whether the policy achieves

a user-defined metric of feasibility or not. Similar to $J_{\boldsymbol{v}_k}(\boldsymbol{\theta}_e)$, we obtain $F_{\boldsymbol{v}_k}(\boldsymbol{\theta}_e)$ by computing $F_{\boldsymbol{v}_k}(\boldsymbol{\theta}_{e1}), F_{\boldsymbol{v}_k}(\boldsymbol{\theta}_{e2}), \ldots, F_{\boldsymbol{v}_k}(\boldsymbol{\theta}_{et})$ for the sequence of evaluations $\boldsymbol{\theta}_{e1}, \boldsymbol{\theta}_{e2}, \ldots, \boldsymbol{\theta}_{et}$. For more details about the pipeline, we refer the reader to the policy optimization section in [May+21; May+22c].

To model $\hat{J}$ and $\hat{F}$, we obtain a sequence of BTMG parameter vectors, $\boldsymbol{\theta}_{e1}, \boldsymbol{\theta}_{e2}$, $\ldots, \boldsymbol{\theta}_{et}$, along with their corresponding reward values $J_{\boldsymbol{v}_k}(\boldsymbol{\theta}_{e1}), J_{\boldsymbol{v}_k}(\boldsymbol{\theta}_{e2}), \ldots,$ $J_{\boldsymbol{v}_k}(\boldsymbol{\theta}_{et})$ and feasibility values $F_{\boldsymbol{v}_k}(\boldsymbol{\theta}_{e1}), F_{\boldsymbol{v}_k}(\boldsymbol{\theta}_{e2}), \ldots, F_{\boldsymbol{v}_k}(\boldsymbol{\theta}_{et})$ for task variations. We then use these data points to train a GP and a weighted SVM classifier. This enables us to effectively model the underlying J and F.

4.2 Query Phase

The goal of this phase is to query the trained model with a new task variation $\boldsymbol{v}_p \in \mathbb{V}_{test}$ and obtain a $\hat{\boldsymbol{\theta}}_e$ by optimizing $\hat{J}(\boldsymbol{\theta}_{et}|\boldsymbol{v}_p)$ under the feasibility constraint $\hat{F}(\boldsymbol{\theta}_{et}|\boldsymbol{v}_p)$ (Figure 6.3). For this purpose, we use the $\hat{J}$ and $\hat{F}$ obtained in the training phase. We solve this as an optimization problem over a sequence of $\boldsymbol{\theta}_e$ for a new $\boldsymbol{v}_p$.

We begin the optimization process by specifying the optimizer type, the bounds for $\boldsymbol{\theta}_e$, and the maximum number of iterations $t_{\max}$. In our experiments, we used the Limited-memory Broyden–Fletcher–Goldfarb–Shanno (L-BFGS) [Byr+95; Zhu+97] algorithm, which refines an initial estimate of $\boldsymbol{\theta}_{e1}$ to iteratively obtain improved evaluation points $\boldsymbol{\theta}_{et}$, where $t \leq t_{\max}$, using the derivative as the driving function. For each new task variation $\boldsymbol{v}_k$, we run the optimizer to obtain a sequence of evaluation points $\boldsymbol{\theta}_{et}$.

Using $\hat{J}$ and $\hat{F}$, we define a surrogate reward $r_{\boldsymbol{v}_p} = \hat{r}_{\boldsymbol{\theta}_{et},\boldsymbol{v}_p} - (1 - \hat{f}_{\boldsymbol{\theta}_{et},\boldsymbol{v}_p})) * \mu$. Here, the first term corresponds to the output reward value computed by $\hat{J}$, while the second term penalizes the reward if $\hat{f}_{\boldsymbol{\theta}_{et},\boldsymbol{v}_p}$ maps to 0. We penalize the reward $\hat{r}_{\boldsymbol{\theta}_{ei},\boldsymbol{v}_p}$ by a small factor μ. We query the surrogate reward $r_{\boldsymbol{v}_p}$ for defined number of iterations or until the optimizer converges.

After the optimization phase, we select the $\boldsymbol{\theta}_{et}$ that maximizes both $\hat{J}(\boldsymbol{\theta}_{et}|\boldsymbol{v}_p)$ and is feasible $\hat{F}(\boldsymbol{\theta}_{et}|\boldsymbol{v}_p)$.

5 Experiments

We evaluated the efficacy of our approach in simulation and also by transferring of the simulation results to a real *KUKA iiwa* manipulator for two tasks: an

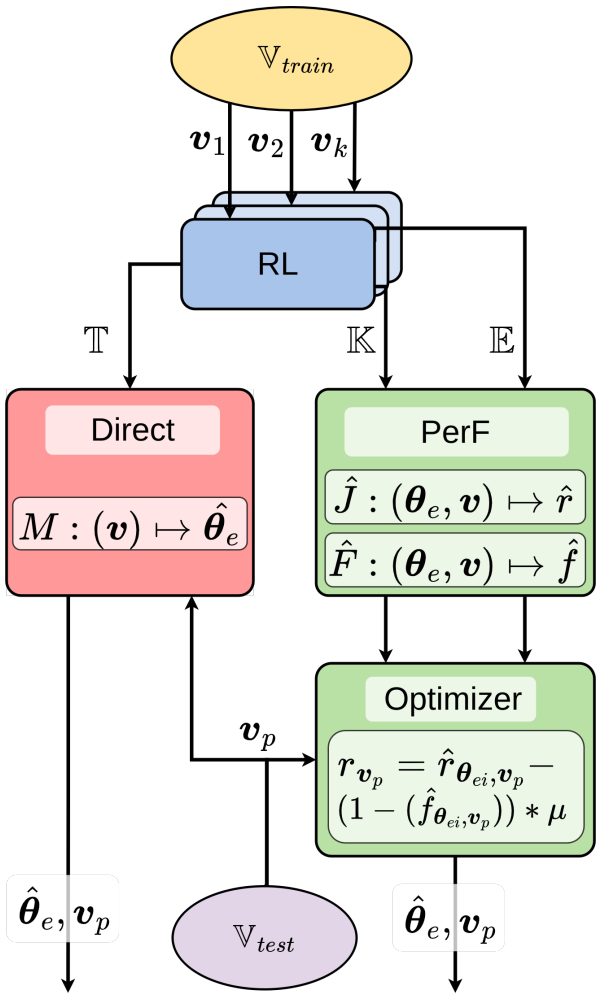


Figure 6.3: The pipeline of our approach and the *direct* model baseline. For every task variation $\mathbf{v}$, an RL problem is solved and the respective results are provided to the GP models. When querying for a new task variation $\mathbf{v}_p$ both models are queried for a set of extrinsic parameters $\hat{\boldsymbol{\theta}}_e$.

obstacle avoidance task and a pushing task, each having its own challenges. For simulation, we utilized the *DART* simulation toolkit [Lee+18] and in both simulation and reality, the robot arm was controlled using a Cartesian impedance controller [MS22], which helps reduce the disparities between simulation and reality. Additionally, for the push task, we further reduce the sim-to-real gap by adjusting the friction coefficient appropriately. For more detailed information on bridging the sim-to-real gap, please refer to [May+21].

To train our model, we considered 20 task variations that are learned for the same amount of iterations each. Using the method detailed in Sec. 4.1, we train the GP and the weighted SVM classifier with the resulting BTMG parameters, the feasibility, and the reward values. The weights of the SVM classifier are adjusted automatically to adjust bias induced by an unequal number of feasible and non-feasible policies. We then tested our approach on 20 unknown task variations. This experiment is repeated five times for both tasks to show the robustness of the approach.

We compare the performance of our approach with four baselines:

1. *Learned*: This baseline uses the RL pipeline described in [May+22c] to learn the BTMG parameters directly for the test variations. It shows which performance could be achieved if a new variation is learned from scratch instead of querying the model. Notably, our training data is generated in this way.

2. *Direct*: This model takes the best parameters for the training variations ($\mathbb{T}$) and learns a direct mapping from task variations to BTMG parameters without explicitly learning the reward.

3. *Nearest Neighbor*: For each test variation, we select the closest task variation in the training set and choose the corresponding BTMG parameters.

4. *Single Policy*: The learned BTMG parameters of a single training variation are used for all test variations. This baseline shows how well and how often the learned parameters for one task variation can be utilized in a different one without any changes.

Although our baselines may seem simplistic, they are deliberately selected to provide insights into the functionality and performance of our approach. Each of these baselines serves a specific purpose in understanding the capabilities and limitations of our approach.

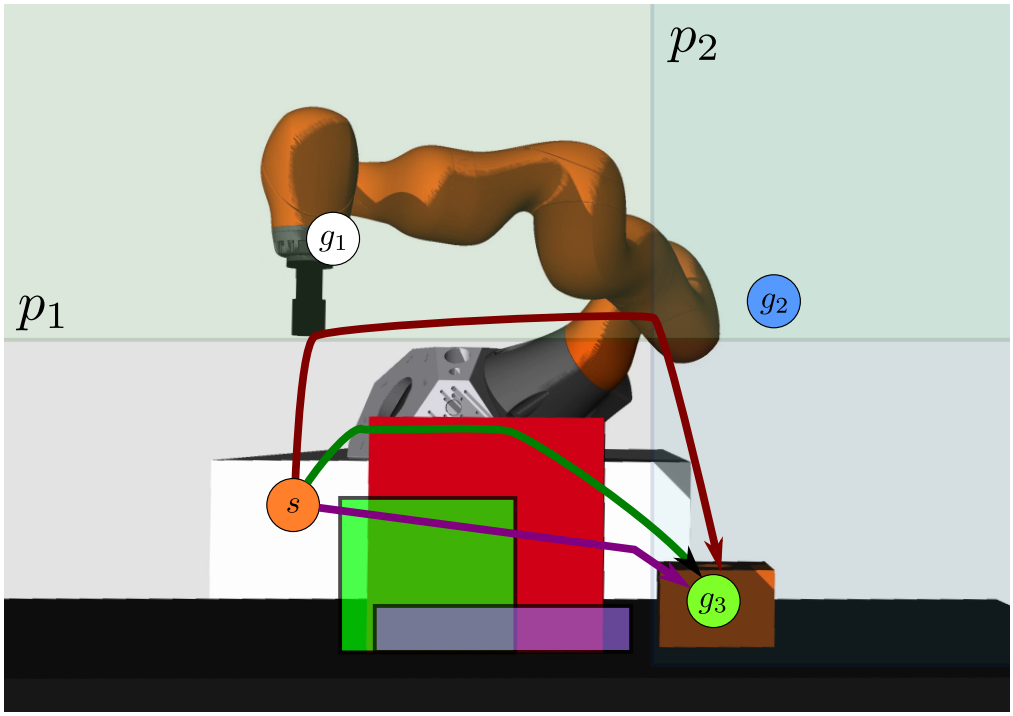


Figure 6.4: The obstacle task with some of the variations of the object location, width, and height. For each object configuration, valid example trajectories are shown in the same color. For the red trajectory, the intermediate goal points ($\mathbf{g}_1$ and $\mathbf{g}_2$) and two motion switching thresholds (p_1 and p_2) are shown.

We consider task-specific reward functions for both tasks. The rewards and feasibility measures for the tasks are defined separately in their respective sections.

5.1 Obstacle Avoidance Task

The objective of the obstacle avoidance task is to move the robot's end effector from the start to the goal location while avoiding an obstacle in the workspace. As shown in Fig. 6.4, the obstacle can vary in size and position. The goal is to find policies that navigate the robot around the obstacle while completing the task as quickly as possible, without violating the safety constraints that require the end effector to maintain a safe distance from the obstacle.

We consider three task variations: 1) obstacle height, 2) obstacle width, and 3) obstacle position in a horizontal direction (left-right in Fig. 6.4). The obstacle

varies in height from 0.049 m to 0.331 m and in width from 0.09 m to 0.331 m. The horizontal position ranges from 0.274 m to 0.311 m with respect to the origin. We use Latin hypercube sampling to ensure a more even sample distribution and obtain 20 task variations from the specified ranges. We learn each variation for 120 iterations.

This learning problem formulation has three rewards: 1) a fixed success reward, 2) a goal distance reward, and 3) an obstacle avoidance reward. The fixed success reward assigns a fixed reward if the BT finishes successfully. The positive goal distance reward increases, the closer the end effector gets to the goal. The obstacle avoidance reward is a negative function that penalizes end-effector states that are close to the obstacle. These reward functions are combined to encourage fast execution while discouraging getting too close to the obstacle. A policy is considered feasible if it satisfies two conditions: First, the end effector does not come closer to the obstacle than 40 mm. Second, the policy must successfully complete the BT by bringing the end effector to the goal position.

The policy for this task has six learnable parameters consisting of two coordinates of the intermediate goal points and two thresholds to transition between goal points. A more detailed description of the task is provided in [May+21; May+22c]. Notably, the structure of this policy with its thresholds allows for different movement strategies. For example, for flat obstacles, the goal can be reached with only a single intermediate point, while larger obstacles require both intermediate points, as shown in Fig. 6.4.

Results and Discussion

For the evaluation, we randomly sample 20 new task variations ($\mathbb{V}_{test}$) that are not included in the training set, and compare the performance of our proposed model and the baseline methods. Specifically, we assess the execution time and the reward achieved by each parameter configuration in the new task variation. The reward value is chosen as a performance metric as it reflects how well a policy balances between the goal-reaching and obstacle-avoidance objectives expressed in the reward functions.

The simulation results are shown in Fig. 6.5a) and b) and Table 6.1. They show that the policies obtained by optimizing the output of our *PerF* model performs similarly to the policies that are explicitly learned. Our model achieves a success percentage of 87 % compared to the 89 % of the learned ones and a total reward in a similar range. In contrast to that, the nearest neighbor baseline succeeds only in 71 % of the variations. The *direct* model also only achieves a

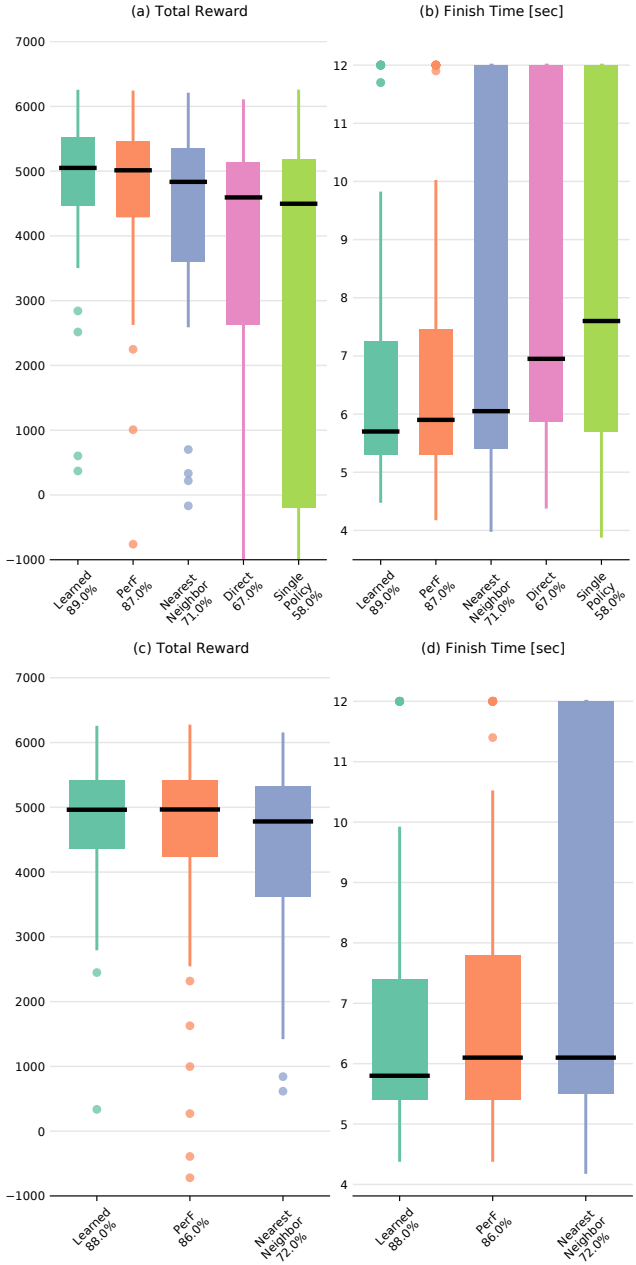


Figure 6.5: The total reward (a, c) and the execution time (b, d) of the obstacle task in simulation (a, b) and on the real system (c, d). The box plots show the median (black line) and interquartile range (25^{th} and 75^{th} percentile); the lines extend to the most extreme data points not considered outliers, and outliers are plotted individually. The success percentages are shown below the method names.

success percentage of 67 % and has significantly more outliers in the reward. Further investigation indicates that the reason for the low performance is that an interpolation between policies is often not valid. This is especially the case between motion configurations that use a single or both intermediate points.

Based on these results from simulation we also evaluated the learned policies, our model outputs and the nearest neighbor policies on the real robot system. Although this includes a transfer from simulation to the real system, the results shown in Fig. 6.5c) and d) have only minor variations from the simulation results. This also demonstrates the robustness of this policy formulation as a whole.

5.2 Push task

The goal of this task is to push an object from a varying start location to a varying goal location. The object is shown in Fig. 6.1 and has a skewed weight distribution with respect to its bounds.

We consider two types of task variations: 1) the starting position of the object in both horizontal directions and 2) the goal position of the object in both horizontal directions. For the starting position, we consider samples from a circle with a diameter of 0.16 m around a center point. For the goal position, a triangular-shaped region is used. Fig. 6.1 shows the start and goal positions for a single repetition.

The learning formulation has two rewards: 1) the object position reward, which is a function of the difference between the actual and desired goal position, and 2) the object orientation reward, which is based on the difference between the actual and desired goal orientation. For our experiment, we prioritize the object position reward, which is weighted 10 times more heavily than the orientation reward.

Similarly to previous work [May+22a; May+22c], the push task has four BTMG parameters that are learned. They are depicted in Fig. 6.2. These parameters control additional start and goal offsets in the horizontal directions (x, y), determining the shape of the push vector that is indicated in Fig. 6.2. The start and goal orientation of the object for this task are fixed.

The object being pushed is an right-angled triangular object with dimensions 0.3 m x 0.15 m x 0.07 m, and a weight of 2.5 kg. The tool on the end effector is a cubic peg with side lengths of 45 mm and therefore covers less than 15 % of the side length of the object. In this task, the error between the desired goal position and orientation and the achieved one serves as direct performance

Table 6.1: The median performance values and the 25^{th} and 75^{th} percentiles for both tasks. A "-" indicates that configuration was not evaluated.

Task		**Obstacle**			
Performance Measure		Total Reward		Finish Time [sec]	
	Environment	Simulation	Reality	Simulation	Reality
Learned	Mean	5050	4963	5.7	5.8
	Percentiles	(4467, 5531)	(4357, 5414)	(5.3, 7.3)	(5.4, 7.4)
PerF (ours)	Mean	**5013**	**4966**	**5.9**	**6.1**
	Percentiles	(4290, 5462)	(4238, 5426)	(5.3, 7.5)	(5.4, 7.8)
Nearest Neighbour	Mean	4834	4782	6.1	6.1
	Percentiles	(3607, 5357)	(3625, 5327)	(5.4, 12)	(5.5, 12)
Direct	Mean	4594	–	7	–
	Percentiles	(2635, 5143)	–	(5.9, 12)	–
Single Policy	Mean	4496	–	7.6	–
	Percentiles	(-200, 5184)	–	(5.7, 12)	–

Task		**Push**			
Performance Measure		Position Error [m]		Orientation Error [deg]	
	Environment	Simulation	Reality	Simulation	Reality
Learned	Mean	0.002	0.01	0.15	1.51
	Percentiles	(0.002, 0.003)	(0.007, 0.014)	(0.07, 0.33)	(0.98, 2.76)
PerF (ours)	Mean	**0.006**	**0.009**	**0.16**	**1.42**
	Percentiles	(0.004, 0.009)	(0.007, 0.012)	(0.07, 0.34)	(0.94, 2.74)
Nearest Neighbour	Mean	0.011	0.016	0.18	2.58
	Percentiles	(0.007, 0.083)	(0.01, 0.028)	(0.08, 29.48)	(1.18, 6.23)
Direct	Mean	0.007	–	0.11	–
	Percentiles	(0.005, 0.014)	–	(0.06, 0.26)	–
Single Policy	Mean	0.016	–	0.29	–
	Percentiles	(0.007, 0.123)	–	(0.1, 45.76)	–

measures for the policy.

Results and Discussion

The results for the simulation are shown in Figure 6.6a) and b). We consider a policy feasible if the position error between the goal location of the object and the desired goal location is less than 11 mm and the orientation error is less than 30 deg. The high success percentage of 97 % for the learned policies shows that it is generally possible to

solve this task. Our proposed model solves 86 % of the configuration and outperforms all baselines that do not require explicit learning. The gap to the *direct* model, which achieved a success rate of 65 %, is significant. The nearest neigh-

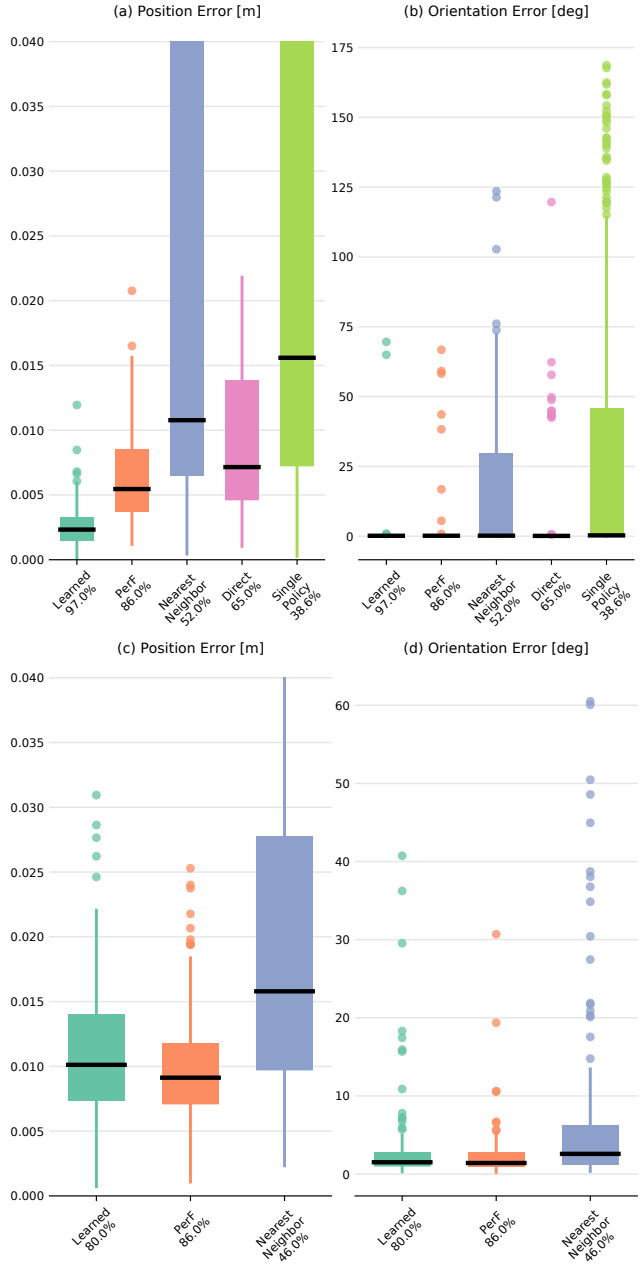


Figure 6.6: The final position error (a, c) and orientation error (b, d) of the push task in simulation (a, b) and on the real system (c, d). The box plots show the median (black line) and interquartile range (25^{th} and 75^{th} percentile); the lines extend to the most extreme data points not considered outliers, and outliers are plotted individually. The success percentages are shown below the method names.

bor and the single policy approach only achieved 52 % and 38 %, which shows not only the difficulty of the task but also excludes them as practical solutions.

Similar to the obstacle task, we also executed the learned policies on the real robot system. To account for the differences of such a contact-rich task to the simulation, we increase the allowed final position error by 4 mm but keep the same angular maximum.

The results for the evaluation on the real system are in Fig. 6.6c) and d) as well as in Table 6.1. As intuitively expected, the success percentages generally drop as not all policies transfer to the real system. Similar to the evaluation in simulation, the nearest neighbor baseline performs poorly. However, it is notable that our model now outperforms the explicitly learned policies in both the success rate and the final error. A possible explanation for this is that our model needed to generalize, whereas an explicitly learned policy is able to exploit the simulation to the maximum extent possible. During the experiments, we also observed that policies from our model generally kept a larger distance from the object when approaching it and also had fewer collisions with it.

To determine the time efficiency of our approach, we compute time required to compute BTMG parameters for 60 new task variations. This analysis compares learning BTMG parameters from scratch using the RL-pipeline and obtaining BTMG parameters using our approach. Starting from scratch with the RL-pipeline, median completion times were 770.315 seconds for the obstacle task and 1232.625 seconds for the push task. In contrast, the optimization phase of our approach achieved median completion times of 1.27 seconds for the obstacle task and 5.189 seconds for the push task. Additionally, obtaining a trained PERF model took an average of 66.628 seconds for the obstacle task and 317.025 seconds for the push task. During optimization, we observed some outliers, likely stemming from the stochastic nature of the process. The analysis was performed on a laptop equipped with an Intel(R) Core(TM) i7-10870H CPU running at 2.20GHz with 8 physical cores and hyper-threading, along with 64GB of RAM.

6 Conclusion and Future Work

Agile robotics requires that a system adapts quickly to changing conditions. In this work, we introduced an extension to BTMGs, a motion representation based on behavior trees and motion generators, which addresses this challenge. Our approach enables the use of learned policies in previously unseen variations of a task, allowing for fast adaption of robot behavior to changes in the task or

environment.

The experimental evaluation demonstrates that our approach effectively learns a model capable of adapting to new task variations. Our method exhibits comparable performance to explicitly trained policies and consistently outperforms all other baseline models. Furthermore, experiments conducted on the real robotic system demonstrate the successful transferability of our approach from simulation to reality, even in a contact-rich task. Notably, our proposed method can even outperform explicitly learned policies in the same contact-rich task, indicating superior generalization capabilities.

In future work, it is worth exploring whether the uncertainty modeled by the GP can be leveraged to make more accurate predictions about successful execution. This uncertainty measure could also be used for out-of-distribution detection. Another promising direction is to use the learned model to return policy parameters for task parameters, such as friction, for which the values are not known a priori. In this case, we could jointly optimize over both policy and task parameters to identify a compatible set of learned parameters.

References

[Ahm+22] Faseeh Ahmad et al. "Generalizing Behavior Trees and Motion-Generator (BTMG) Policy Representation for Robotic Tasks over Scenario Parameters". In: *2022 IJCAI Planning and Reinforcement Learning Workshop* (2022).

[AMB16] Tohid Alizadeh, Milad Malekzadeh and Soheila Barzegari. "Learning from Demonstration with Partially Observable Task Parameters Using Dynamic Movement Primitives and Gaussian Process Regression". In: *2016 IEEE International Conference on Advanced Intelligent Mechatronics (AIM)*. July 2016, pp. 889–894.

[Byr+95] Richard H Byrd et al. "A Limited Memory Algorithm for Bound Constrained Optimization". In: *SIAM Journal on scientific computing* 16.5 (1995), pp. 1190–1208.

[CÖ14] Michele Colledanchise and Petter Ögren. "How Behavior Trees Modularize Robustness and Safety in Hybrid Systems". In: *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*. Sept. 2014, pp. 1482–1488.

[CÖ17a] Michele Colledanchise and Petter Ögren. *Behavior Trees in Robotics and AI: An Introduction*. Chapman \& Hall/CRC Press, Aug. 2017.

[FUH16] Yunis Fanger, Jonas Umlauft and Sandra Hirche. "Gaussian Processes for Dynamic Movement Primitives with Application in Knowledge-Based Cooperation". In: *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Oct. 2016, pp. 3913–3919.

[FH05] Tamar Flash and Binyamin Hochner. "Motor Primitives in Vertebrates and Invertebrates". In: *Current opinion in neurobiology* 15.6 (2005), pp. 660–666.

[FUG11] Denis Forte, Aleš Ude and Andrej Gams. "Real-Time Generalization and Integration of Different Movement Primitives". In: *2011 11th IEEE-RAS International Conference on Humanoid Robots*. Oct. 2011, pp. 590–595.

[Gom+16] Sebastian Gomez-Gonzalez et al. "Using Probabilistic Movement Primitives for Striking Movements". In: *2016 IEEE-RAS 16th International Conference on Humanoid Robots (Humanoids)*. Nov. 2016, pp. 502–508.

[INS01] A. J. Ijspeert, J. Nakanishi and S. Schaal. "Trajectory Formation for Imitation with Nonlinear Dynamical Systems". In: *Proceedings 2001 IEEE/RSJ International Conference on Intelligent Robots and Systems. Expanding the Societal Role of Robotics in the the Next Millennium (Cat. No.01CH37180)*. Vol. 2. Oct. 2001, 752–757 vol.2.

[INS02] Auke Jan Ijspeert, Jun Nakanishi and Stefan Schaal. "Learning Rhythmic Movements by Demonstration Using Nonlinear Oscillators". In: *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS2002)*. CONF. 2002, pp. 958–963.

[Ijs+12] Auke Jan Ijspeert et al. "Dynamical Movement Primitives: Learning Attractor Models for Motor Behaviors". In: *Neural Computation* 25.2 (Nov. 2012), pp. 328–373.

[Lee+18] Jeongseok Lee et al. "DART: Dynamic Animation and Robotics Toolkit". In: *Journal of Open Source Software* 3.22 (Feb. 2018), p. 500.

[Mae+17] Guilherme J. Maeda et al. “Probabilistic Movement Primitives for Coordination of Multiple Human–Robot Collaborative Tasks”. In: *Autonomous Robots* 41.3 (Mar. 2017), pp. 593–612.

[MS22] Matthias Mayr and Julian M Salt-Ducaju. “A C++ Implementation of a Cartesian Impedance Controller for Robotic Manipulators”. In: *arXiv preprint arXiv:2212.11215* (2022). arXiv: 2212.11215.

[May+21] Matthias Mayr et al. “Learning of Parameters in Behavior Trees for Movement Skills”. In: *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2021 IEEE/RSJ International Conference on Intelligent Robots and Systems. IEEE, 2021, pp. 7572–7579.

[May+22a] Matthias Mayr et al. “Combining Planning, Reasoning and Reinforcement Learning to Solve Industrial Robot Tasks”. In: *IROS 2022 Workshop on Workshop on Trends and Advances in Machine Learning and Automated Reasoning for Intelligent Robots and Systems* (2022).

[May+22b] Matthias Mayr et al. “Learning Skill-Based Industrial Robot Tasks with User Priors”. In: *2022 IEEE 18th International Conference on Automation Science and Engineering (CASE)*. IEEE, 2022, pp. 1485–1492.

[May+22c] Matthias Mayr et al. “Skill-Based Multi-Objective Reinforcement Learning of Industrial Robot Tasks with Planning and Knowledge Integration”. In: *2022 IEEE International Conference on Robotics and Biomimetics (ROBIO)*. IEEE, 2022, pp. 1995–2002.

[MKP10] Katharina Muelling, Jens Kober and Jan Peters. “Learning Table Tennis with a Mixture of Motor Primitives”. In: *2010 10th IEEE-RAS International Conference on Humanoid Robots*. Dec. 2010, pp. 411–416.

[Mül+13] Katharina Mülling et al. “Learning to Select and Generalize Striking Movements in Robot Table Tennis”. In: *The International Journal of Robotics Research* 32.3 (Mar. 2013), pp. 263–279.

[Mus99] Ferdinando A Mussa-Ivaldi. “Modular Features of Motor Control and Learning”. In: *Current Opinion in Neurobiology* 9.6 (Dec. 1999), pp. 713–717.

[Nin+11] KeJun Ning et al. “Accurate Position and Velocity Control for Trajectories Based on Dynamic Movement Primitives”. In: *2011 IEEE International Conference on Robotics and Automation*. May 2011, pp. 5006–5011.

[Nin+12] KeJun Ning et al. “A Novel Trajectory Generation Method for Robot Control”. In: *Journal of Intelligent & Robotic Systems* 68.2 (Nov. 2012), pp. 165–184.

[Par+13] Alexandros Paraschos et al. “Probabilistic Movement Primitives”. In: *Advances in Neural Information Processing Systems*. Vol. 26. Curran Associates, Inc., 2013.

[RW06] Carl Edward Rasmussen and Christopher K. I. Williams. *Gaussian Processes for Machine Learning*. Adaptive Computation and Machine Learning. Cambridge, Mass: MIT Press, 2006.

[RGK17] F. Rovida, B. Grossmann and V. Krüger. “Extended Behavior Trees for Quick Definition of Flexible Robotic Tasks”. In: *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Sept. 2017, pp. 6793–6800.

[Rov+18] F. Rovida et al. “Motion Generators Combined with Behavior Trees: A Novel Approach to Skill Modelling”. In: *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Oct. 2018, pp. 5964–5971.

[Rov+16] Francesco Rovida et al. “Planning for Sustainable and Reliable Robotic Part Handling in Manufacturing Automation”. In: *Proceedings of the Twenty-Sixth International Conference on Automated Planning and Scheduling (ICAPS 2016)*. 2016.

[WA18] Roman Weitschat and Harald Aschemann. “Safe and Efficient Human–Robot Collaboration Part II: Optimal Generalized Human-in-the-Loop Real-Time Motion Generation”. In: *IEEE Robotics and Automation Letters* 3.4 (Oct. 2018), pp. 3781–3788.

[Wei+13] Roman Weitschat et al. “Dynamic Optimality in Real-Time: A Learning Framework for near-Optimal Robot Motions”. In: *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*. Nov. 2013, pp. 5636–5643.

[ZGA19] You Zhou, Jianfeng Gao and Tamim Asfour. “Learning Via-Point Movement Primitives with Inter- and Extrapolation Capabilities”. In: *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Nov. 2019, pp. 4301–4308.

[Zhu+97] Ciyou Zhu et al. “Algorithm 778: L-BFGS-B: Fortran Subroutines for Large-Scale Bound-Constrained Optimization”. In: *ACM Transactions on mathematical software (TOMS)* 23.4 (1997), pp. 550–560.

Paper VII

Paper 7

BeBOP – Combining Reactive Planning and Bayesian Optimization to Solve Robotic Manipulation Tasks

Jonathan Styrud
ABB Robotics
Royal Institute of Technology (KTH)
jonathan.styrud@se.abb.com

Matthias Mayr
Lund University
matthias.mayr@cs.lth.se

Erik Hellsten
Lund University
erik.hellsten@cs.lth.se

Volker Krueger
Lund University
volker.krueger@cs.lth.se

Christian Smith
Royal Institute of Technology (KTH)
ccs@kth.se

Abstract

Robotic systems for manipulation tasks are increasingly expected to be easy to configure for new tasks. While in the past, robot programs were often written statically and tuned manually, the current, faster transition times call for robust, modular and interpretable solutions that also allow a robotic system to learn how to perform a task. We propose the method Behavior-based Bayesian Optimization and Planning (*BeBOP*) that combines two approaches for generating behavior trees: we build the structure using a reactive planner and learn specific parameters with Bayesian optimization. The method is evaluated on a set of robotic manipulation benchmarks and is shown to outperform state-of-the-art reinforcement learning algorithms by being up to 46 times faster while simultaneously being less dependent on reward shaping. We also propose a modification to the uncertainty estimate for the random forest surrogate models that drastically improves the results.

1 Introduction

Modern robots are capable of solving complex tasks in controlled environments with high reliability and precision. However, recent trends are pointing towards smaller product batches and more frequent updates of robot programs. At the same time, the market share of collaborative robots is growing steadily, while workspaces shared with humans makes for more unpredictable environments. As a result, it is becoming increasingly important to create new robot policies or programs quickly without the need for advanced programming skills and for those programs to be reactive to changes in the environment. There are two main groups of methods to generate policies automatically, both with their own advantages and drawbacks. Firstly, automated planners [GNT16] can be very efficient, but require that the planning domain is modelled sufficiently well. As an example, a planner can only avoid obstacles that are represented in the domain. Planners also tend not to scale well to higher task complexity. The second group, colloquially known as Machine Learning (ML), typically builds a model by interacting with the environment and is thus not limited by preexisting knowledge. There are also cases where ML methods scale better than planners, as they can use a probabilistic approach instead of an exhaustive approach. However, the learning is often not very efficient and for smaller tasks, an automated planner can be many orders of magnitude faster. This hampers the use of ML-based methods as even state-of-the-art methods can take hours to days of interaction time to learn a new task. As an example, the *MAPLE* runs for the benchmarks in these paper takes several days on a normal workstation to learn. Another considerable drawback is that many of the ML algorithms, often in the Reinforcement Learning (RL) subgroup, are designed to use neural networks that are known to lack the transparency and modularity of other architectures.

An increasingly popular alternative in robotics is to instead represent the policy with Behavior Trees (BTs) [Iov+22; CÖ17a]. The main advantages are that BTs have explicit support for task hierarchy, action sequencing, reactivity and they are inherently modular. They are also transparent and readable, which enables manual and automated analysis and validation [CÖ17b] as well as manual editing. Those are strong advantages when compared to neural networks, especially in an industrial setting.

In this work we present Behavior-based Bayesian Optimization and Planning (*BeBOP*) that generates BTs by building a reactive tree structure using a planner and then subsequently learns the BT parameters with Bayesian Optimization (BO). With the tree structure as a prior, BO can then focus on tuning parameters that are difficult to plan and reason about. The method is evaluated

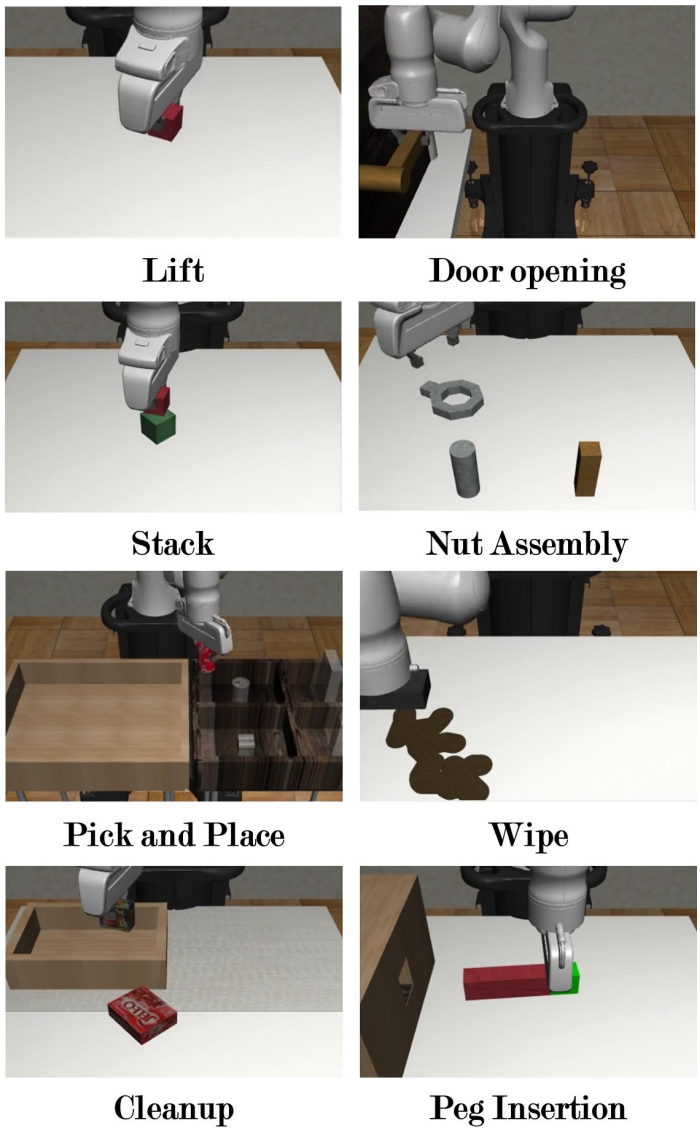


Figure 7.1: The eight simulation environments with the Franka Emika Robot (Panda). They range from easy tasks like a lifting a cube to sequential multi-step tasks like picking up a peg and inserting it into a hole.

in a simulation environment with eight different manipulation tasks.

It drastically outperforms the award-winning state-of-the-art RL algorithm *MAPLE* [NLZ22] in terms of the number of simulation steps needed to learn to solve the tasks while using exactly the same behavior primitives. By induction this also means that *BeBOP* is much more efficient than popular algorithms like *HIRO* [Nac+18] and *DAC* [ZW19]. The speedup could even enable training on real robot systems instead of just simulation as a lot fewer evaluations are needed. Furthermore, it is also shown that our method is less dependent on reward shaping in the form of affordances compared to the benchmark method.

Lastly, another advantage of the modular tree structure is that it also allows the task to be divided into subtasks and learned in sequence for even faster progress.

The main contributions of this paper are:

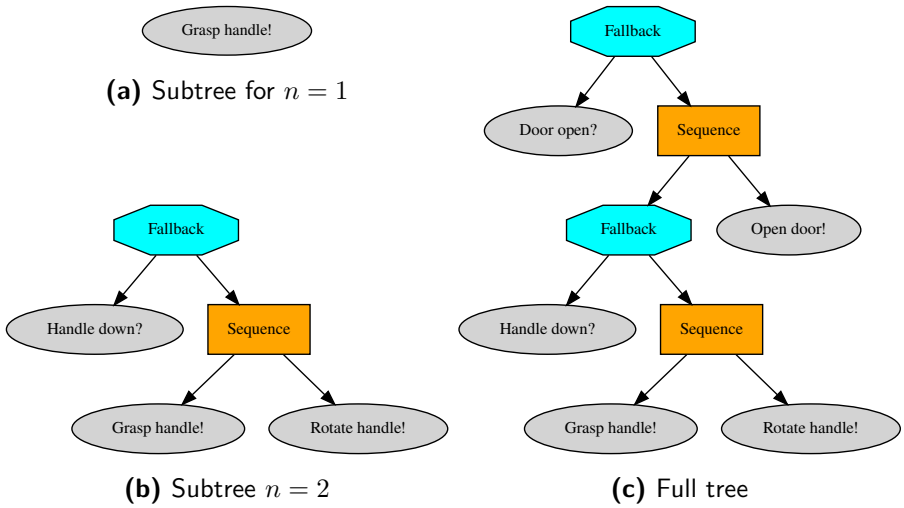
- A novel method, *BeBOP*, that combines reactive planning with efficient parameter tuning to yield state-of-the-art learning performance, while leading to an interpretable and robust policy.
- A new method to calculate the uncertainty of a random forest surrogate model within BO that outperforms the standard method on several tasks.
- A set of experiments verifying and validating our approach in comparison to the state-of-the-art RL method *MAPLE* [NLZ22], showing that *BeBOP* learns to solve the given tasks up to 46 times faster.

2 Background and Related Work

In this section we provide the relevant background on behavior trees and Bayesian optimization and discuss the related work.

2.1 Behavior Trees

Behavior Trees (BTs) were first used in the computer game industry, but have recently seen increased use in robotics [CÖ17a; Iov+22]. A BT is a directed tree where a tick signal propagates from the root node down to the leaves. The nodes are executed only when they receive the tick signal and return one of the states *Success*, *Failure* and *Running*. The non-leaf nodes are called *control flow nodes*. The flow nodes most commonly used are *Sequence*, which ticks



(a) Subtree for $n = 1$ **(b)** Subtree $n = 2$ **(c)** Full tree

Figure 7.2: Different subtrees for a door opening scenario. In a cascaded learning setup, the method starts by learning how to grasp the handle (a) before the tree gets extended by the handle rotation (b) and finally the full tree is constructed (c).

children sequentially from left to right, returning once all succeed or one fails, and *Fallback* (or *Selector*) which also runs sequentially but returns when one succeeds or all fail. Leaves are called *execution nodes* or *behaviors* and are usually separated into the types *Action*("!") and *Condition*("?"). Conditions encode status checks and sensory readings, only returning *Success* or *Failure* while actions encode robot skills that can take more than one tick to complete and therefore can also return *Running*. Figure 7.2 shows three example BTs for a door opening scenario.

The main advantages of BTs are that they are readable and have inherent support for task hierarchy, action sequencing and reactivity. They are also inherently modular [CÖ17a], in fact even optimally modular [BZS22]. The *Running* return state grants the reactivity property because a running action can be preempted by higher priority ones. BTs have been shown to improve on other representations, such as finite state machines, especially in terms of modularity and reactivity [Iov+23b; CÖ17b; BZS22].

2.2 Bayesian Optimization

In many practical optimization problems, there is no closed form expression available for the function to optimize. Instead, the user can only interact with the system by first selecting a configuration to evaluate and subsequently ob-

serving its performance. Often this black-box function is also expensive to evaluate; in the particular setting used in this paper, it amounts to running a simulation of a robot performing the specified task.

Bayesian optimization (BO) is a paradigm developed to efficiently optimize such problems, while limiting the number evaluations. It has recently shown great performance in a variety of applications, such as robotics [Cal+16; May+22c; Rai+18], hyperparameter tuning [Kle+17; Kan+18; Ru+20], and material design [FW16; Pac+17; Hug+21].

We consider the problem of finding a global maximum of an unknown black-box objective function f: $\mathbf{s}^* \in \arg\max_{\mathbf{s}\in\mathbb{S}} f(\mathbf{s})$, over some pre-specified domain $\mathbb{S}$ in D dimensions. The variables defining $\mathbb{S}$ can be real, integer, ordinal and categorical [NKO19]. We further assume that the evaluations of f are disturbed by observation noise and do not provide information on the function gradients.

BO is a sequential approach that iteratively selects new configurations to evaluate, trading off exploration and exploitation. It uses a surrogate model of the objective function and effectively learns the function as it gathers more data. The most common models are Gaussian Processes (GPs) [RW06] for their natural ability to quantify uncertainty on top of yielding accurate predictions and Random Forests (RFs) [Lin+22; Sha+16] for their versatility and scalability to a higher number of samples. Which configuration to select next is chosen by maximizing an acquisition function that quantifies the exploration-exploitation trade-off. Common examples are the *expected improvement* or *upper confidence bound*. For a more thorough introduction to BO, see [Fra18].

2.3 Related work

Various combinations of planning and RL have previously been proposed for other domains in [GK08; Fau+18; Fra+19; Moe+23]. In particular, [Sty+22] uses BTs as the underlying structure, and [Koz92; SG20] combine a planner with genetic programming. Using genetic programming to learn BTs has been done primarily for computer games [CPO18], but there are also examples for robotic manipulation applications [Iov+21; Sty+22; Iov+23a]. A more extensive analysis is given in [Iov+22]. The combination of a sequential planner and learning with BTs was also proposed in [May+22a; May+22c] for a peg-insertion and a pushing task. In [May+22b] it is shown how priors defined by operators or based on experience can accelerate learning and increase the safety during learning. As an extension, [AMK23] learns a GP model to generalize to task variations.

Regarding automated planners, we use an adaptation of the Planning Domain Definition Language (PDDL)-style planner from [CAÖ19] that creates BTs by leveraging backchaining. We build on the later adaptations of the same planner from [Sty+22; Gus+22; Iov+23a]. The main advantage of this planner is its simplicity, but there are other more advanced planners for BTs as well. Some examples are Linear Temporal Logic (LTL) [Tum+14; CMÖ17] and Hierarchical-Task-Network (HTN) planning [HG15; RGK17]. There are also other PDDL-style planners and we refer to Section 4.2 in [Iov+22] for a more exhaustive list.

We compare our method against RL with parameterized actions [MRK16; DPS21] - specifically with *MAPLE* [NLZ22]. Although these are completely different algorithms from ours, the types of problems they solve are essentially the same.

3 Approach

The key assumption in this work is that if there exists a set of parameterized actions that a robot can execute, these actions were likely designed with an intended use and effect on the robots environment. It is then possible to create a plan by using the actions under the assumption that for some values of the action's parameters, the actions will succeed and work in the intended way. Utilizing this, our proposed method consists of using a planner to obtain the structure of the BT and then employing a BO algorithm in an RL framework to tune the parameters of the nodes of the BT. The complete code of the planner and all other algorithms are available online[1].

3.1 Planner

In this paper we use a PDDL planner adapted from [CAÖ19] that was later extended in [Sty+22; Gus+22; Iov+23a]. As input to the planner, all behaviors have a set of preconditions that must be fulfilled in order to execute the behavior successfully and a set of postconditions that can be expected to be fulfilled when the behavior is done. A set of goal conditions defines the robot's task. Starting with goal conditions and proceeding backwards, the actions that complete the task or fulfill the necessary conditions for other actions are found iteratively and expanded until all conditions have been met. In this work, we improve the planner from [Gus+22] with some additions. Mainly, we trim the resulting tree

[1] `https://github.com/jstyrud/BeBOP`

by *a)* removing any control nodes with only a single child and by *b)* removing any post-condition nodes that are placed directly to before the corresponding action. The latter step assumes that all behaviors check for post-conditions internally before executing. We further introduce the concept of composite subtrees where, after planning, certain leaves can either be expanded into subtrees with multiple nodes or they can be replaced by some other parameterized behavior. This allows us to exploit the fact that, e.g., the behaviors *Reach* and *Open* together comprise a *Place* skill and that a *Reach* can generalize different behaviors at planning time such as opening a door or moving a grasped object.

3.2 Optimization

In order to optimize a policy, we follow the policy-search formulation [DNP13; Cha+19; Cha+17]. The goal is to find a policy $\pi, \mathbf{u} = \pi(\mathbf{x}|\boldsymbol{\theta})$ with policy parameters $\boldsymbol{\theta}$ such that we maximize the expected long-term reward when executing the policy for T time steps. Here, we use BO to tune the parameters. A given (planned) BT has a set of action nodes, such that each node can have zero or more parameters that can be learned for a given task. To construct a learning problem, we automatically examine a BT to obtain these adjustable parameters, their domains and dependencies.

We use a customized version of the BO implementation in *hypermapper* [NKO19] as it supports a wide range of variable types and user priors for the optimum [Car+22].

To provide a robust reward measure for the surrogate model and to prevent BO from overfitting to the training data, we evaluate each set of parameters in up to 20 episodes using *robotsuite's* domain randomization of the tasks with different seeds. We initially run each set of parameters for three episodes in different randomizations of the task simulation. After that, we estimate the variance after each episode and calculate the probability that this policy will outperform the current best policy. We then continue with more episodes as long as the probability is above 5 percent or until we have reached 20 episodes. We evaluate the parameter sets on the same random seeds for the training episodes to increase consistency of the evaluation results when training the surrogate model.

3.3 Improved Random Forest Surrogate

While GPs are the most common choice of surrogate models in BO, they assume a certain level of smoothness of the objective function, that is often not satisfied

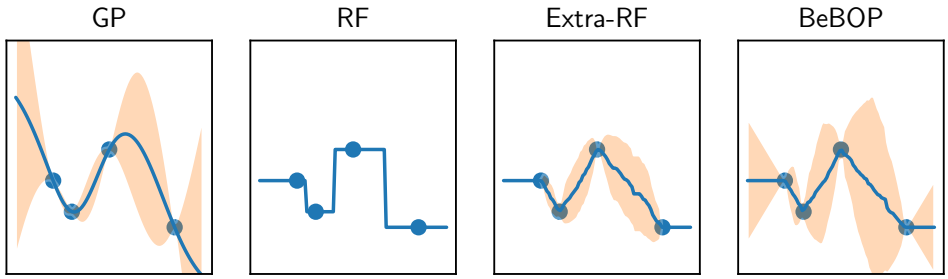


Figure 7.3: The predicted mean and standard deviation on a toy example for 4 different models: Gaussian Processes (GP), Random Forests (RFs), Extremely randomized RFs and our proposed uncertainty measure with an additional linear standard deviation term.

in the robotics tasks we consider. For example when inserting a peg into a hole, a millimeter change in offset in a movement primitive can result in a drastic difference in reward. Because of this, we instead use random forests which are more amenable to non-smooth objective functions. However, in contrast to GPs, RFs do not innately provide a variance estimate for its predictions, which is an essential building block in the BO selection process. Hutter et al. proposed the use of the empirical prediction variance across trees [HHL11], but this suffers from that the uncertainty does not inherently grow further away from previously observed data. This, in turn, hampers exploration.

To improve the performance of the optimization, we propose two adjustments to the RF model. First, we use extremely randomized trees which randomize among all optimal splits in each tree [GEW06]. This makes the prediction surface much smoother, which makes for a richer predicted function surface. This was previously used by, for example, [NKO19] and [Wu+22]. Secondly, we propose a new uncertainty metric, that extends the standard deviation estimate proposed by Hutter et al. by a term proportional to the distance to the closest previous observation. This incentivizes the optimization to continue exploring. To the best of our knowledge, this is the first paper to suggest such a modification in a BO setting. In Fig. 7.3, we show the impact of the modifications. As we will see in Section 5, this significantly improves the results.

3.4 Combining Planning and Bayesian Optimization

To combine the planner with the BO algorithm, we first run the planner on each task to obtain the BT structure. The planner, however, leaves a number

of parameters unsolved for each behavior, and during planning it assumes that there exists some set of parameters for which the behaviors will succeed. These parameters are then given as input to BO. For each parameter, the behaviors also specify an upper and lower bound and these limits should be grounded in the actual application.

Cascaded Learning: We also note that because of the hierarchical nature of the BTs that are output from the planner, the BTs can be divided into a number of subtrees that can be run sequentially with gradually larger subtrees, representing subtasks. For example, grasping would be a sub-task of moving an object. In this way, we can learn the smaller subtask first, using parameters from the solution as priors for the optimum to the next, larger subtree. This way we can potentially speed up the optimization as the learning time typically scales super-linearly with the number of parameters. We call this version of our method *cascaded BeBOP*. Starting with $n = 1$, to find subtree n we start with the first action node and traverse the tree left to right, depth first, counting the action nodes. Continue up to but not including the action node $n + 1$. The last node before the action node $n + 1$ will be the last node in the subtree. Action nodes without free parameters are omitted, as they do not increase the complexity of our optimization problem. All subtrees for smaller n will also be included in subtree n. Fig. 7.2 shows examples of the resulting subtrees for a door opening scenario. Note that the subtree for $n = 1$ consists of only one node and that the behaviors in the figure are only aliases during planning of more generic behaviors, as listed in Section 4.1. For every n-th subtree, we run the BO in batches of 50 iterations until no improvement is seen since the last batch. We then use the best solution found as priors and run BO on the subtree $n+1$. Splitting learning tasks into subtrees has also been proposed in [May+21], but a) without an automated procedure to do this and b) while keeping the previously learned parameters fixed when combining trees. The advantage we get by *not* fixing the parameters of previous subtrees is that the optimal values might be different in the context of the complete tree.

4 Experimental Setup

We benchmark our algorithm using *robosuite* [Zhu+22] in the eight simulated robot manipulation scenarios shown in Fig. 7.1 that are used in *MAPLE* [NLZ22], the method that we compare against. One feature of *robosuite* is that it supports slight domain randomizations of the task environments to allow for the evaluation of the robustness of policies. In the *Door* scenario in the original

benchmark the robot was incapable of grasping the door handle in all instances of domain randomization. Therefore we made a small change by allowing the gripper to rotate in the same way as in the other tasks. This change had no noticeable impact on the performance of *MAPLE* on the task.

In this paper, we refrained from performing additional validation of the resulting policies learned in simulation on real robot systems. It has been shown previously, including in the referred *MAPLE* paper which uses the same behaviors [NLZ22], that policies acting at this abstraction level are easily transferable from simulation [Sty+22; May+21; May+22a; May+22c; May+22b; AMK23].

4.1 Behaviors

We use the same action primitives as *MAPLE* and only communicate with the simulation using the same action and observation vectors as the neural networks in the original benchmarks. We wrap the input and output vectors of the simulation with behaviors that can be used by a behavior tree. The five behaviors correspond to the five primitives used by *MAPLE* and call the corresponding parameterized behavior primitives when executing:

- **Reach:** The robot moves to some position relative to an object or the origin of the coordinate frame. The offset (x, y, z) is specified by the behavior parameters.
- **Grasp:** The robot moves to some position relative to a graspable object and closes the gripper. The offset (x, y, z) and yaw angle are the behavior parameters.
- **Push:** The robot attempts to push an object towards an x, y position by moving to the opposing side of the object and pushing toward the goal position. The behavior parameters are the coordinates x and y.
- **Open:** The robot opens the gripper. No parameters.
- **Atomic:** The robot applies an atomic action for one step, moving a delta calculated from the relative distance between some object position and some (x, y, z) and yaw angle as specified by the behavior parameters.

In addition to behaviors, the BT also needs conditions that process the information in the observation vector. In the experiments, we make use of three different condition nodes.

- **At:** Returns *Success* if the object is at a given position. This is exclusively used for goal conditions and the parameters are therefore fixed.
- **Angle >:** Checks if the angle of some object is larger than a value parameter.
- **Aligned:** Checks the observation vector to see if the object is aligned. No parameters except the object identifier. Used only in the peg-insertion task.

We use pre- and post-conditions of the behaviors for planning and because the behaviors work with relative coordinates, we claim that in most real robot applications and frameworks, such as [MRK23], this knowledge will be readily available. The BTs are implemented using the *PyTrees* framework[2].

4.2 Reward

The reward for the episodes is the same dense rewards with affordances as the original benchmark [NLZ22]. However, since BTs have a natural stop condition when the tree returns *Success* or *Failure*, we can also handle that. We want to avoid solutions where the tree returns *Failure*, so we add a negative reward of -500.0 for any such episode. If the BT stops before the maximum number of steps in the environment, we assume that the same reward that was given for the last step would be given for the rest of the episode and extrapolate it until the maximum number of steps. As in [NLZ22] we evaluate the policy in 20 validation episodes with different random seeds than during training. We only validate the currently best BT found, based on the reward of the training episodes. The affordances and failure penalty are not used for the validation. The affordance penalty as described in [NLZ22] is a form of reward shaping where a penalty is given for an action that is performed outside a manually specified region. In [NLZ22] it is shown that *MAPLE* is unable to make progress even on the simple *Pick and Place* task without the affordances.

[2] `https://github.com/splintered-reality/py_trees`, version 2.2.2. Specifically, we use a forked version with slightly changed visuals: `https://github.com/jstyrud/py_trees`

5 Results

Figure 7.4 shows learning curves for all eight tasks as the mean of five separate repetitions (same as in [NLZ22]) for each method with shaded areas denoting the standard deviation. We run the experiments until the task is solved, but maximally 10^6 time steps. The figure shows that our method outperforms *MAPLE* in 7 of the 8 benchmarks. The number of steps needed to solve a task is often an order of magnitude less than for *MAPLE*. By outperforming *MAPLE* [NLZ22] we consequently also outperform other RL algorithms like *HIRO* [Nac+18] and *DAC* [ZW19] which are not shown here but are discussed in detail in [NLZ22]. We also note that our new uncertainty measure as described in Section 3.2 performs equally or better than the standard RF uncertainty measure (green) for all benchmark tasks except *Stack*. We believe that this is simply because *Stack* is easy enough to solve without it. However, BO with the standard measure fails to solve the more difficult tasks even with more iterations.

The cascaded version of *BeBOP* (orange) performs even better on several benchmarks, especially those that can be logically divided into a sequence of subtasks.

For the *Wipe* task none of the methods, including *MAPLE*, are able to solve the task. It is likely that the allowed time is in fact insufficient to perform the task reliably. The markers to wipe are randomly placed, and it is possible that MAPLE can statistically learn their positions, giving it a slight edge in this benchmark. However, with the only observation of the markers being their average position, not enough information is given to make an efficient wiping pattern.

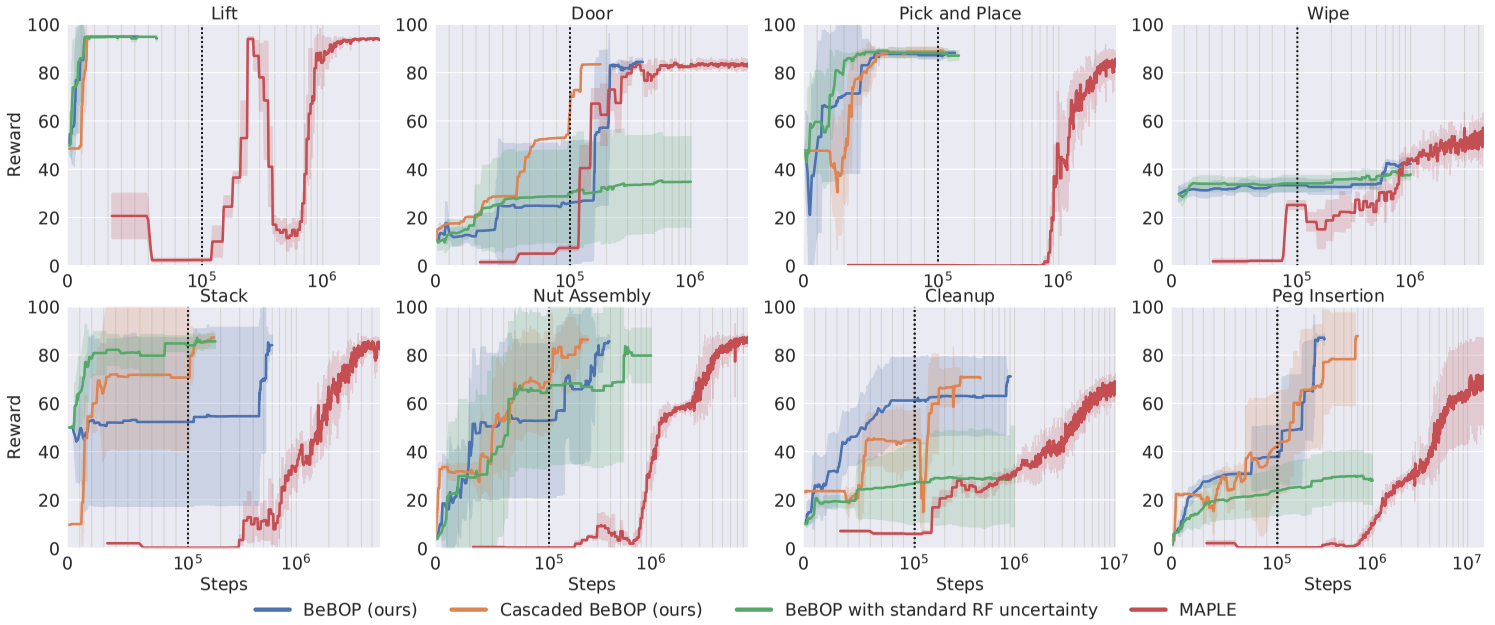


Figure 7.4: Episodic reward learning curves for the eight tasks. The x-axis is linear until 10^5 steps and logarithmic thereafter as marked by the dotted vertical line. The results highlight the importance of our new uncertainty measure and that our approaches outperform *MAPLE* by a large margin.

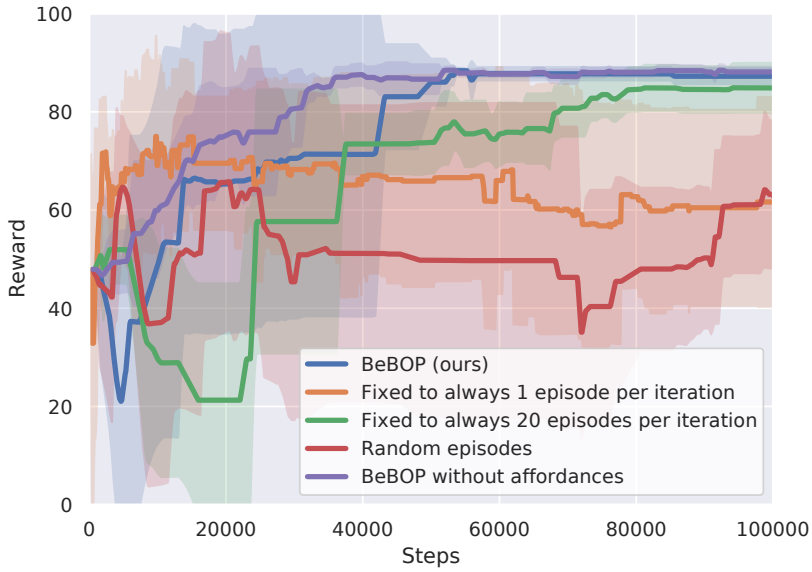


Figure 7.5: The reward of the best policy on the validation data for the *Pick and Place task*. It shows that our evaluation strategy (blue) does not only learn the fastest and most robust, but our method can also learn without the special affordances (purple).

We also studied how the evaluation procedure impacts *BeBOP*'s performance. In Fig. 7.5 we see that solving the *Pick and Place* task without affordances (purple) has no significant impact on the learning rate, while *MAPLE* [NLZ22] was reported to make no progress at all without affordances. In the same figure, we show learning curves for different variants of choosing the training episodes per parameter set. Fixing the number of episodes to 1 (orange) quickly finds a good solution on the training data, but fails to generalize to the validation episodes. Fixing it to always run 20 (green) will generalize to the validation episodes but wastes a lot of steps on poor solutions which results in slow learning. Choosing 20 random environments for evaluations can mean that a policy gets a set of easy or hard environments to evaluate in, which makes a comparison difficult. Such inconsistencies also makes it harder to fit the BO surrogate model. The red line in Fig. 7.5 shows how this negatively affects the learning performance. Finally, our proposed iterative evaluation shown in blue learns fast and achieves robust results.

In Fig. 7.6 we show the actual speedup factor to reach the specified task success rates for *BeBOP* and *Cascaded BeBOP* compared to *MAPLE* for the 7 tasks that were solved. The plot compares the point where the mean success rate of the five repetitions has reached 95% for the first time. In our experiments *MAPLE* failed to solve *Peg Insertion* task in one of the repetitions, so

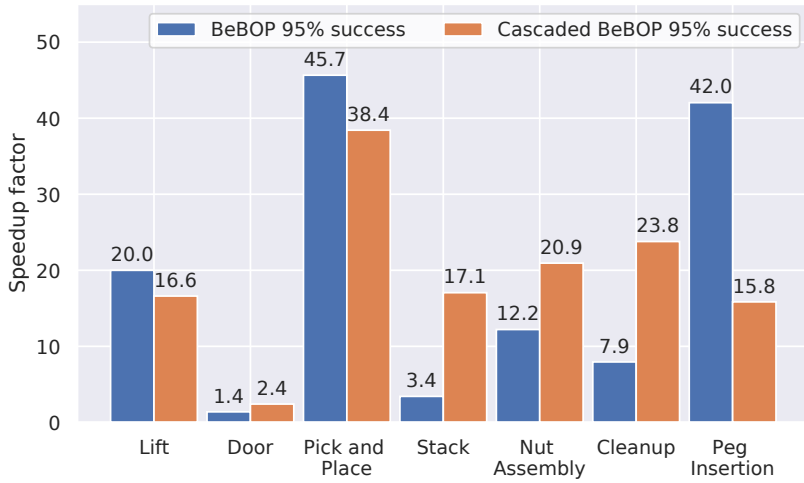


Figure 7.6: The speedup factor when compared to *MAPLE* on the 7 tasks that are solved. The factor is computed at reaching a mean 95% success rate in the validation data.

we only use the other four. As shown in the figure, even the smallest speedup of *BeBOP* compared to *MAPLE* in the *Door* task is still 1.4. For many tasks, *BeBOP* learns more than 15 times faster and reaches a speedup of up to 46 times. This not only saves a lot of compute time when learning in simulation, but also makes learning on a real robotic system much more realistic.

6 Conclusions

We present *BeBOP*, a method for combining reactive task planning and Bayesian optimization to create behavior-tree policies. We show that our method for learning these policies outperforms state-of-the-art RL algorithms such as *MAPLE, HICO* and *DAC* by a large margin. While using exactly the same behavior primitives, our method solves the robotic manipulation benchmarks using on average only about 5% of the steps needed by *MAPLE*. Those results are further improved by utilizing the structure of the behaviour trees to divide the task into a sequence of sub-tasks, which makes *BeBOP* solve many of the benchmarks even faster. An ablation analysis indicates that *BeBOP* is also less dependent on reward shaping in the form of special affordances compared to the benchmark method. Our newly introduced uncertainty measure for the random forest surrogate model accelerates and robustifies the learning with Bayesian optimization in robotics tasks significantly.

At the same time, the obtained BT policies are deterministic as well as inter-

pretable and modifyable by humans. This makes them much more attractive for sensitive environments such as in industrial manufacturing or private households.

7 Future work

One natural direction for future work is to use a more advanced planner and to combine it with platforms that support reasoning such as [MRK23; May+23], as both leave less for the optimization algorithm to learn. In addition, it would be interesting to study the possibility of re-planning the BT structure in case of changes in the environment without having to re-learn all parameters from scratch.

We believe that our method can generalize to many other types of task, perhaps also outside the robotic domain and confirming that would be an intuitive next step.

Certain tasks are well suited for neural networks, such as when decisions need to factor in many different variables. In those cases, it could be a good approach to to include a network as part of a BT policy. That way, the advantages of a transparent and modular BT are kept for the majority of the policy, while still enjoying the strength of the neural network [SÖ22].

References

[AMK23] Faseeh Ahmad, Matthias Mayr and Volker Krueger. “Learning to Adapt the Parameters of Behavior Trees and Motion Generators (BTMGs) to Task Variations”. In: *2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2023.

[BZS22] Oliver Biggar, Mohammad Zamani and Iman Shames. “On Modularity in Reactive Control Architectures, with an Application to Formal Verification”. In: *ACM Transactions on Cyber-Physical Systems* 6.2 (Apr. 2022), 19:1–19:36.

[Cal+16] Roberto Calandra et al. “Bayesian Optimization for Learning Gaits under Uncertainty: An Experimental Comparison on a Dynamic Bipedal Walker”. In: *Annals of Mathematics and Artificial Intelligence* 76.1-2 (Feb. 2016), pp. 5–23.

[Car+22] Carl Hvarfner et al. “piBO: Augmenting Acquisition Functions with User Beliefs for Bayesian Optimization”. In: *International Conference on Learning Representations*. 2022.

[Cha+17] Konstantinos Chatzilygeroudis et al. “Black-Box Data-Efficient Policy Search for Robotics”. In: *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Sept. 2017, pp. 51–58.

[Cha+19] Konstantinos Chatzilygeroudis et al. “A Survey on Policy Search Algorithms for Learning Robot Controllers in a Handful of Trials”. In: *IEEE Transactions on Robotics* (2019), pp. 1–20.

[CPO18] M. Colledanchise, R. Nattanmai Parasuraman and P. Ogren. “Learning of Behavior Trees for Autonomous Agents”. In: *IEEE Transactions on Games* (2018), pp. 1–1.

[CAÖ19] Michele Colledanchise, Diogo Almeida and Petter Ögren. “Towards Blended Reactive Planning and Acting Using Behavior Trees”. In: *2019 International Conference on Robotics and Automation (ICRA)*. May 2019, pp. 8839–8845.

[CMÖ17] Michele Colledanchise, Richard M. Murray and Petter Ögren. “Synthesis of Correct-by-Construction Behavior Trees”. In: *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Sept. 2017, pp. 6039–6046.

[CÖ17a] Michele Colledanchise and Petter Ögren. *Behavior Trees in Robotics and AI: An Introduction*. Chapman \& Hall/CRC Press, Aug. 2017.

[CÖ17b] Michele Colledanchise and Petter Ögren. “How Behavior Trees Modularize Hybrid Control Systems and Generalize Sequential Behavior Compositions, the Subsumption Architecture, and Decision Trees”. In: *IEEE Transactions on Robotics* 33.2 (Apr. 2017), pp. 372–389.

[DPS21] Murtaza Dalal, Deepak Pathak and Russ R Salakhutdinov. “Accelerating Robotic Reinforcement Learning via Parameterized Action Primitives”. In: *Advances in Neural Information Processing Systems*. Vol. 34. Curran Associates, Inc., 2021, pp. 21847–21859.

[DNP13] Marc Peter Deisenroth, Gerhard Neumann and Jan Peters. “A Survey on Policy Search for Robotics”. In: *Foundations and Trends® in Robotics* 2.1–2 (Aug. 2013), pp. 1–142.

[Fau+18] Aleksandra Faust et al. “PRM-RL: Long-range Robotic Navigation Tasks by Combining Reinforcement Learning and Sampling-Based Planning”. In: *2018 IEEE International Conference on Robotics and Automation (ICRA)*. May 2018, pp. 5113–5120.

[Fra+19] Vincent Francois-Lavet et al. “Combined Reinforcement Learning via Abstract Representations”. In: *Proceedings of the AAAI Conference on Artificial Intelligence* 33.01 (July 2019), pp. 3582–3589.

[Fra18] Peter I. Frazier. *A Tutorial on Bayesian Optimization*. July 2018. arXiv: 1807.02811 [cs, math, stat].

[FW16] Peter I. Frazier and Jialei Wang. “Bayesian Optimization for Materials Design”. In: *Information Science for Materials Discovery and Design*. Ed. by Turab Lookman, Francis J. Alexander and Krishna Rajan. Springer Series in Materials Science. Cham: Springer International Publishing, 2016, pp. 45–75.

[GEW06] Pierre Geurts, Damien Ernst and Louis Wehenkel. “Extremely Randomized Trees”. In: *Machine Learning* 63.1 (Apr. 2006), pp. 3–42.

[GNT16] Malik Ghallab, Dana Nau and Paolo Traverso. *Automated Planning and Acting*. Cambridge University Press, Aug. 2016.

[GK08] Matthew Grounds and Daniel Kudenko. “Combining Reinforcement Learning with Symbolic Planning”. In: *Adaptive Agents and Multi-Agent Systems III. Adaptation and Multi-Agent Learning*. Ed. by Karl Tuyls et al. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2008, pp. 75–86.

[Gus+22] Oscar Gustavsson et al. “Combining Context Awareness and Planning to Learn Behavior Trees from Demonstration”. In: *2022 31st IEEE International Conference on Robot and Human Interactive Communication (RO-MAN)*. Aug. 2022, pp. 1153–1160.

[HG15] Matthias Hölzl and Thomas Gabor. “Reasoning and Learning for Awareness and Adaptation”. In: *Software Engineering for Collective Autonomic Systems: The ASCENS Approach*. Ed. by Martin Wirsing et al. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2015, pp. 249–290.

[Hug+21] Zak E Hughes et al. “Tuning Materials-Binding Peptide Sequences toward Gold-and Silver-Binding Selectivity with Bayesian Optimization”. In: *ACS nano* 15.11 (2021), pp. 18260–18269.

[HHL11] Frank Hutter, Holger H. Hoos and Kevin Leyton-Brown. “Sequential Model-Based Optimization for General Algorithm Configuration”. In: *Learning and Intelligent Optimization*. Ed. by Carlos A. Coello Coello. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2011, pp. 507–523.

[Iov+21] Matteo Iovino et al. “Learning Behavior Trees with Genetic Programming in Unpredictable Environments”. In: *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2021, pp. 4591–4597.

[Iov+22] Matteo Iovino et al. “A Survey of Behavior Trees in Robotics and AI”. In: *Robotics and Autonomous Systems* 154 (Aug. 2022), p. 104096.

[Iov+23a] Matteo Iovino et al. “A Framework for Learning Behavior Trees in Collaborative Robotic Applications”. In: *2023 IEEE International Conference on Automation Science and Engineering (CASE)* (2023).

[Iov+23b] Matteo Iovino et al. “On the Programming Effort Required to Generate Behavior Trees and Finite State Machines for Robotic Applications”. In: *2023 IEEE International Conference on Robotics and Automation (ICRA)*. London, United Kingdom: IEEE, May 2023, pp. 5807–5813.

[Kan+18] Kirthevasan Kandasamy et al. “Neural Architecture Search with Bayesian Optimisation and Optimal Transport”. In: *Advances in Neural Information Processing Systems*. Vol. 31. Curran Associates, Inc., 2018.

[Kle+17] Aaron Klein et al. “Fast Bayesian Optimization of Machine Learning Hyperparameters on Large Datasets”. In: *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics*. PMLR, Apr. 2017, pp. 528–536.

[Koz92] John R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, 1992.

[Lin+22] Marius Lindauer et al. “SMAC3: A Versatile Bayesian Optimization Package for Hyperparameter Optimization”. In: *The Journal of Machine Learning Research* 23.1 (Jan. 2022), 54:2475–54:2483.

[MRK16] Warwick Masson, Pravesh Ranchod and George Konidaris. “Reinforcement Learning with Parameterized Actions”. In: *Proceedings of the AAAI Conference on Artificial Intelligence* 30.1 (Feb. 2016).

[MRK23] Matthias Mayr, Francesco Rovida and Volker Krueger. “SkiROS2 - a Skill-Based Robot Control Platform for ROS”. In: *2023 IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2023.

[May+21] Matthias Mayr et al. “Learning of Parameters in Behavior Trees for Movement Skills”. In: *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2021 IEEE/RSJ International Conference on Intelligent Robots and Systems. IEEE, 2021, pp. 7572–7579.

[May+22a] Matthias Mayr et al. “Combining Planning, Reasoning and Reinforcement Learning to Solve Industrial Robot Tasks”. In: *IROS 2022 Workshop on Workshop on Trends and Advances in Machine Learning and Automated Reasoning for Intelligent Robots and Systems* (2022).

[May+22b] Matthias Mayr et al. “Learning Skill-Based Industrial Robot Tasks with User Priors”. In: *2022 IEEE 18th International Conference on Automation Science and Engineering (CASE)*. IEEE, 2022, pp. 1485–1492.

[May+22c] Matthias Mayr et al. “Skill-Based Multi-Objective Reinforcement Learning of Industrial Robot Tasks with Planning and Knowledge Integration”. In: *2022 IEEE International Conference on Robotics and Biomimetics (ROBIO)*. IEEE, 2022, pp. 1995–2002.

[May+23] Matthias Mayr et al. “Using Knowledge Representation and Task Planning for Robot-Agnostic Skills on the Example of Contact-Rich Wiping Tasks”. In: *2023 IEEE 19th International Conference on Automation Science and Engineering (CASE)*. IEEE, 2023, pp. 1–7.

[Moe+23] Thomas M. Moerland et al. “Model-Based Reinforcement Learning: A Survey”. In: *Foundations and Trends® in Machine Learning* 16.1 (Jan. 2023), pp. 1–118.

[Nac+18] Ofir Nachum et al. “Data-Efficient Hierarchical Reinforcement Learning”. In: *Advances in Neural Information Processing Systems*. Vol. 31. Curran Associates, Inc., 2018.

[NKO19] Luigi Nardi, David Koeplinger and Kunle Olukotun. “Practical Design Space Exploration”. In: *2019 IEEE 27th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*. Oct. 2019, pp. 347–358.

[NLZ22] Soroush Nasiriany, Huihan Liu and Yuke Zhu. “Augmenting Reinforcement Learning with Behavior Primitives for Diverse Manipulation Tasks”. In: *2022 International Conference on Robotics and Automation (ICRA)*. IEEE, 2022, pp. 7477–7484.

[Pac+17] Daniel Packwood et al. *Bayesian Optimization for Materials Science*. Springer, 2017.

[Rai+18] Akshara Rai et al. “Bayesian Optimization Using Domain Knowledge on the ATRIAS Biped”. In: *2018 IEEE International Conference on Robotics and Automation (ICRA)*. May 2018, pp. 1771–1778.

[RW06] Carl Edward Rasmussen and Christopher K. I. Williams. *Gaussian Processes for Machine Learning*. Adaptive Computation and Machine Learning. Cambridge, Mass: MIT Press, 2006.

[RGK17] F. Rovida, B. Grossmann and V. Krüger. “Extended Behavior Trees for Quick Definition of Flexible Robotic Tasks”. In: *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Sept. 2017, pp. 6793–6800.

[Ru+20] Binxin Ru et al. “Interpretable Neural Architecture Search via Bayesian Optimisation with Weisfeiler-Lehman Kernels”. In: *International Conference on Learning Representations*. Oct. 2020.

[Sha+16] Bobak Shahriari et al. “Taking the Human Out of the Loop: A Review of Bayesian Optimization”. In: *Proceedings of the IEEE* 104.1 (Jan. 2016), pp. 148–175.

[SG20] Andrew N. Sloss and Steven Gustafson. “2019 Evolutionary Algorithms Review”. In: *Genetic Programming Theory and Practice XVII*. Ed. by Wolfgang Banzhaf et al. Genetic and Evolutionary Computation. Cham: Springer International Publishing, 2020, pp. 307–344.

[SÖ22] Christopher Iliffe Sprague and Petter Ögren. “Adding Neural Network Controllers to Behavior Trees without Destroying Performance Guarantees”. In: *2022 IEEE 61st Conference on Decision and Control (CDC)*. Dec. 2022, pp. 3989–3996.

[Sty+22] Jonathan Styrud et al. “Combining Planning and Learning of Behavior Trees for Robotic Assembly”. In: *2022 International Conference on Robotics and Automation (ICRA)*. May 2022, pp. 11511–11517.

[Tum+14] Jana Tumova et al. “Maximally Satisfying LTL Action Planning”. In: *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*. Sept. 2014, pp. 1503–1510.

[Wu+22] Xingfu Wu et al. “Autotuning PolyBench Benchmarks with LLVM Clang/Polly Loop Optimization Pragmas Using Bayesian Optimization”. In: *Concurrency and Computation: Practice and Experience* 34.20 (2022), e6683.

[ZW19] Shangtong Zhang and Shimon Whiteson. “DAC: The Double Actor-Critic Architecture for Learning Options”. In: *Advances in Neural Information Processing Systems*. Vol. 32. Curran Associates, Inc., 2019.

[Zhu+22] Yuke Zhu et al. *Robosuite: A Modular Simulation Framework and Benchmark for Robot Learning*. Nov. 2022. arXiv: 2009.12293 [cs].