Towards self-reliant robots

skill learning, failure recovery, and real-time adaptation: integrating behavior trees, reinforcement learning, and vision-language models for robust robotic autonomy

Ahmad, Faseeh

2025

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

Towards Self-Reliant Robots: Skill Learning, Failure Recovery, and Real-Time Adaptation

Towards Self-Reliant Robots: Skill Learning, Failure Recovery, and Real-Time Adaptation

Integrating Behavior Trees, Reinforcement Learning, and Vision-Language Models for Robust Robotic Autonomy

by Faseeh Ahmad

LUND
UNIVERSITY

Thesis for the degree of Computer Science
Thesis advisors: Prof. Dr. Volker Krueger, Prof. Dr. Jacek Malec
Faculty opponent: Prof. Dr. Lazaros Nalpantidis

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 10th of October 2025 at 13:00.

Organization	Document name
LUND UNIVERSITY Department of Computer Science Box 118 SE–221 00 LUND Sweden	**DOCTORAL DISSERTATION** Date of disputation 10-10-2025 Sponsoring organizationWallenberg AI, Autonomous Systems and Software Program (WASP)
Author(s) Faseeh Ahmad	

Title and subtitle

Towards Self-Reliant Robots: Skill Learning, Failure Recovery, and Real-Time Adaptation: Integrating Behavior Trees, Reinforcement Learning, and Vision-Language Models for Robust Robotic Autonomy

Abstract

Robots operating in real-world settings must manage task variability, environmental uncertainty, and failures during execution. This thesis presents a unified framework for building self-reliant robotic systems by integrating symbolic planning, reinforcement learning, behavior trees (BTs), and vision-language models (VLMs).

At the core of the approach is an interpretable policy representation based on behavior trees and motion generators (BTMGs), supporting both manual design and automated parameter tuning. Multi-objective Bayesian optimization enables learning skill parameters that balance performance metrics such as safety, speed, and task success. Policies are trained in simulation and successfully transferred to real robots for contact-rich manipulation tasks.

To support generalization, the framework models task variations using gaussian processes, enabling interpolation of BTMG parameters across unseen scenarios. This allows adaptive behavior without retraining for each new task instance.

Failure recovery is addressed through a hierarchical scheme. BTs are extended with a reactive planner that dynamically updates execution policies based on runtime observations. Vision-language models assist in detecting and identifying failures, and in generating symbolic corrections when tasks are predicted to fail.

The thesis concludes with a discussion of future work, including (1) using vision-language-action (VLA) models or diffusion policies to generate new skills on the fly from multimodal inputs, and (2) extending the reactive planner with proactive failure prediction to anticipate and prevent execution errors before they occur. Together, these directions aim to advance robotic systems that are more robust, adaptable, and autonomous.

Key words

Autonomous Robotics, Behavior Trees, Reinforcement Learning, Vision-Language Models, Failure Recovery

Classification system and/or index terms (if any)

Supplementary bibliographical information	Language
	English
ISSN and key title 1404-1219	ISBN 978-91-8104-681-6 (print) 978-91-8104-682-3 (pdf)
Recipient's notes	Number of pages 258 / Price
	Security classification

Towards Self-Reliant Robots: Skill Learning, Failure Recovery, and Real-Time Adaptation

Integrating Behavior Trees, Reinforcement Learning, and Vision-Language Models for Robust Robotic Autonomy

by Faseeh Ahmad

LUND
UNIVERSITY

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

Faculty of Engineering (LTH), Department of Computer Science

ISBN: 978-91-8104-681-6 (print)
ISBN: 978-91-8104-682-3 (pdf)
ISSN: 1404-1219
Dissertation: 83, 2025
LU-CS-DISS: 2025-05

Printed in Sweden by Media-Tryck, Lund University, Lund 2025

Media-Tryck is a Nordic Swan Ecolabel certified provider of printed material. Read more about our environmental work at www.mediatryck.lu.se

MADE IN SWEDEN

Dedicated to my family

Contents

List of publications vi
Acknowledgements ix
Popular summary in English xi
Populärvetenskaplig sammanfattning på svenska xiii

Towards Self-Reliant Robots: Skill Learning, Failure Recovery, and Real-Time Adaptation **1**

1 Introduction 1
1.1 Overview of Aims and Research Questions 3
1.2 Thesis Outline 6

Background and Related Work 7

2 Skill Representation and Execution Frameworks 8
2.1 Behavior Trees 8
2.2 Reactive Planning for BT Generation and Extension 11
2.3 Motion Generators 14
2.4 BTMG Framework: Integrating Behavior Trees with Motion Generators 17
2.5 Symbolic Planning and Skill-Based Task Execution 18
3 Learning and Adaptation for Robotics 21
3.1 Learning Policies in Robotics 21
3.2 Bayesian Optimization for Efficient Policy Search 24
3.3 Multi-objective Optimization and Reward Design 25
3.4 Gaussian Processes for Policy Generalization and Adaptation . . . 28
4 Failure Recovery in Robotics 30
4.1 Failure Models and Detection 30
4.2 Failure Recovery Strategies: Reactive *vs.* Proactive 32
4.3 Recovery-Behavior Synthesis 33
4.4 Runtime Monitoring and Adaptation 35
5 Vision-Language Models for Robot Reasoning 37
5.1 Technical Foundations of Vision-Language Models (VLMs) . . . 37
5.2 Applications in Robotics 38
5.3 Scene Understanding and Goal Inference 40

5.4 Failure Detection and Explanation with VLMs 40

Part I – Adaptive Skill Learning for Robot Autonomy **43**

6 Structuring Robot Policies for Modularity, Interpretability, and Data-Efficient Learning 43
6.1 Motivation and Positioning within Existing Work 43
6.2 Structuring Policies with BTMG and Parameter Optimization . . 45
6.3 Method Overview 46
6.4 Experimental Evaluation 48
6.5 Discussion 51
7 Generalizing Modular Policies 52
7.1 Motivation and Research Framing 52
7.2 Approach: Gaussian Process-Based Parameter Generalization . . . 53
7.3 Experimental Setup 54
7.4 Discussion 55
8 Real-Time Prediction of Policy Parameters 57
8.1 Motivation and Research Framing 57
8.2 Approach: Surrogate Inference for Real-Time Parameter Prediction 57
8.3 Experimental Setup and Evaluation 60
8.4 Discussion 61

Part II – Failure Detection, Explanation, and Recovery **65**

9 Structuring Modular and Adaptive Recovery Behaviors 65
9.1 Motivation and Positioning within Existing Work 65
9.2 Modeling Recovery in the BTMG Framework 66
9.3 Failure Cases and Recovery Behavior Design 68
9.4 Execution Scenarios and Learning Setup 69
9.5 Discussion 71
10 Pre-Execution Failure Handling with Vision–Language Models 72
10.1 Motivation and Positioning within Existing Work 72
10.2 Approach Overview 74
10.3 Illustrative Scenarios 76
10.4 Evaluation and Findings 78
10.5 Discussion 79
11 Real-Time Monitoring and Reactive Recovery 80
11.1 Motivation and Positioning within Existing Work 80
11.2 Approach Overview 81
11.3 Visual Pipeline and Scene Graph Maintenance 84
11.4 Demonstrated Scenarios 85
11.5 Discussion 88

Conclusions **91**

Scientific publications 109
Contribution Statements . . . 109

1 Skill-based Multi-objective Reinforcement Learning of Industrial Robot Tasks with Planning and Knowledge Integration 115
1 Abstract . . . 115
2 Introduction . . . 116
3 Related Work . . . 118
3.1 Skill-based Systems . . . 118
3.2 Policy Representation and Learning . . . 118
3.3 Planning and Learning . . . 119
4 Approach . . . 119
4.1 Behavior Trees . . . 120
4.2 Planning and Knowledge Integration . . . 121
4.3 Policy Optimization . . . 122
4.4 Bayesian Optimization . . . 123
4.5 Multi-objective Optimization . . . 124
4.6 Motion Generator and Robot Control . . . 124
5 Experiments . . . 125
5.1 Reward Functions . . . 125
5.2 Push Task . . . 126
5.3 Peg-in-Hole Task . . . 129
6 Conclusion . . . 130
References . . . 131

2 Generalizing Behavior Trees and Motion-Generator (BTMG) Policy Representation for Robotic Tasks over Scenario Parameters 139
1 Abstract . . . 139
2 Introduction . . . 139
3 Formalization . . . 141
4 Mapping . . . 141
5 Experiments . . . 142
6 Future Work . . . 142
References . . . 143

3 Learning to Adapt the Parameters of Behavior Trees and Motion Generators (BTMGs) to Task Variations 147
1 Abstract . . . 147
2 Introduction . . . 148
3 Related Work . . . 150
4 BTMG and Task Variations . . . 151
5 Approach . . . 152

5.1 Training Phase . . . 153
5.2 Query Phase . . . 154
6 Experiments . . . 155
6.1 Obstacle Avoidance Task . . . 157
6.2 Push task . . . 160
7 Conclusion and Future Work . . . 164
References . . . 165

4 Adaptable Recovery Behaviors in Robotics: A Behavior Trees and Motion Generators (BTMG) Approach for Failure Management 171
1 Abstract . . . 171
2 Introduction . . . 172
3 Related Work . . . 174
4 Background . . . 175
4.1 Behavior Trees . . . 175
4.2 Behavior Trees and Motion Generators (BTMG) . . . 176
4.3 Learning parameters of BTMG . . . 178
5 Approach . . . 178
5.1 Assumptions . . . 178
5.2 Recovery Behaviors . . . 179
5.3 Planner . . . 180
5.4 Scenarios . . . 181
6 Experimental Setup . . . 183
6.1 Peg-in-a-hole Task . . . 183
6.2 Results and Discussion . . . 184
7 Conclusion and Future Work . . . 187
References . . . 188

5 Addressing Failures in Robotics using Vision-Based Language Models (VLMs) and Behavior Trees (BT) 195
1 Abstract . . . 195
2 Introduction . . . 195
3 Background . . . 196
3.1 Behavior Trees (BT) . . . 197
3.2 Reactive Planner . . . 197
3.3 Vision Language Models (VLM) in Robotics . . . 197
4 Approach . . . 197
4.1 Failure Detection and Idenitification . . . 198
4.2 Monitoring using VLM . . . 199
4.3 Condition and Skill Template Generation . . . 199
5 Experiments . . . 201
6 Evaluation Metrics and Results . . . 201

7 Conclusion and Future Work 202
References 202

6 A Unified Framework for Real-Time Failure Handling in Robotics Using Vision-Language Models, Reactive Planner and Behavior Trees 215
1 Abstract 215
2 Introduction 216
3 Related Work 218
3.1 Traditional Failure Recovery Strategies 218
3.2 Learning-Based Failure Recovery 219
3.3 Failure Recovery with Large Language Models (LLMs) and Vision-Language Models (VLMs) 219
4 Background 220
4.1 Behavior Trees 220
4.2 Reactive Planner 220
4.3 Vision-Language Models 221
5 Approach 221
5.1 Pre-Execution Failure Verification 223
5.2 Real-Time Failure Monitoring 225
5.3 Skill Addition 226
5.4 Scene Graph Representation 228
5.5 Execution History 229
6 Experiments and Results 229
6.1 Simulation Experiments 230
6.2 Real-World Experiments 230
7 Conclusion and Future Work 233
8 Acknowledgements 233
References 234

List of publications

This thesis is based on the following publications:

I **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

II **Generalizing Behavior Trees and Motion-Generator (BTMG) Policy Representation for Robotic Tasks Over Scenario Parameters**
F. Ahmad, M. Mayr, E.A. Topp, J. Malec, V. Krueger
IEEE International Joint Conferences on Artificial Intelligence (IJCAI), Vienna, Austria, 2022, Workshop on Bridging the Gap Between AI Planning and Reinforcement Learning

III **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, MI, USA, 2023, pp. 10133-10140, doi: 10.1109/IROS55552.2023.10341636

IV **Adaptable Recovery Behaviors in Robotics: A Behavior Trees and Motion Generators (BTMG) Approach for Failure Management**
F. Ahmad, M. Mayr, S. Suresh-Fazeela, V. Krueger
IEEE 20th International Conference on Automation Science and Engineering (CASE), Bari, Italy, 2024, pp. 1815-1822, doi: 10.1109/CASE59546.2024.10711715

V **Addressing Failures in Robotics using Vision-Based Language Models (VLMs) and Behavior Trees (BT)**
F. Ahmad, J. Styrud, V. Krueger
To appear in European Robotics Forum 2025, vol. 36, Springer Proceedings in Advanced Robotics, Springer, 2025, ch. 43, ISBN 978-3-031-89470-1. arXiv:2411.01568, 2024.

VI **A Unified Framework for Real-Time Failure Handling in Robotics Using Vision-Language Models, Reactive Planner and Behavior Trees**
F. Ahmad*, **H. Ismail***, J. Styrud, V. Krueger
arXiv:2503.15202, 2025.

All papers are reproduced with permission of their respective publishers.

Publications not included in this thesis:

VII **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

VIII **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 Trend and Advances in Machine Learning and Automated Reasoning for Intelligent Robots and Systems

IX **Learning Generalized Robotic Skills**

F. Ahmad, V. Krueger, E.A. Topp, J. Malec
Swedish AI Society (SAIS) 34th annual Workshop, Stockholm, Sweden, 2022, track for ongoing Ph.D. projects

X **Hybrid planning for challenging construction problems: An Answer Set Programming approach**
F. Ahmad, V. Patoglu, E. Erdem
Artificial Intelligence, vol. 319, p. 103902, 2023. doi: 10.1016/j.artint.2023.103902

XI **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

XII **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

XIII **Hybrid planning for challenging construction problems: An Answer Set Programming approach (Extended Abstract)**
F. Ahmad, V. Patoglu, E. Erdem
Extended abstract presented at KR 2025, Recently Published Research (RPR) Track, Melbourne, Australia, 2025.

Acknowledgements

Finally, it is done. There are many people to thank, and if I have forgotten someone, please do not take it personally. Writing acknowledgements on the last day before the final print was probably not the best idea, so I apologize in advance.

I would like to thank the RSS group, especially my supervisor Volker Krueger for his support and feedback. I am also grateful to my co-supervisors Jacek Malec and Elin Anna Topp for their guidance. I also thank Maj Stenmark for her collaboration on a paper and for her valuable feedback. I am also grateful to Görel Hedin, Martin Höst, and Per Runeson for their valuable insights and encouragement during my PhD journey.

I want to thank my colleagues. I worked the most with Matthias Mayr, who taught me many things, especially about organizing day-to-day life such as schedules, calendars, and tasks, which I still do not quite get completely. Thank you for your patience and support. I also thank Alexander Dürr, who organized one of the best midsummer parties in a rented summer house, and who also hosted unforgettable BBQs. Thanks to Johan Oxenstierna, who sat next to me, for his unique perspective on things, his dark humor that took some time to get used to, and for the unforgettable moment at a party when others were listing the drinks they had brought and he simply said, with a straight face, "I brought a cup." And thanks to Hampus Åström for our conversations about the struggles of PhD life.

I would also like to thank Simon Kristoffersson Lind for motivating me to go to the gym and for sharing tomato plants, Ayesha Jena for always remembering campus rules and practical details better than me even though I had been here longer, Esranur Ertürk for making me laugh with your mispronunciations and for the chocoballs, Hashim Ismail for continuing conversations long after everyone else had moved on, Jonas Conneryd for being the math person I could rely on, Jialong Li for introducing me to Chinese traditions, and Marcus Klang for teaching me about networking and for helping me find bugs in the robot's programming code.

I am also grateful for all the fika and lunchroom conversations with my colleagues, both within and outside the RSS group, which made daily life so much better. In particular, thanks to Anton Risberg Alaküla for raising questions like "What is the actual difference between a burger and a sandwich?", which I am still not sure about, and also for organizing another great midsummer party. Thanks to Rikard Olajos and Gustaf Waldemarson for your witty replies. Thanks also to Robbert Hofman, who visited from Belgium for research, for the enjoyable moments we shared during his stay and the good times both inside and outside work. A big thank you to Michael Doggett for supporting me in the only LundaLoppet I ever ran, 10 km, and, funny enough, I stopped running LundaLoppet after that one. Thanks also to Gareth Callanan for the training runs for LundaLoppet and for the unforgettable "ice wine" story, which Alexander and I misheard in a very funny

way, even though it was not meant as a joke. Thanks to Flavius Gruian for your unusual choice of cups and sharp humor, and to Michail Boulasikis for finding humor in the most unexpected ways and for your hilarious commentary on box office bombs. And thanks to Peng Kuang for introducing me to Chinese snacks.

I also want to thank Noric Couderc for witty comments and for letting us stay at your mom's place during vacation. I still cannot spell the name of the hometown, but it was beautiful. Thanks to Idriss Riouak for fika conversations and for teaching me the ABCs of Italian gestures.

Thanks also to Alexandru Dura and Jesper Öqvist for thought-provoking discussions, including those about parenting, to Erik Hellsten, Carl Hvarfner, Leonard Papenmeier, and Luigi Nardi for valuable discussions on Gaussian processes and Bayesian optimization, to Christoph Reichenbach for taking the time to discuss safeguards for large language model outputs, and to Jonathan Styrud for our collaboration on papers and his valuable feedback.

To the administrative staff, thank you. A special thanks to Anders Bruce, I might hold the record for breaking the most laptops in five years, but you always had a solution. And to Heidi Adolfsson, thank you for always fixing my expense reports, I still do not know how to fill them.

Finally, I want to thank my family and friends for their support. A very special thanks to my parents, even though they are no longer in this world, I feel they are still with me. Lastly, to my wonderful wife Momina Rizwan and my daughter Rehana, who are the pillars of my life. And of course, thanks to Allah Almighty, who gave me the strength to stand again even in the darkest hours.

Best regards,
Faseeh Ahmad

Popular summary in English

Collaborative robots are becoming increasingly common in factories, warehouses, and even homes. While they perform well in controlled environments, they often struggle when something unexpected happens, such as an object being missing, misaligned, or blocked. These types of failures are especially common in dynamic environments like small-batch manufacturing or service robotics. In such situations, most robots simply stop and wait for human assistance, leading to delays and increased supervision. To be truly useful in everyday settings, robots must be able to recognize problems, understand what went wrong, and recover on their own, just as humans do.

This research focuses on developing self-reliant robotic systems capable of detecting, explaining, and recovering from both expected and unforeseen failures. A central part of the approach involves the use of modular and interpretable robot behaviors, known as robot skills. These skills are organized and executed using behavior trees (BTs), which allow robots to respond to changing conditions in a structured and reactive way. Each skill is further parameterized using motion generators (MGs), which define how a specific movement should be carried out. To fine-tune these parameters across varying task conditions, the framework uses reinforcement learning. Specifically, data-efficient black-box optimization methods, such as bayesian optimization (BO), are used to optimize skill parameters with respect to multiple objectives, such as safety, speed, and task success, using simulation and domain randomization. This setup enables robots to flexibly adapt their behavior without retraining complex models from scratch and provides interpretability and control that are often missing in deep learning-based approaches.

To handle unexpected situations during task execution, the framework integrates vision-language models (VLMs), which combine visual understanding with natural language reasoning. These models help the robot identify what went wrong and suggest possible corrections in human-readable terms. This combination of planning, learning, and semantic reasoning ensures that robot behavior remains both adaptable and transparent, in contrast to traditional black-box systems.

The proposed approaches have been tested in a variety of simulated environments, including RobotDART, MuJoCo, AI2-THOR, and Isaac Sim, as well as on physical robotic platforms such as the ABB YuMi, KUKA iiwa, and a mobile manipulator composed of a UR5e arm mounted on a MiR200 base. Tasks included peg insertion, object pushing, surface wiping, item sorting, and drawer placement, many of which required physical interaction under uncertainty.

Looking ahead, the framework is being extended in two directions: proactive failure prediction using runtime monitoring, and on-the-fly skill generation using emerging vision-language-action (VLA) models or diffusion policies. While these capabilities are still under

development, they represent promising steps toward further reducing human intervention and increasing robot autonomy in complex, real-world settings. This work contributes to the development of robotic systems that are not only more adaptable but also more autonomous and trustworthy.

Populärvetenskaplig sammanfattning på svenska

Samarbetsrobotar blir allt vanligare i fabriker, lager och till och med i hem. Även om de fungerar bra i kontrollerade miljöer har de ofta svårt att hantera oväntade situationer, till exempel om ett föremål saknas, är felplacerat eller blockerat. Sådana fel är särskilt vanliga i dynamiska miljöer som småskalig tillverkning eller servicerobotik. I dessa fall stannar roboten vanligtvis och väntar på mänsklig hjälp, vilket leder till förseningar och ett ökat behov av övervakning. För att vara verkligt användbara i vardagliga miljöer måste robotar kunna upptäcka problem, förstå vad som gick fel och återhämta sig själva, precis som människor gör.

Denna forskning fokuserar på att utveckla självständiga robotsystem som kan upptäcka, förklara och återhämta sig från både förväntade och oväntade fel. En central del av tillvägagångssättet är användningen av modulära och begripliga robotbeteenden, så kallade robotfärdigheter. Dessa organiseras och körs med hjälp av beteendeträd (behavior trees, BT), vilket gör att roboten kan reagera strukturerat och flexibelt på förändrade förhållanden. Varje färdighet finjusteras med hjälp av rörelsegeneratorer (motion generators, MG), som definierar hur en specifik rörelse ska genomföras. För att anpassa parametrarna till olika uppgifter används förstärkningsinlärning. Mer specifikt används dataeffektiva optimeringsmetoder, såsom bayesiansk optimering, för att hitta parametrar som balanserar mål som säkerhet, hastighet och uppgiftsframgång. Detta sker i simulerade miljöer med variation och gör att roboten kan anpassa sitt beteende utan att behöva träna om komplexa modeller. Samtidigt bibehålls transparens och kontroll, vilket ofta saknas i djupinlärningsbaserade metoder.

För att hantera oväntade situationer under uppgifternas gång integreras vision-språkmodeller (vision-language models, VLMs), som kombinerar visuell förståelse med språkligt resonemang. Dessa modeller hjälper roboten att identifiera vad som gått fel och föreslå möjliga korrigeringar i ett format som människor kan förstå. Kombinationen av planering, inlärning och semantisk förståelse ger ett beteende som är både anpassningsbart och transparent, i kontrast till traditionella svarta lådan-system.

Metoderna har testats i flera simulerade miljöer, inklusive RobotDART, MuJoCo, AI2-THOR och Isaac Sim, samt på fysiska robotplattformar som ABB YuMi, KUKA iiwa och en mobil manipulator med en UR5e-arm monterad på en MiR200-bas. Uppgifterna omfattade bland annat instick av pluggar, föremålspushning, avtorkning av ytor, sortering av objekt och placering av lådor, många av dem med krav på fysisk interaktion i osäkra miljöer.

Framåt utökas ramverket i två riktningar: proaktiv felsökning genom övervakning under körning samt förmågan att generera nya färdigheter i farten med hjälp av framväxande vision-språk-handlingsmodeller (VLA) eller diffusionspolicys. Även om dessa funktioner fortfarande är under utveckling utgör de lovande steg mot att ytterligare minska behovet

av mänsklig inblandning och öka robotars autonomi i komplexa miljöer. Denna forskning bidrar till utvecklingen av robotsystem som inte bara är mer anpassningsbara utan också mer självständiga och pålitliga.

Towards Self-Reliant Robots: Skill Learning, Failure Recovery, and Real-Time Adaptation

1 Introduction

Robots have revolutionized industrial processes by performing repetitive, high-precision tasks in structured environments such as manufacturing lines and warehouses [1]. These systems are typically programmed to execute fixed routines and operate under strict environmental assumptions, which has led to gains in productivity, safety, and cost-efficiency. However, the scope of robotics is now extending to unstructured and dynamic environments, such as homes, kitchens, and small-batch industries, where robots must interact with unpredictable objects, people, and evolving task requirements [2].

To operate in such settings, robots must demonstrate a higher degree of autonomy. *Autonomous robots* perceive their environment, make decisions, and act without constant human oversight [3, 4, 5]. These capabilities are vital for applications like home assistance and customized manufacturing, where fixed policies cannot address the diversity of real-world tasks [6].

Despite being autonomous, robots can still encounter unexpected situations such as misplaced objects, occlusions, or partial task failures, which may prevent successful task completion. These challenges highlight the need for *self-reliant robots*: robots that go beyond autonomy by identifying failures, understanding what went wrong, and recovering without human intervention [7]. In collaborative environments like homes or shared workplaces, such capabilities ensure reliability and continuity [8].

The current state of robotic systems reveals important limitations that hinder this goal:

- they rely on predefined routines that often break under unexpected situations,

- they require human intervention for diagnosing and correcting errors,
- and they provide limited transparency into their internal decision-making processes [9].

While transparency is not essential for self-reliance, it remains a valuable attribute, as it supports debugging, builds user trust, and helps human operators understand robotic behavior. Thus, systems that explain their decisions are more usable and trustworthy in real-world settings.

Looking beyond basic autonomy, enabling self-reliance in robots calls for the ability to monitor execution, detect failures, and modify plans accordingly, without relying on human intervention. Achieving this requires integrating symbolic reasoning, adaptability to task variation, and mechanisms for real-time recovery. Prior research shows that combining structured control architectures with data-driven components improves robustness and adaptability in unstructured or dynamic environments [10, 11].

Symbolic control frameworks such as behavior trees (BTs) offer modular and interpretable policy structures that support robust task execution and easier debugging [12]. BTs define how a robot selects and sequences symbolic skills to achieve a task. These skills can be further grounded in parameterized motion policies, forming behavior trees and motion generators (BTMGs) [13]. In this framework, the symbolic structure of the BT encodes the logical flow of skills, while each skill invokes a motion generator (MG), a compliant controller that operates at the trajectory level and enables interaction with the physical world through impedance control. This structured layering supports both clarity in control flow and flexibility in low-level motion.

At the same time, recent advances in multimodal and foundation models [14, 11, 15], such as vision-language models (VLMs), vision-language-action models (VLAs) [16, 17] and diffusion policies [18], offer complementary capabilities. These models provide high-capacity perception and reasoning grounded in visual and textual modalities. While they often trade off strict interpretability, they enable flexible semantic inference and contextual generalization. Such models can assist in detecting anomalies, suggesting corrections, and predicting failures, capabilities crucial for self-reliant operation in open-ended settings [8, 3].

This thesis brings these elements together into a unified framework. By combining symbolic execution via BTs and BTMGs, low-level control through MGs, and high-level reasoning using multimodal models, we aim to support adaptation to task variation, recovery from failures, and reduced reliance on human intervention. The overall aim is:

To enable robotic systems to adapt to task variations, recover from execution failures, and maintain modularity, interpretability or explainability, and minimal human dependence.

1.1 Overview of Aims and Research Questions

Achieving the overall goal of enabling robots to adapt to task variations, recover from failures, and operate with minimal human input involves two key challenges: modeling adaptable and interpretable robot behaviors, and enabling autonomous failure detection, explanation, and recovery. To address these, the thesis is divided into two aims, each accompanied by specific research questions and corresponding methods.

Aim 1: *To enable robots to solve complex, varied tasks with modular, interpretable, and adaptable policies.*

This aim targets the core challenge of building robot controllers that are both understandable and adaptable to different task scenarios. Traditional policies lack modularity and make it difficult to generalize across variations. To address this, we adopt BTs for structuring policies and integrate them with parameterized MGs, resulting in BTMG policies. These policies support clear task decomposition while allowing low-level control to be tuned for different situations [6].

RQ1.1: How can robot behavior policies be structured to remain modular, interpretable, and tunable for varying tasks?

This question examines how to design robot policies that balance symbolic structure with parametric flexibility, enabling both clarity and adaptability. To address this, Paper I [6] employs the BTMG policy framework in combination with bayesian optimization (BO) in a reinforcement learning (RL) setting. This allows low-level motion parameters to be optimized while preserving the modular and human-readable structure of BTs, making the resulting policies easier to understand, maintain, and generalize across tasks.

RQ1.2: How can such policies be adapted to new task variations without retraining?

A key requirement for real-world deployment is the ability to adapt policies to new task configurations without retraining. While BTMGs offer modularity and interpretability, their parameters typically need to be tuned for each specific task instance, which limits generalization. This research question addresses the challenge of extending BTMGs to generalize across task variations.

In Paper II [19], we address this by learning a mapping from scenario parameters (e.g., goal pose) to policy parameters (e.g., offsets defining motion trajectories) using gaussian processes. This allows interpolation of suitable BTMG parameters for unseen task instances, enabling policy reuse without retraining.

RQ1.3: How can policy parameters be predicted efficiently for real-time adaptation to un-

seen tasks?

Real-world robotics requires not only adaptability but also responsiveness. While learned task-parameter mappings (RQ1.2) help generalize to new scenarios, they may still involve non-negligible inference time. This question focuses on enabling rapid prediction of policy parameters suitable for unseen task variations.

To address this, we introduce PerF, a lightweight predictive model that maps task variations directly to BTMG parameters. Combined with a local optimizer, it generates feasible and near-optimal policy parameters within seconds. This approach supports real-time deployment of skill policies without retraining or extensive computation. These contributions are presented in Paper III [7].

Aim 2: *To enable autonomous failure detection, explanation, and recovery in robots without human intervention.*

While adaptability is critical for task performance, real-world deployment also demands robustness to failures that arise during execution due to dynamic environments, mechanical noise, or perception errors. To meet this requirement, robots must not only detect and identify such failures but also generate appropriate recovery actions without relying on human assistance. This aim addresses the challenge of building such self-reliant systems and is guided by three research questions.

RQ2.1: How can recovery behaviors be designed and structured so they are modular and reusable?

This question focuses on the structure and generality of failure handling strategies, a critical component for autonomous recovery. Instead of designing ad-hoc responses for each failure type, we represent recovery strategies using the same BTMG abstraction applied to skill policies. Paper IV [9] demonstrates how this shared representation allows recovery behaviors to be modeled as parameterized, modular components that integrate naturally into the overall task structure. These behaviors are both interpretable and transferable across failure contexts, supporting the goal of robust and reusable recovery.

RQ2.2: How can robots detect and explain failures before executing a skill, enabling preemptive intervention?

Preemptively identifying likely failures is crucial for preventing unnecessary execution attempts and improving robustness in dynamic environments. This research question focuses on how robots can use contextual information to assess the feasibility of upcoming skills and adjust plans before execution.

In Paper V [8], we address this by integrating VLMs with a reactive planner and beha-

vior trees to perform pre-execution reasoning. The VLM queries the scene, skill plan, and preconditions to predict whether a skill is likely to fail and identifies missing or violated preconditions. This enables the robot to take preventive actions, such as modifying the behavior tree or inserting recovery steps before failure occurs.

RQ2.3: How can robots generate corrective behaviors at runtime and recover without restarting the task?

Execution-time failures require prompt, context-aware interventions to ensure task continuity. This research question focuses on how robots can detect such failures during skill execution and insert corrective actions into their current plan without resetting or restarting the entire task.

Paper VI[3] extends the framework of Paper V by incorporating real-time monitoring into the VLM–reactive planner–BT loop. During execution, the VLM continuously verifies preconditions and postconditions against the evolving scene, while also identifying missing preconditions or proposing alternative skills as in Paper V. When a failure is detected, the reactive planner augments the running BT with corrective behaviors, enabling in-place recovery and uninterrupted task progress.

Before outlining the thesis structure, we clarify the abstraction hierarchy used throughout this work. Since the research questions span low-level motion adaptation, skill reuse, and high-level failure reasoning, this hierarchy helps situate where each contribution fits within the robot control stack.

Abstraction hierarchy: This thesis considers robot behavior across three levels of abstraction:

- **Trajectory level:** Low-level motion primitives such as MGs [13] or dynamic movement primitives (DMPs) [20] generate physical motion in the robot's workspace.
- **Skill level:** These trajectories are wrapped into symbolic skills with semantic annotations such as preconditions and postconditions, enabling modular reuse and parameter adaptation.
- **Task level:** Skills are composed into high-level plans that pursue user-defined goals, typically using symbolic planners and BTs for structure and reactivity.

This hierarchy enables the combination of adaptable control with symbolic structure. It is used throughout the thesis to frame how different methods contribute to building robust and flexible robot behavior.

1.2 Thesis Outline

The remainder of this thesis is structured as follows:

- **Chapter 2** introduces key tools for behavior modeling, including BTs, MGs, reactive planning, and symbolic control.
- **Chapter 3** reviews learning-based approaches for skill adaptation and policy optimization.
- **Chapter 4** presents foundational concepts in failure detection, recovery strategies, and runtime adaptation.
- **Chapter 5** covers VLMs and their role in perception-grounded robotic reasoning.
- **Chapters 6–8** correspond to Part I of the thesis, addressing **Aim I**, which focuses on modular and adaptable robot policies. These chapters discuss the structuring of policies (RQ1.1), adaptation to task variations (RQ1.2), and real-time parameter prediction (RQ1.3), respectively.
- **Chapters 9–11** comprise Part II of the thesis, addressing **Aim II**, which targets autonomous failure detection and recovery. These chapters address the design of reusable recovery behaviors (RQ2.1), pre-execution failure detection using VLMs (RQ2.2), and runtime failure handling and correction (RQ2.3).
- **Chapter 12** concludes the thesis by summarizing the contributions, discussing broader implications and limitations, and outlining directions for future work toward self-reliant robotic systems.

Background and Related Work

Chapters 2 through 5 provide the background tools, and conceptual foundations required to understand and contextualize the contributions of this thesis. Each chapter introduces essential methods, frameworks, or models and reviews the relevant literature related to the specific capabilities needed for skill adaptation and failure recovery in robotics.

Chapter 2 introduces the key components of robot behavior modeling, including BTs, reactive planning, MGs, and their integration via the BTMG framework. It also presents symbolic planning with SkiROS2 (a skill-based platform).

Chapter 3 reviews learning-based approaches for adapting and optimizing robot skills. It covers reinforcement learning, policy search, multi-objective optimization, bayesian optimization, and gaussian processes, with a focus on generalizing behavior across task variations.

Chapter 4 lays the theoretical foundation for failure recovery. It discusses different models of execution failure, compares reactive and proactive recovery strategies, surveys methods for generating corrective behaviors, and introduces runtime monitoring and adaptation techniques.

Chapter 5 provides an overview of VLMs and their role in robotic reasoning. It discusses technical foundations and practical applications such as scene understanding, instruction following, and failure explanation.

One important point about how related work and background tools are presented in these chapters is that they are discussed in context rather than repeated across sections. When a method or paper contributes to multiple aspects of the thesis, it is cited at the most relevant location. For example, the same work may be mentioned in one section for its contribution to failure detection, and in another for its role in runtime adaptation. This helps avoid repetition and makes it easier to follow how each piece of prior work supports specific parts of the thesis.

2 Skill Representation and Execution Frameworks

This chapter introduces the core frameworks used for representing and executing robotic skills in a modular and reactive manner. It covers BTs as the primary structure for organizing robot actions, reactive planning for dynamically constructing and adapting BTs, and MGs for low-level compliant control. The chapter also describes how these components are integrated through the BTMG framework and how symbolic planning is used to ground skill semantics and guide execution.

2.1 Behavior Trees

BTs are hierarchical control architectures that originated in the video game industry to model complex agent behaviors in a modular and reactive manner [21, 22]. In robotics, their adoption has grown due to their structural clarity and support for real-time decision making [10]. BTs enable complex behaviors to be built from smaller, reusable components, supporting modular design and dynamic execution.

A BT is a directed rooted tree used to structure decision making and control flow in robotics. It consists of two main types of nodes: control flow nodes and execution nodes. Control flow nodes define the execution structure. The `sequence` node executes its children from left to right and returns `success` only if all children succeed. The `fallback` (or `selector`) node also visits children in order but returns `success` as soon as any child does. The `parallel` node runs all children simultaneously and uses configurable thresholds to determine overall success or failure. The `decorator` node modifies the behavior of a single child, such as inverting its result or retrying execution.

Execution nodes represent the leaf-level actions and conditions. `Action` nodes correspond to atomic robot operations like picking or moving and return one of three statuses: `success`, `failure`, or `running`. `Condition` nodes evaluate boolean predicates such as whether an object is at a certain position and return `success` or `failure` accordingly.

Execution begins with a periodic 'tick' sent from the root. The tick traverses the tree top-down, conventionally left to right, triggering node evaluation based on the defined control flow [23]. Each node returns a status, `success`, `failure`, or `running`, which guides the flow of control and enables responsive behavior switching [12]. Since skills often include semantic preconditions (e.g., an object must be in a specific location before it can be picked), BTs can include condition nodes to check these preconditions during execution. This structure allows developers to encode retry mechanisms, precondition checks, and timeouts in a modular way without changing the overall behavior logic.

Figure 1 illustrates a peg-in-hole task organized via BT. Sequential nodes enforce task or-

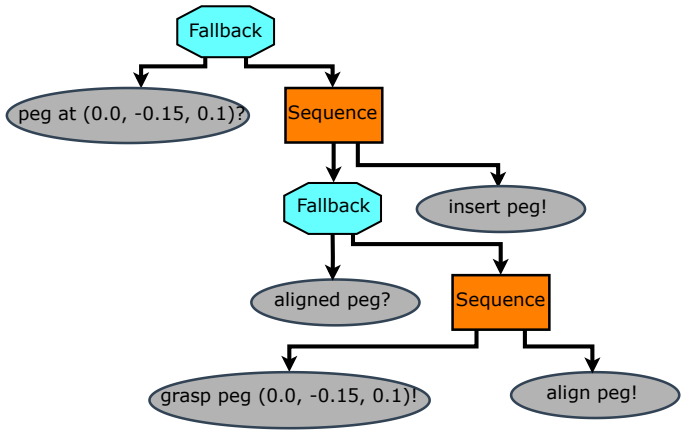


Figure 1: Behavior tree for a peg-in-hole task. The tree includes condition nodes (e.g., checking alignment), action nodes (e.g., grasp, insert), and control flow nodes (`Sequence`, `Fallback`) to manage flow and recovery.

der, while fallback branches enable recovery from failed conditions. This modular and interpretable structure supports real-time behavior control in uncertain environments.

Alternative Control Architectures

Before focusing on behavior trees, it is important to briefly consider alternative control architectures that have been used in robotics. These include finite state machines (FSMs), hierarchical FSMs (HFSMs), petri nets, and hierarchical task networks (HTNs), each with specific strengths and limitations. This provides context for why BTs are chosen in this thesis and highlights their advantages over other strategies.

FSMs define robot behavior as a set of discrete states connected by transitions. They are conceptually simple and easy to implement, but they suffer from state explosion as tasks grow in complexity, making them hard to scale in practical applications [24]. **HFSMs** attempt to mitigate this problem by introducing nested states, which improve modularity. However, this added hierarchy makes transitions more complex to manage and reduces runtime flexibility, especially in dynamic environments [25].

Petri nets offer another approach by modeling concurrency and synchronization explicitly [26]. They are powerful for tasks that require parallel actions and formal analysis but are often too low-level to represent high-level robotic tasks effectively. In contrast, **HTNs** [27] excel in symbolic task decomposition, providing structured plans for long-horizon goals. However, HTNs generally assume static and fully observable environments,

and their plans lack built-in reactivity, making them less suitable for highly dynamic or uncertain conditions.

Hybrid architectures attempt to combine the benefits of these methods. For instance, **HTN-BT** combinations [28] integrate the deliberative planning of HTNs with the reactive capabilities of BTs. While effective in some contexts, these hybrid approaches increase system complexity and require careful coordination between layers.

In comparison, behavior trees offer a balanced solution by combining modularity, scalability, and real-time adaptability [29, 30, 31]. Their hierarchical structure allows complex behaviors to be composed from reusable components, while their tick-based execution supports dynamic reaction to changing conditions without discarding previous progress. This makes BTs particularly suitable for robotic applications where both interpretability and runtime flexibility are essential.

Design and Synthesis of Behavior Trees

BTs are often designed manually using frameworks such as `BehaviorTree.CPP` or `py_trees` [32]. While manual construction offers full control and interpretability, it becomes time consuming and error prone in complex or highly variable tasks. To overcome these limitations, several automated synthesis methods have been proposed, constructing BTs from data, symbolic models, or interaction with the environment.

One approach is **genetic programming**, which evolves BT structures through mutation and crossover operations guided by a task specific fitness function [33]. Genetic programming can generate novel behaviors without requiring manual design but may produce large or inefficient trees if not properly constrained.

Another approach is **learning from demonstration (LfD)**, where expert trajectories are segmented and organized into interpretable BT structures [34, 35, 36]. This method allows robots to reproduce demonstrated behaviors in a modular form, but it generally requires multiple demonstrations to achieve robustness and generalization.

A third approach is **planning-based compilation**, which converts symbolic task plans into BTs [37]. In this method, the planner generates an action sequence that is compiled into a BT, often including fallback branches and retry loops for added reactivity. While principled and structured, these BTs remain static during execution unless explicitly recompiled, which can limit responsiveness in dynamic environments.

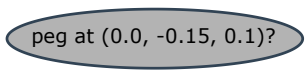
Finally, a fourth class of methods, **reactive planning** [10], constructs and updates BTs by combining planning with execution. Reactive planning can generate initial plans offline and dynamically insert new subtrees during execution when conditions fail or unexpected

situations arise, allowing robots to adapt without discarding prior progress. This approach is central to this thesis and is discussed in the following subsection.

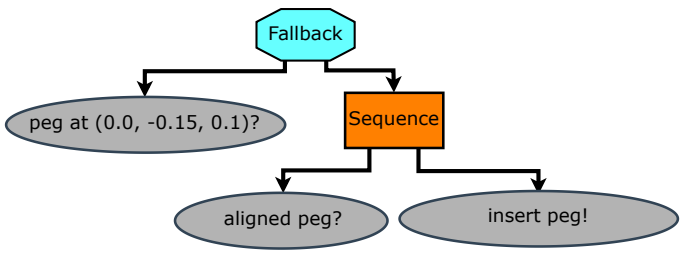
2.2 Reactive Planning for BT Generation and Extension

Reactive planning incrementally builds or updates robot behavior policies during task execution by alternating between acting and planning [10, 38]. Here, *behavior policies* refer to task execution strategies represented as BTs, where each policy specifies a structured sequence of condition checks and parameterized actions to achieve the goal. Unlike classical planning, which generates a complete action sequence before execution begins, reactive planning dynamically extends or adjusts the current policy based on the observed state. This enables the system to respond to environmental changes without restarting the entire planning process.

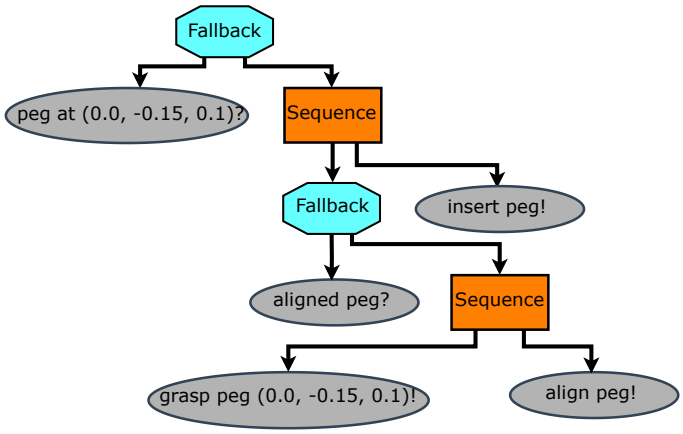
BTs are well-suited for this paradigm due to their modular and hierarchical structure. Execution may begin from an abstract or minimal BT. When a precondition fails (e.g., a required object is not yet grasped), the planner searches for an action whose postcondition satisfies the unmet precondition. This process is recursive: for each candidate action, the planner checks whether it is immediately executable or whether additional *preconditions* must be first met. The process continues, using symbolic backchaining, until the preconditions are satisfied or resolved into concrete actions (leaf nodes). The resulting sequence of supporting actions is composed into a subtree and inserted into the BT at runtime [39]. Figure 2 illustrates this mechanism with a peg-in-hole task, where the tree is incrementally constructed through subgoal expansion.



(a) Initial BT: check peg position.



(b) Subgoal added for peg alignment.



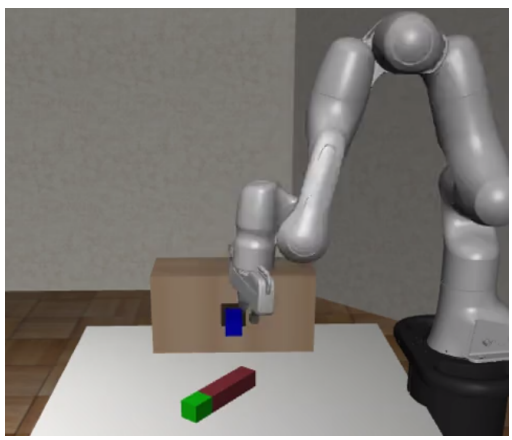
(c) Final BT: complete task structure.

Figure 2: Behavior tree constructed incrementally via reactive planning.

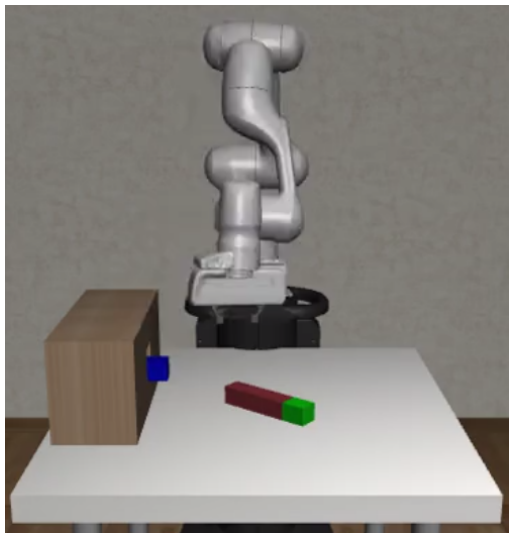
Some approaches extend reactive planning with parameter optimization. For example, BeBOP [38] combines BT-based planning with bayesian optimization to adjust action parameters dynamically, improving task robustness under uncertainty.

BT Generation

Figure 2 shows how a BT is constructed incrementally. Starting from a high-level goal (a), the planner introduces subgoals such as alignment (b), eventually producing a complete behavior structure (c).



(a) Side view of the peg-in-hole task.

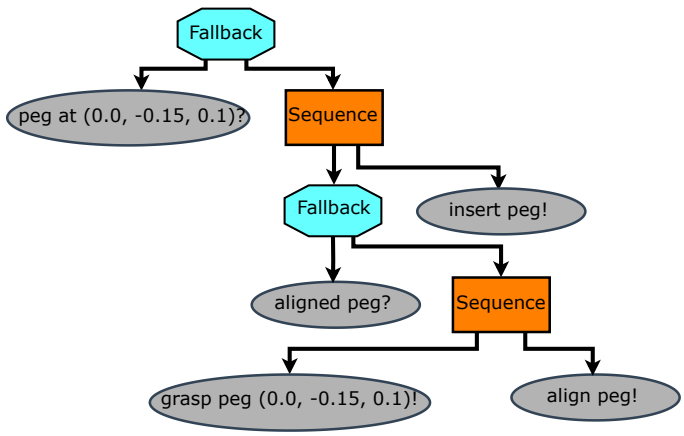


(b) Front view of the peg-in-hole task.

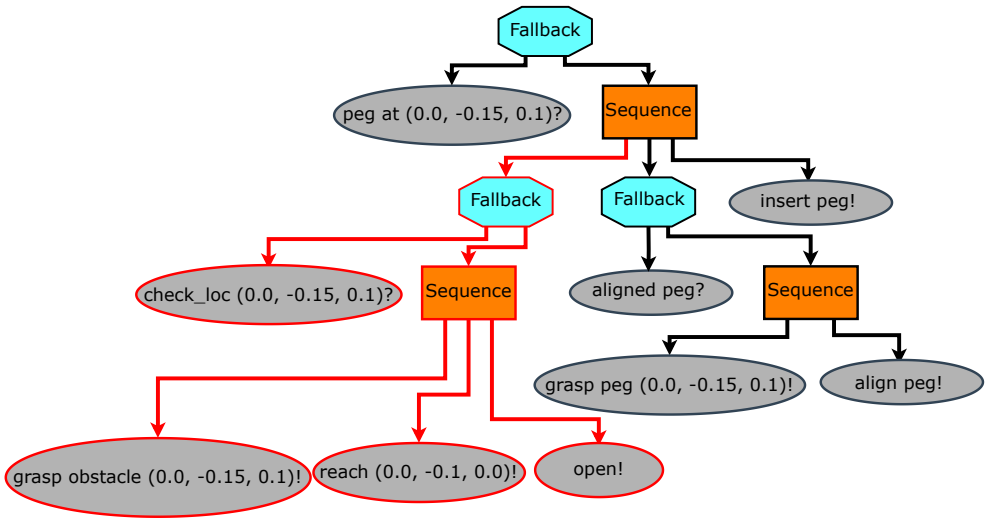
Figure 3: Obstructed scene requiring BT extension.

BT Extension

Reactive planning not only constructs BTs from scratch but also extends them during execution. Figure 3 shows the *peg-in-hole* task scene where the hole is obstructed. The planner inserts a subtree to handle obstacle removal, as shown in Figure 4, allowing the task to proceed without discarding prior progress.



(a) BT before extension.



(b) BT after extension.

Figure 4: BT extension via reactive planning.

Comparison with Classical Planning and Re-Planning

Classical symbolic planners construct complete action sequences offline using models such as STRIPS or HTNs [40]. Once execution begins, deviations from the expected state often require halting and re-planning the full sequence [41]. While re-planning improves robustness compared to static plans, it still incurs delays and often discards partial execution history.

In contrast, reactive planning operates online, incrementally repairing or extending the current plan while preserving previous structure. It integrates planning into execution, using symbolic reasoning to patch the behavior tree in response to failures or unexpected events. This fine-grained adaptation makes reactive planning especially suitable for real world robotics, where uncertainty and variability are unavoidable [10, 38].

2.3 Motion Generators

MGs were introduced by Rovida et al.[13] as a class of impedance controllers that (i) allow the superposition of generic cartesian wrenches and (ii) constrain velocities, accelerations, and torques to ensure safety. Under these conditions, MGs produce low-level, compliant end-effector motions using second-order dynamical systems that emulate virtual springs and dampers in cartesian space [13](See Figure 5). They are particularly effective for contact-

rich tasks, such as assembly, where safe interaction and adaptability are required.

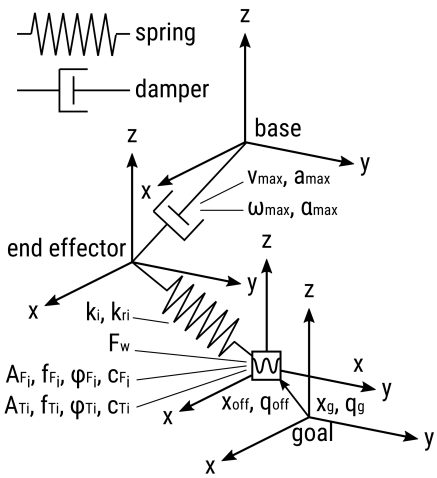


Figure 5: Representation of the motion generator parameters used in this thesis. Reprinted from [13], © 2018 IEEE, with permission.

The equations below follow the original formulation introduced by Rovida et al. [13], but are restated with adapted notation and expanded explanations for clarity. The end-effector motion is governed by:

$$m\dot{v} + bv = \bar{F}_c + F_d + F_w,$$

where m is the apparent end-effector mass, v and $\dot{v}$ are velocity and acceleration, b is the damping coefficient, F_d represents external disturbances (e.g., contact forces), and F_w is a feedforward term for compensation (e.g., gravity). The control force $\bar{F}_c$ is a saturated version of the virtual force:

$$\bar{F}_c = \begin{cases} F_c, & \|F_c\| \leq F_{\max}, \\ F_{\max} \dfrac{F_c}{\|F_c\|}, & \text{otherwise.} \end{cases}$$

Here, $F_{\max}$ is the maximum admissible force, and F_c is the composite virtual control force generated by springs, dampers, and optional excitation:

$$F_c = -K\big(x - (x_g + x_{\text{off}})\big) + F_e,$$

where $K = \text{diag}(k_1, k_2, k_3)$ is the stiffness matrix, x the current position, x_g the goal position, x_{off} an optional offset for strategy adjustments, and F_e the excitation force that can induce oscillatory or superimposed motions. For example, in a peg-in-hole task, F_e can generate a downward insertion motion combined with an Archimedean spiral search trajectory, configured through parameters such as path velocity, spiral radius, and step distance, to efficiently locate the hole under uncertainty.

By tuning stiffness, damping, and excitation parameters, MGs can realize behaviors such as guarded insertion, compliant alignment, and oscillatory probing. Multiple primitives can also be superimposed on orthogonal parameter subspaces to create hybrid behaviors (e.g., vertical insertion with lateral oscillation), improving adaptability without task-specific scripts. Unlike DMPs [42, 20], which encode motion implicitly from demonstrations, MGs expose explicit parametric control, improving interpretability, safety, and suitability for industrial applications.

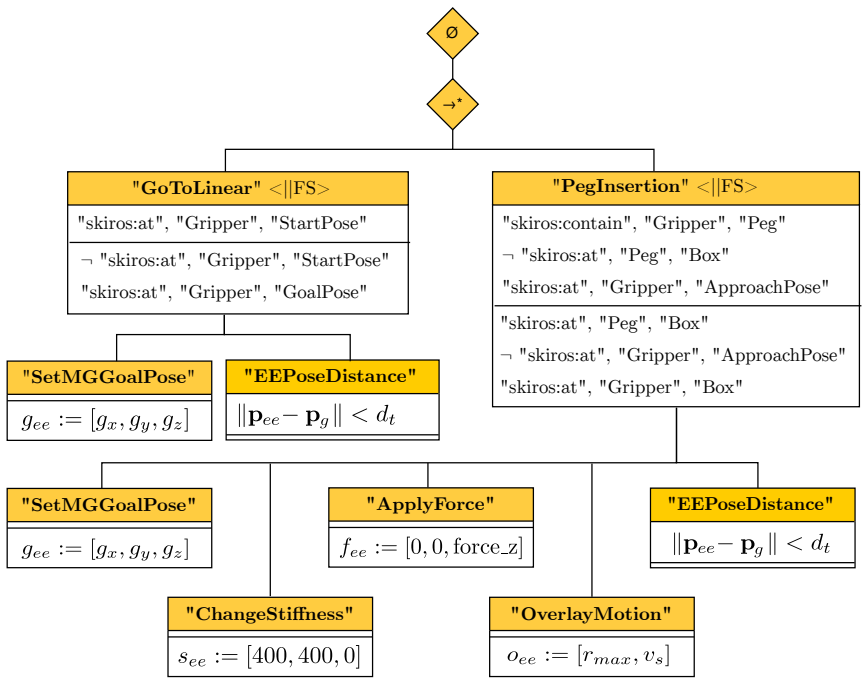


Figure 6: BTMG representation of a peg-in-hole task. Each BT node encodes a symbolic skill and invokes a motion generator with task-adapted parameters.

MGs operate at the *trajectory level* of abstraction, producing time-indexed cartesian motions without direct symbolic semantics. They are typically integrated into higher-level structures, such as BTs, which select and configure MG parameters at runtime. This combination allows symbolic reasoning at the skill and task levels, while MGs handle compliant

low-level execution.

Figure 6 illustrates MG configurations as part of a BTMG representation for a peg-in-hole task, where an Archimedean spiral trajectory combined with a downward force is configured through parameters such as spiral radius, path velocity, and step distance.

2.4 BTMG Framework: Integrating Behavior Trees with Motion Generators

The BTMG framework integrates symbolic control and reactive execution by combining BTs with parameterized MGs [13]. In this architecture, each BT leaf node invokes a context aware MG, enabling high-level symbolic decisions to trigger adaptable low-level motions.

Within the abstraction hierarchy, BTMGs operate at the *skill level* [28], elevating *trajectory level* control to reusable symbolic behaviors. Each BT leaf encapsulates a parameterized MG, forming a skill with semantic meaning and associated pre/postconditions. These skills can then be composed to form *task level* plans. This design enables symbolic planning over skills, while retaining motion adaptability through MG parameters.

Extended BTs (eBTs) [28] form the structural basis of BTMGs. They embed symbolic annotations such as preconditions, postconditions, and execution memory into BT nodes. This enables observation, runtime monitoring, and dynamic insertion of subtrees during execution, improving modularity and reactivity.

BTMGs use a parameterization scheme that separates the symbolic structure from the motion-level details, enabling reuse across tasks while allowing fine-grained adaptation. These parameters are broadly classified into two types. The **intrinsic parameters** (θ_i) define the BT structure and the ordered sequence of symbolic skills. These remain fixed across different task instances and capture the high-level behavior logic. The **extrinsic parameters** (θ_e) specify the numerical configuration of the MGs, such as stiffness values, positional offsets, or force directions. These are adapted to the current task context, allowing the same symbolic skills to perform appropriately under different variations.

Figure 6 illustrates a BTMG for a peg-in-hole task, where BT nodes represent symbolic skills such as grasping or insertion, each annotated with preconditions and postconditions, and linked to MGs configured with task-specific parameters.

This separation allows the symbolic control logic to remain constant while motion-level behaviors are tuned for new scenarios. Extrinsic parameters can be *manually tuned* using expert knowledge, *inferred through reasoning* over symbolic world knowledge [28], or *learned from data* using methods such as gaussian process regression or reinforcement learning [7].

In this thesis, the data-driven approach is of particular importance: extrinsic parameters are learned using BO in RL setting to adapt motion generators efficiently across different task

variations, enabling real-time reuse and fine-tuned performance. By preserving the intrinsic BT structure while adapting extrinsic MG parameters, BTMGs support generalizable and flexible robot behaviors in diverse and dynamic environments.

2.5 Symbolic Planning and Skill-Based Task Execution

This section addresses how symbolic reasoning is used to define, organize, and execute robot skills at the *task level*. These abstractions sit above the *trajectory* and *skill layers* discussed in Sections 2.3 and 2.4, enabling robots to reason about what to do and in what order, rather than how to perform each motion. Skills are defined as symbolic actions with preconditions and effects [43], which are then composed into high-level plans using symbolic planners. The resulting task-level plans are executed using frameworks that support integration with motion-level primitives.

Symbolic Planning

Symbolic planning enables robots to reason over abstract task representations to generate action sequences that achieve a defined goal. Planning problems are typically expressed in declarative languages such as PDDL (Planning Domain Definition Language) [44], where the world state, available actions, and goal conditions are specified symbolically.

Heuristic search algorithms, such as those used in Fast Downward [45], are then applied to find valid plans. This approach is well suited for long-horizon, structured tasks such as assembly, navigation, or multi-step manipulation, where the solution space can be explored efficiently through symbolic reasoning.

However, classical symbolic planners assume a fully known and static world model, which limits their robustness in dynamic or uncertain environments. For instance, when object positions change or unexpected obstacles occur, symbolic planners must either replan entirely or be combined with reactive architectures. To address this, symbolic planning is often integrated with BTs, which handle real-time feedback and partial plan repair while retaining symbolic grounding [34, 46].

Ontologies, Skill Models, and SkiROS2

Symbolic planning depends on well-structured representations of robot capabilities, environment semantics, and action outcomes. Ontologies provide this structure by defining a formal vocabulary of object types, relations, skills, and constraints [47, 48]. They support

reasoning over compatibility, type checking, and object affordances, enabling modular, robot-agnostic behavior design.

Robot skills are modeled as semantic abstractions with defined parameters, preconditions, postconditions, and an execution body [49]. These models allow planners to reason at a high level, deferring motion specifics to lower control layers.

One such skill-based framework is **SkiROS2** [43](Figure 7), which is also utilized in this thesis. It integrates symbolic planning, semantic reasoning, and skill execution. Skills are described using RDF-based representations (extending IEEE Std 1872™-2015 CORA) and include *preconditions*, which must hold before execution; *hold conditions*, which remain valid during execution; and *postconditions*, which describe the effects after execution. A semantic *world model* (WM) based on OWL-DL ontologies tracks object states and task context in real time.

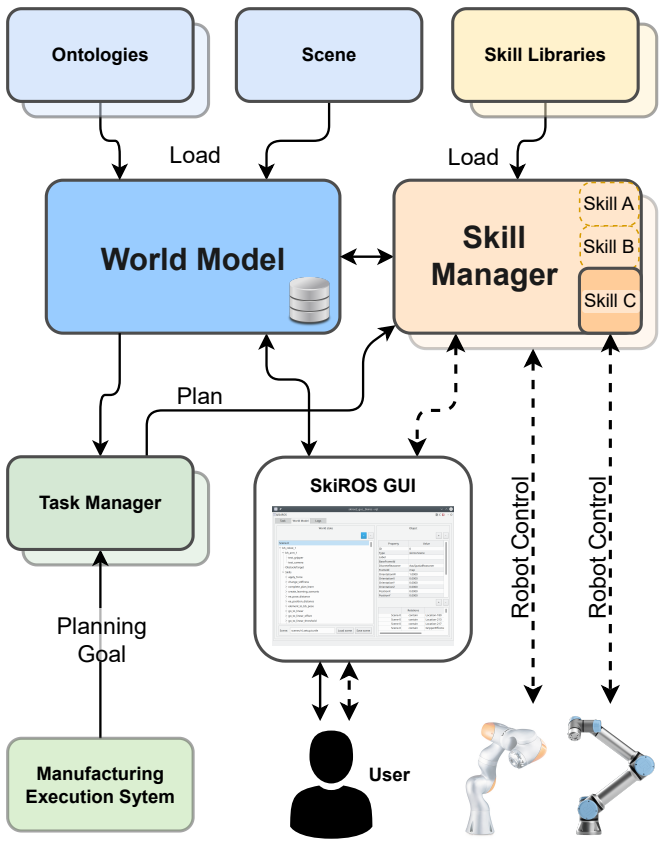


Figure 7: Schematic overview of the SkiROS2 control architecture with symbolic skills, world model, and task planner. Reprinted from [43], © 2023 IEEE, with permission.

SkiROS2 uses a *Task Manager* to formulate planning problems in PDDL, solve them with symbolic planners such as Fast Downward, and compile the resulting plans into eBTs for execution. A *Skill Manager* supervises skill execution and provides interfaces for monitoring. The framework also supports additional reasoning modules, such as geometric or temporal reasoners, which can infer missing parameters like tool poses, reachable surfaces, or compatible grippers based on the current world state.

Together, ontologies, semantic skill models, and frameworks such as SkiROS2 provide the foundation for structured, generalizable, and reactive robot control that remains robust under uncertainty.

3 Learning and Adaptation for Robotics

Robotic systems must operate across a wide range of environments and task configurations, which introduces variability in object properties, spatial arrangements, and required motions. While symbolic frameworks like BTMG support structured and reactive execution, they typically rely on fixed parameters and predefined models. To achieve generalization and robustness, learning-based methods are used to optimize skill parameters, adapt to task variations, and balance competing objectives such as performance and safety. This section reviews key approaches for learning and adaptation in robotics, including reinforcement learning, policy search, bayesian optimization, and gaussian process-based generalization.

3.1 Learning Policies in Robotics

This subsection contributes to RQ1 and RQ2 by reviewing how robotic policies can be learned and adapted to new task conditions. A central approach for learning such adaptive behaviors is *reinforcement learning* (RL), where robots improve their actions through trial and error interaction with the environment to maximize long-term performance (Figure 8). RL provides a formal framework for mapping sensory observations to actions under uncertainty, making it a natural choice for data-driven policy learning in robotics [50]. Formally, RL is modeled as a markov decision process (MDP), defined by the tuple (S, A, P, R, γ), where S is the set of states, A the set of actions, $P(s'|s, a)$ the transition function, $R(s, a)$ the reward function, and $\gamma \in [0, 1]$ the discount factor. The agent selects actions according to a policy $\pi(a|s)$ to maximize the expected return:

$$J(\pi) = \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t) \right]$$

Two central concepts in RL are the state-value function and the action-value function. The state-value function $V_\pi(s)$ measures the expected return from state s while following policy π:

$$V_\pi(s) = \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t R_{t+1} \mid S_0 = s \right]$$

The action-value function $Q_\pi(s, a)$ evaluates the expected return of taking action a in state s and then following policy π:

$$Q_\pi(s, a) = \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t R_{t+1} \mid S_0 = s, A_0 = a \right]$$

While V_π evaluates the desirability of states, Q_π directly guides action selection by considering state-action pairs, forming the basis for many RL algorithms.

Deep Reinforcement Learning (**DRL**) extends classical RL by using neural networks to approximate π, V, or Q. Methods such as Q-learning estimate $Q^*(s, a)$ directly, while actor–critic approaches maintain both a policy network (actor) and a value estimator (critic) to stabilize training [51]. DRL has been applied to robotic tasks including locomotion, grasping, and navigation, especially in simulation [52]. However, real-world application remains challenging due to sample inefficiency, safety concerns, and the sim-to-real gap [53].

To overcome sample inefficiency, safety concerns, and the sim-to-real gap of DRL, several strategies have been developed. **Off-policy algorithms** such as Deep Deterministic Policy Gradient (DDPG) [54] and Soft Actor-Critic (SAC) [55] reuse past experience stored in a replay buffer, allowing multiple updates per interaction and improving sample efficiency. SAC further enhances stability through a maximum entropy objective, which encourages exploration while maintaining robustness.

Model-based RL reduces the need for real-world trials by learning a probabilistic model of the environment for planning and policy optimization. PILCO [56], for example, uses gaussian processes to model dynamics and uncertainty, enabling highly data-efficient policy learning.

Domain randomization helps bridge the sim-to-real gap by training policies under diverse simulated variations, such as randomized lighting, textures, or physics [57]. Policies trained in this way can generalize to real-world conditions without extensive fine-tuning. This approach is also used in this thesis to learn the extrinsic parameters of BTMG skills, ensuring that policies remain robust to environmental variations.

Finally, **hierarchical RL** decomposes tasks into reusable sub-skills, enabling learning at multiple abstraction levels. Methods like HIRO [58] learn high-level goals and low-level controllers in an off-policy, data-efficient manner, improving both modularity and transferability.

By combining these ideas, off-policy learning for sample efficiency, model-based planning for data reduction, domain randomization for sim-to-real robustness, and hierarchical decomposition for modularity, modern DRL approaches can better support real-world robotic deployment.

Policy Search and Parameter Learning

In skill-based robotic systems, policies are often parameterized by vectors θ that define motion primitives or low-level control modules, such as pose offsets, stiffness, or insertion velocities. Policy search aims to find the optimal parameters θ^* that maximize task performance:

$$\theta^* = \arg\max_{\theta} \mathbb{E}[R(\tau_\theta)]$$

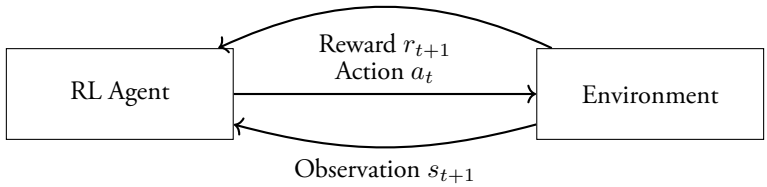


Figure 8: General reinforcement learning framework. The agent interacts with the environment by taking actions and receiving observations and rewards.

where τ_θ denotes the trajectory induced by parameters θ.

In many robotic scenarios, the policy structure is *non-differentiable*, for instance when execution involves BTs with discrete branching or conditional logic. This prevents the use of classical gradient-based methods, since gradients cannot propagate through symbolic decision nodes or discontinuous dynamics. Instead, gradient-free optimization methods are employed.

One common approach is **bayesian optimization** (**BO**), which builds a surrogate model of the reward landscape, often a gaussian process (GP), to propose informative parameter samples for evaluation [59]. BO is particularly effective for expensive or noisy evaluations, and in our work, it is used to tune the extrinsic parameters of BTMGs. We also enhance BO with domain randomization during training to ensure the learned parameters generalize across variations in task context.

Evolutionary strategies (**ES**) represent another class of gradient-free, population-based methods [60]. They maintain a population of candidate policies, perturb parameters using stochastic mutations, evaluate them in parallel, and update the population based on fitness. Population-based search is robust to noise, discontinuities, and long-horizon credit assignment challenges, making ES a scalable black-box optimization method. While powerful, ES generally requires more evaluations than BO for low-dimensional policy spaces like BTMG parameter tuning.

Another approach is **random search**, which samples policy parameters uniformly across the search space [61]. While conceptually simple, random search can perform surprisingly well in low-dimensional settings and often serves as a useful benchmark when evaluating more sophisticated optimization strategies.

To improve generalization across task contexts, **contextual policy search** [62] learns a mapping $\theta = f(c)$ from context variables c (e.g., object pose or geometry) to optimal policy parameters. This approach allows a single policy model to adapt its parameters on-the-fly when faced with new task conditions. Similarly, **neural policy regressors** [63] directly learn this mapping using supervised learning or few-shot adaptation, enabling skill parameter-

ization for unseen scenarios. BO can be extended with this principle by conditioning the GP surrogate on context variables, enabling data-efficient learning of parameterized skills for varying environments.

In this thesis, we primarily adopt BO due to its sample efficiency and suitability for low-dimensional parameter spaces in BTMGs. Combined with domain randomization, BO allows robust and generalizable policy parameter learning while maintaining modularity and interpretability, complementing the symbolic planning layer and supporting real-time task adaptation.

3.2 Bayesian Optimization for Efficient Policy Search

BO [64] is a sample-efficient approach for optimizing expensive black-box functions, particularly suited to robotics where function evaluations can be slow, costly, and derivative-free. BO maintains a surrogate model, often a GP, which provides a predictive mean $\mu(x)$ and variance $\sigma^2(x)$ over the input space $\mathcal{X}$. An acquisition function $\alpha(x)$ then uses these predictions to decide where to evaluate next, balancing exploration of uncertain regions with exploitation of promising candidates.

Formally, BO seeks to maximize an unknown function $f(x)$:

$$x^* = \arg\max_{x \in \mathcal{X}} f(x),$$

with the next evaluation point selected by

$$x_{t+1} = \arg\max_{x \in \mathcal{X}} \alpha(x; \mu(x), \sigma(x)).$$

Several acquisition functions are commonly used. *Expected Improvement (EI)* [65] evaluates the expected gain relative to the current best observation, naturally trading off exploration and exploitation. *Upper Confidence Bound (UCB)* [66] instead selects x with the highest $\mu(x) + \beta\sigma(x)$, where β controls the exploration–exploitation balance; larger β favors exploring uncertain regions. *Probability of Improvement (PI)* [67] measures the likelihood that a candidate will improve upon the best-so-far value, but it tends to overexploit unless the improvement threshold is carefully chosen. In practice, EI is widely adopted for its robustness, while UCB is preferred in scenarios requiring explicit exploration control. PI is simple but risk-averse and is often paired with adaptive thresholds.

For multi-objective optimization, BO can be extended with acquisition functions such as *Expected Hypervolume Improvement (EHVI)* to focus sampling on pareto-optimal regions (see Section 3.3) [68]. These mechanisms make BO effective for policy parameter tuning, controller adaptation, and robot behavior optimization, where each evaluation is costly and uncertainty-guided exploration is crucial.

3.3 Multi-objective Optimization and Reward Design

Many robotic tasks inherently involve conflicting objectives. For example, achieving reliable task success often requires applying sufficient force or motion speed, yet excessive force may compromise safety, energy efficiency, or hardware longevity. In the context of this thesis, where we investigate adaptive skill execution for task variations (**RQ2**), multi-objective optimization provides a principled framework to navigate such trade-offs. Instead of compressing multiple performance criteria into a single scalar reward, which can obscure important compromises, we explicitly consider multiple objectives to identify policies that represent desirable trade-offs across success, safety, and efficiency.

Motivation and Formulation

Multi-objective optimization seeks solutions that are *pareto-optimal*, meaning that no objective can be improved without degrading another [69, 70]. Formally, given k objectives $f_1, \ldots, f_k$ and a policy set Π, the problem is:

$$\max_{\pi \in \Pi} \big(f_1(\pi), f_2(\pi), \ldots, f_k(\pi)\big)$$

A solution $\pi^* \in \Pi$ is pareto-optimal if there is no other $\pi \in \Pi$ that improves at least one f_i without lowering another [69, 70]. The image of all pareto-optimal policies is called the *pareto front*, representing the set of non-dominated trade-offs among objectives. In practice, multi-objective optimization does not always return a fully connected front, but rather a pareto set (or approximation) that reflects achievable trade-offs [71]. This front is useful because it enables explicit selection of a policy according to application-specific preferences without collapsing all metrics into a single scalar.

Figure 9 shows a learned pareto front for a peg-in-hole task, where the objectives are insertion success and contact force. Each color corresponds to an independent optimization run, illustrating the range of achievable compromises. In this work, we use BO to explore the low-dimensional parameter space of the BTMG skills. BO proposes candidate BTMG parameter configurations, such as insertion velocity or compliance settings, evaluated under multiple objectives, which allows us to construct the pareto front and select policies that best align with desired performance profiles.

Common strategies for learning pareto fronts in multi-objective optimization include scalarization, multi-objective bayesian optimization (MOBO), and evolutionary algorithms. **Scalarization** reduces multiple objectives into a single scalar reward via a weighted sum, $r = \sum_i w_i f_i(\pi)$ [70]. While simple and computationally efficient, scalarization is highly sensitive to the choice of weights and can fail to capture non-convex regions of the pareto

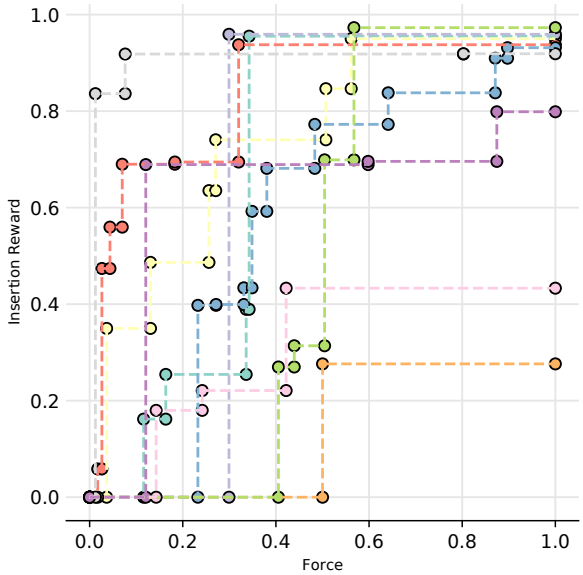


Figure 9: Example pareto front showing trade-off between insertion success and contact force in a peg-in-hole task [6]. Each color indicates a distinct optimization run or configuration.

front. This means that exploring diverse trade-offs often requires multiple weight configurations.

Multi-objective bayesian optimization treats each objective independently using surrogate models, typically GPs, and selects candidate evaluations using specialized acquisition functions. EHVI [72] is a prominent choice, as it maximizes the expected increase in the dominated hypervolume [68]. MOBO is particularly sample-efficient and is well-suited for robotic applications where function evaluations (e.g., real-world trials) are costly or risky.

Evolutionary algorithms, such as NSGA-II [73], evolve a population of candidate solutions over generations using selection, crossover, and mutation to approximate the pareto front. These methods excel at preserving diversity and handling complex, non-convex pareto fronts but typically require a large number of evaluations, which can be impractical for physical robotic experiments.

In this thesis, MOBO is particularly used to optimize BTMG parameters under multiple objectives. This allows discovering parameter configurations that balance task success, safety, and efficiency, while minimizing real-world trials. The resulting pareto front provides decision makers with a set of policies reflecting different trade-offs, from conservative but safe execution to faster but riskier behavior.

Reward Design Strategies

Reward design is critical in robotic learning because it directly shapes agent behavior and affects both learning efficiency and safety. Poorly specified rewards can lead to unintended or suboptimal behaviors, slow convergence, or unsafe actions [74]. We discuss it here because effective reward structures are fundamental for both single- and multi-objective optimization in our thesis, influencing the performance of bayesian optimization and policy learning.

In this thesis, we primarily use **sparse**, **dense**, and **shaped** rewards. Sparse rewards provide feedback only upon task completion, such as indicating success or failure of a behavior tree execution. While simple and unbiased, they often result in slow learning due to limited guidance. Dense rewards, in contrast, provide continuous feedback, for example, penalizing distance to goal or excessive contact forces, which accelerates convergence but may bias policies toward locally optimal behaviors. Shaped rewards combine these ideas by adding intermediate guidance or potential-based signals [75], helping to direct learning toward desirable behaviors without altering the optimal policy.

Our contact-rich manipulation tasks use composite reward functions to capture multiple aspects of task performance:

$$r(s, a) = w_1 r_{\text{success}} + w_2 r_{\text{force}} + w_3 r_{\text{energy}} + w_4 r_{\text{time}}$$

Here, r_{success} reflects task completion, r_{force} penalizes large contact forces, r_{energy} discourages high actuator effort, and r_{time} penalizes long execution durations. The weights w_i are manually tuned to balance success and safety. In our experiments, we primarily use sparse success rewards combined with dense force and energy penalties, with occasional shaping for intermediate guidance.

Other reward strategies also exist. **Curriculum learning** gradually increases task difficulty to guide progressive policy improvement [76]. **Constrained reinforcement learning** incorporates safety or resource constraints using auxiliary cost functions or Lagrangian penalties [77]. These methods are particularly relevant in safety-critical applications but are not directly applied in this thesis.

While scalarization of reward components is practical for single objective optimization, it can obscure trade-offs between competing objectives such as efficiency and safety. In scenarios with significant or unknown trade-offs, multi-objective learning and pareto-front exploration provides a more principled approach.

3.4 Gaussian Processes for Policy Generalization and Adaptation

GPs [78] are non-parametric bayesian models that define a distribution over functions. In robotics, they are widely used to generalize skill execution across task variations by learning a mapping from scenario variables (e.g., object position or size) to policy parameters.

Given training data $D = \{(x_i, y_i)\}_{i=1}^{n}$, where $x_i \in \mathbb{R}^p$ denotes a task context and $y_i \in \mathbb{R}^d$ the corresponding policy parameters, the GP defines a posterior over functions:

$$y(x) \sim \mathcal{GP}(m(x), k(x, x')),$$

with mean function $m(x)$ (often set to zero) and kernel $k(x, x')$ expressing similarity between task instances.

The GP posterior yields the following expressions for predictive mean and variance at a new input x_*:

$$\mu(x_*) = k(x_*, X)[K(X, X) + \sigma_n^2 I]^{-1} y,$$
$$\sigma^2(x_*) = k(x_*, x_*) - k(x_*, X)[K(X, X) + \sigma_n^2 I]^{-1} k(X, x_*).$$

At test time, the GP provides a predictive distribution over outputs:

$$p(y_*|x_*, D) = \mathcal{N}(\mu(x_*), \sigma^2(x_*)).$$

This distribution allows not only interpolation between known training points via the predictive mean $\mu(x_*)$, but also quantifies uncertainty via the variance $\sigma^2(x_*)$. In robotics, this enables confidence-aware adaptation, where high uncertainty may trigger fallback strategies or require human oversight, particularly in safety critical scenarios.

Applications and Complementary Strategies in Robotics

GPs are particularly effective for robotic applications [79, 80] that demand both data efficiency and principled uncertainty estimation. They are widely used to model contextual policies by mapping scenario features, such as object position, orientation, or size, to corresponding motion or control parameters [59]. This allows robots to generalize skills across task variations without retraining entire policies.

A key strength of GPs is their predictive posterior: the mean enables smooth interpolation between known parameterizations, while the variance quantifies model confidence. High variance identifies regions with limited training support, guiding active data collection or triggering conservative fallback strategies and human supervision in safety-critical scenarios. In our work [19], we exploit these properties to interpolate MG parameters in BTMG policies, achieving few-shot adaptation to varied peg-in-hole configurations with minimal data.

GP-based generalization can be further strengthened by complementary strategies. Domain randomization (DR) exposes policies to varied simulated environments, altering dynamics, friction, or sensor noise, to enhance robustness and improve sim-to-real transfer [81]. Meanwhile, robust RL benchmarks, such as the Robust Reinforcement Learning Suite (RRLS) [82], enable systematic evaluation of policies under perturbations and adversarial conditions, ensuring reliability in dynamic environments. Together, GPs, DR, and robust RL evaluation provide a practical and interpretable foundation for safe, data-efficient, and adaptable skill learning in real-world robotics.

4 Failure Recovery in Robotics

Robotic systems in real-world settings inevitably encounter execution failures caused by hardware malfunctions, sensor noise, environmental uncertainty, or task variations. Achieving reliable autonomy therefore requires mechanisms that can detect, diagnose, and recover from such errors, beyond what learning and adaptation alone can provide.

Failure recovery is especially important in contact-rich and dynamic environments, where small deviations can escalate quickly and compromise both safety and task success. This chapter reviews the core elements of failure recovery, failure detection, recovery strategy synthesis, and runtime adaptation, which together enable autonomous handling of both expected and unexpected deviations. These foundations support the unified framework evaluated in the subsequent chapters.

4.1 Failure Models and Detection

Failures in robotics are deviations from expected behavior that prevent successful task execution. They arise from three primary sources: *hardware failures*, *sensor failures*, and *execution failures*. *Hardware failures* involve mechanical or electrical malfunctions such as actuator wear, joint backlash, or power loss, typically requiring low-level diagnostics or physical maintenance. *Sensor failures* occur when measurements are corrupted, missing, or misleading due to occlusions, signal degradation, or calibration drift, which indirectly impair task performance.

This thesis primarily focuses on *execution failures*, which occur even when hardware and sensing are nominal. These failures often manifest in contact-rich tasks as misalignments, slips, or unmodeled environmental interactions caused by task variability or dynamic disturbances. At the symbolic level, they correspond to violations of task preconditions or the inability to satisfy postconditions during skill execution. For example, a grasp may fail if an object is slightly displaced, breaking its precondition, or an insertion may stall due to unexpected contact forces that prevent the postcondition from being met.

Detecting execution failures requires methods that continuously track both the robot's physical behavior and symbolic task state to identify deviations early. The following sections review representative strategies for failure detection.

Model-Based Detection

Model-based approaches predict nominal system behavior using analytical or physical models and detect failures through residuals, the discrepancies between expected and observed

states. Observer-based techniques, such as Kalman filters [83] and sliding-mode observers [84], as well as parity-space methods [85], provide low-latency detection with strong theoretical guarantees. However, they depend on accurate dynamics and noise models, which are difficult to obtain in unstructured environments [86].

Data-Driven Detection

Data-driven methods learn patterns of failure directly from sensory data. Cho et al. [87] trained neural networks to detect actuator faults from raw torque signals. RECOVER [88] combines multimodal perception (RGB, depth, force) with symbolic conditions to train classifiers for manipulation error detection. Khansari et al. [89] modeled obstacle avoidance dynamics to detect anomalies in real time. These approaches generalize well to new scenarios but often require large labeled datasets covering diverse failure cases.

Hybrid Approaches

Hybrid methods combine analytical structure with learning flexibility to improve generalization and reliability. Inceoglu et al. [90] introduced FINO-Net, which fuses interpretable task-level cues with proprioceptive and tactile representations, enhancing detection of manipulation failures. Xu et al. [86] proposed FAIL-Detect, a two-stage hybrid framework that extracts scalar confidence signals and applies conformal prediction to flag deviations with statistical guarantees, reducing reliance on explicit failure labels.

Multimodal Sensor Fusion

Multimodal sensor fusion integrates complementary sensing modalities, vision, force, tactile, and proprioception, to detect subtle or ambiguous anomalies that unimodal methods may miss. FINO-Net [90] exemplifies this approach by integrating RGB and force-torque signals through modality-specific CNNs and temporal convolutions, improving detection of object slippage and grasp misalignment. REFLECT [91] similarly fuses RGB-D, audio, and contact data into hierarchical summaries, structured history traces that can be queried by an LLM for failure explanation and recovery. DoReMi [92] and AHA [93] extend this idea by combining multimodal perception with language grounding, enabling context-aware reasoning over ambiguous failures.

The unified framework proposed in this thesis [3] follows a multimodal design: symbolic checks from behavior trees are combined with visual inference from VLMs. This joint processing of symbolic and perceptual signals enables early detection of execution failures and supports real-time recovery, as detailed in Section 11.

In summary, multimodal and hybrid detection approaches enhance robustness and interpretability in unstructured environments, where relying on a single modality or purely model-based reasoning often leads to brittle or incomplete failure detection.

4.2 Failure Recovery Strategies: Reactive vs. Proactive

Robotic systems can recover from failures using two main strategies: *reactive recovery*, which responds after an error occurs, and *proactive recovery*, which aims to anticipate and prevent failures. This distinction has been formalized in recent surveys [94, 95] and studied extensively in autonomous manipulation [91].

Reactive Failure Recovery

Reactive recovery restores task execution after a failure through retries, adaptation, or corrective planning [96]. These methods are essential in dynamic environments where unexpected disturbances can disrupt execution.

Early approaches rely on BTs with predefined retry and recovery branches. Wu et al. [97] automate modular recovery in mobile manipulation by triggering error handlers such as reattempts or resets.

Learning-based policies offer greater adaptability when hardcoded strategies fail. Lee et al. [98] trained a quadrupedal robot with deep reinforcement learning to recover from falls and external pushes without explicit failure modeling. Booher et al. [99] proposed CIMRL, combining imitation learning for efficient initialization with reinforcement learning for long-term correction in autonomous driving.

Neuro-symbolic and VLM-driven systems enable more flexible recovery. RECOVER [88] monitors each executed action with an ontology-based sub-goal verifier, then invokes an LLM re-planner to generate corrective actions online. ReplanVLM [100] performs dual-loop error correction, where the external loop reacts to observed failures by regenerating task plans. DoReMi [92] delegates constraint generation to an LLM, while a VLM monitors execution and flags violations, enabling semantic diagnosis of the failed goal. MultiReAct [101] uses a CLIP-based visual reward signal to monitor subtask completion and triggers corrective behavior on drops in confidence.

Language-based methods extend reactive recovery to high-level reasoning. AHA [93] interprets visual and textual traces to classify failures and propose planner-level corrections. REFLECT [91] leverages hierarchical execution summaries to prompt LLMs for reactive recovery suggestions. RACER [102] and SC-MLLM [103] use language-conditioned policies

that detect mismatches between expected and observed states and reissue corrected commands in real time.

Together, these methods span a spectrum from simple retries to adaptive learning controllers and neuro-symbolic or VLM-driven replanning, enabling recovery from both low-level control errors and high-level task deviations.

Proactive Recovery

Proactive strategies prevent failures by validating constraints or predicting risky states before execution. They are less common due to the difficulty of modeling unstructured environments and anticipating all failure modes.

Alvanpour et al. [104] use SHAP-based explainable models to predict grasp failures from contextual cues. Diehl et al. [105] apply causal reasoning to compute counterfactuals, identifying minimal interventions to avoid failure. Curriculum-based methods like CurricuLLM [106] also exhibit proactive behavior by structuring training data to reduce exposure to unsafe or failure-prone states.

Spanning Both Categories

Some frameworks integrate proactive checks with reactive adaptation to enhance robustness. Code-as-Monitor [95] compiles user-defined constraints into executable monitors that validate tasks pre-execution and continue checking them during runtime, halting or providing semantic feedback upon violations. Our unified framework [3] similarly combines pre-execution validation with real-time recovery: VLM-guided symbolic planning ensures initial feasibility, while runtime monitoring triggers behavior tree modifications or replanning when failures occur (see Sections 10 and 11).

4.3 Recovery-Behavior Synthesis

Detecting a failure is not sufficient for robust execution; the system must also generate a corrective policy to restore task feasibility. Recovery behavior synthesis converts detection signals into actionable plans, enabling the robot to resume or complete the task. Approaches fall into three broad categories: rule-based, learning-based, and hybrid neuro-symbolic methods.

Rule-based synthesis

Symbolic planners generate corrective behaviors from skill annotations defining preconditions and postconditions. Reactive planners can backchain from the failure point to construct a BT that restores goal feasibility [10, 107]. Frameworks like SkiROS2 support modular recovery using this approach, which is widely applied in manipulation, navigation, and human-robot interaction [23, 32]. Rule-based synthesis is fast, interpretable, and verifiable but brittle when facing novel or ambiguous failures.

Learning-based synthesis

Learning-based methods derive recovery policies from data rather than hand-coded logic. RecoveryChaining [108] employs hierarchical reinforcement learning to recover from execution failures, using a hybrid action space that switches between primitive actions and nominal skills to return the system to a solvable state. Inverse Reinforcement Learning from Failure (IRLF) [109] infers reward functions from both successful and failed demonstrations, helping the robot recognize and avoid states leading to failure. Composition of Conditional Diffusion Policies (CCDP) [110] leverages diffusion models conditioned on failure events, steering generated actions away from previously unsuccessful regions and composing multiple policies to handle long action sequences. These methods adapt well to diverse scenarios but often require large datasets and lack formal safety guarantees.

Hybrid synthesis

Hybrid approaches combine symbolic structure with learning-based adaptability. RECOVER [88] uses ontology-driven reasoning to identify failure sources and invokes an LLM-based re-planner to generate online recovery plans. ReplanVLM [100] performs dual-loop error correction, where external correction reacts to observed failures by regenerating updated task plans. Code-as-Monitor [95] compiles task constraints into executable runtime checks, providing verifiable structure that can support symbolic replanning but does not natively generate recovery behaviors.

Our unified framework [3] integrates both symbolic and data-driven components in a two-phase process. During planning, behavior trees are statically validated using VLM-generated summaries, and missing preconditions trigger automatic subtree insertion via reactive planning. During execution, a runtime monitor tracks skill outcomes and scene graph updates; if a violation is detected, the reactive planner grafts a corrective subtree, using VLM input to select or generate appropriate skills. Recovery skills are further optimized using reinforcement learning by tuning the parameters of BTMG representations [9].

Trade-offs

Rule-based methods are efficient and verifiable but brittle. Learning-based approaches generalize better across unmodeled conditions yet require extensive data and are harder to validate for safety. Hybrid methods seek to combine these strengths: RECOVER and ReplanVLM pair language and perception with symbolic reasoning, while Code-as-Monitor emphasizes verifiability at the cost of autonomous recovery generation. Our framework offers proactive BT validation and reactive real-time adaptation, balancing interpretability and generality, but depends on VLM accuracy and the completeness of the skill library.

In summary, synthesis methods trade off adaptability, verifiability, and efficiency. Robust systems benefit from blending structured planning with data-driven policies, provided that decision-making remains interpretable and computationally tractable.

4.4 Runtime Monitoring and Adaptation

Runtime monitoring is essential for robust robotic execution in dynamic and uncertain environments. It continuously evaluates whether the robot's actions align with expected task progress and anticipates or detects potential failures. By combining symbolic condition checks with perceptual scene understanding, monitoring ensures that deviations are detected early so that the system can respond before they compromise task success. This section reviews representative approaches to runtime monitoring and adaptation, which form the backbone of reliable failure handling.

Monitoring techniques

Traditional monitoring relies on threshold-based checks of sensor signals. For example, Shahsavari et al. [111] monitor quadcopter motor RPMs via a remote controller, where failures are detected using the ChangeFinder algorithm and logistic regression, triggering emergency control for safe landing.

Model-based approaches provide foresight by simulating future states. Liu et al. [112] learn a latent-space dynamics model and a failure classifier to predict potential out-of-distribution (OOD) states and upcoming failures within an interactive imitation learning framework. This predictive capability enables preemptive intervention and reduces the need for continuous human supervision.

Symbolic and semantic monitoring operates at a higher abstraction level. Code-as-Monitor [95] compiles task constraints into runtime assertions, effectively turning specifications into self-checking execution. AHA [93] uses a VLM to determine whether the current scene still

satisfies task preconditions and flags violations as potential failures.

These techniques collectively span low-level sensor checks, predictive modeling, and high-level semantic reasoning.

Adaptation mechanisms

Adaptation mechanisms define how a system responds after a failure or a risk of failure is detected. They involve modifying the robot's behavior to recover, either by switching to a safe state, replanning, or issuing corrective actions.

RecoveryChaining [108] trains hierarchical reinforcement learning policies to recover from execution failures. When a failure is detected (e.g., object slip or collision), the recovery policy steers the robot to a nearby safe state and then transitions control back to the nominal policy to complete the task. ReplanVLM [100] employs a dual-loop error correction strategy. Internal error correction inspects the current code and task plan to fix errors proactively, while external error correction reacts to observed failures by regenerating plans based on the updated environmental state. RECOVER [88] leverages ontology-driven symbolic reasoning to identify failure points, then invokes an LLM-based re-planner to generate a recovery plan online without resetting the environment.

Runtime monitoring and adaptation in our framework

Our framework unifies symbolic and perceptual monitoring with reactive adaptation in a single runtime loop [3]. At each BT tick, a VLM verifies pre- and postconditions against the scene graph. If a violation is found, the reactive planner grafts a corrective subtree, selecting from existing skills or generating new ones with VLM guidance. The scene graph and execution history are updated continuously, enabling causal reasoning and real-time recovery without interrupting execution. See Section 11 for further details.

5 Vision-Language Models for Robot Reasoning

VLMs are transforming robotic reasoning by unifying perception and decision-making. They jointly encode visual and textual inputs into a shared representation, allowing robots to interpret scenes, understand instructions, and generate task-relevant insights in complex environments.

By aligning visual observations with language-based semantics, VLMs enable robots to extract symbolic cues, reason about affordances, infer goals, and support dynamic replanning or failure recovery, capabilities that go beyond hand-crafted perception pipelines and predefined rules. These strengths make VLMs well-suited for embodied agents requiring interpretable, adaptive, and data-driven reasoning in real-world tasks.

This section introduces the foundations of VLMs and their applications in robotics. We first review the technical architecture of modern VLMs, then examine their role in instruction following, policy generation, scene understanding, and failure handling, showing how they support both reactive and deliberative reasoning within the unified framework of this thesis.

5.1 Technical Foundations of Vision-Language Models (VLMs)

Foundation models are large-scale pre-trained neural networks that generalize across diverse tasks with minimal task-specific adaptation [113]. In natural language processing, autoregressive models such as GPT-3/4 [114, 115] and PaLM [116] exhibit strong generative and reasoning abilities, while masked language models like BERT [117] and T5 [118] excel at contextual representation learning. Extending this paradigm, VLMs jointly encode visual and textual inputs, enabling tasks such as image captioning, visual question answering, semantic scene understanding, and high-level reasoning for robotics.

A typical VLM consists of an image encoder (e.g., ResNet [119] or ViT [120]), a text encoder (e.g., BERT or GPT-style transformer), and a multimodal fusion module [121]. Based on the fusion strategy, VLMs are generally categorized into early-fusion, late-fusion, and cross-modal attention models [122, 123]. Early-fusion models combine low-level feature embeddings from multiple modalities at the input stage, which allows joint learning but risks interference when signals are misaligned. Late-fusion models process each modality independently before merging their high-level representations, preserving modality-specific learning but limiting cross-modal interactions. Cross-modal attention models, such as MulT [122], allow one modality to attend to another during intermediate layers, yielding richer joint representations and supporting temporal and spatial reasoning.

Building on these fusion strategies, VLMs are further divided into contrastive and generative families. Contrastive VLMs, such as CLIP [124] and ALIGN [125], align image

and text embeddings by maximizing the similarity of paired samples and minimizing it for unpaired ones. This approach enables open-vocabulary classification, zero-shot recognition, and image-text retrieval. Generative VLMs, including Flamingo [14], BLIP-2 [126], and MiniGPT-4 [127], adopt encoder-decoder or decoder-only architectures that generate textual or multimodal outputs conditioned on visual inputs. This design supports tasks such as captioning, visual question answering, and instruction following. Representative architectures are shown in Figure 10.

VLMs are typically trained on large-scale image-text datasets such as LAION-400M [128] and CC3M [129], using objectives that include contrastive loss, masked language modeling, image-text matching, and next-token prediction [126, 127]. Recent methods improve efficiency by freezing vision encoders and training lightweight adapters or Q-Former modules [126]. Some approaches further integrate external LLMs, such as Vicuna or OPT, enhancing language reasoning while grounding visual inputs through projection heads and prompt tuning [127, 130].

In robotics, VLMs bridge low-level perception and high-level reasoning by mapping raw visual inputs to structured or symbolic representations. This alignment of perceptual features with language-based semantics enables object grounding, semantic scene understanding, task planning, and failure recovery. Consequently, VLMs provide a powerful foundation for interpretable and adaptive control in real-world robotic systems [131, 132].

5.2 Applications in Robotics

VLMs are becoming central to embodied AI because they unify visual perception with language-grounded reasoning, allowing robots to interpret instructions, perceive complex environments, and make context-aware decisions. Their applications in robotics can be grouped into four main areas.

Instruction following involves translating high-level natural language commands into actionable robot behaviors. Early examples such as SayCan [133] and PaLM-E [11] demonstrated zero-shot execution on mobile-manipulation platforms by connecting language understanding to waypoint-level control. More recent systems like OpenVLA [134] and RT-2 [17] extend this approach by fine-tuning vision–language–action (VLA) models on large-scale video–text datasets, enabling robots to follow instructions with minimal task-specific retraining.

Policy generation from multimodal context builds on this capability by conditioning diffusion or autoregressive policies directly on both language and egocentric visual inputs. Models such as PerAct [135], VIMA [136], and RT-1 [16] use this approach to perform table-top manipulation and tool-use tasks without the need for explicit reprogramming,

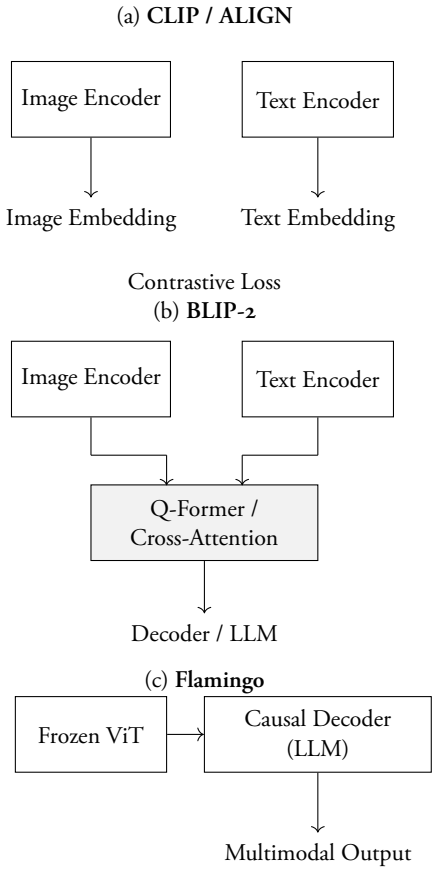


Figure 10: Representative VLM architectures. (a) CLIP-style models align image and text embeddings through a contrastive objective. (b) BLIP-2 uses a Q-Former to fuse modalities before decoding with an LLM. (c) Flamingo combines a frozen vision encoder with a causal language decoder to produce multimodal outputs.

demonstrating how VLM-based policies can generalize across scenarios.

Dynamic replanning and failure handling uses VLMs at runtime to recover from deviations or errors. ReplanVLM [100] and AHA [93] detect goal divergence and propose revised action sequences by querying a frozen VLM with updated environmental context. Our own work [8] follows a similar principle but integrates VLM queries into a behavior tree execution framework, where unmet preconditions are automatically reformulated as language prompts for corrective skill insertion.

Scene understanding and affordance reasoning focuses on extracting structured representations from complex visual scenes. CLIPORT [131] and the scene graph pipeline of Chen

et al. [137] convert RGB-D observations into symbolic object–affordance graphs, which downstream planners exploit for pick-and-place, tool selection, and navigation.

Together, these applications illustrate a shift from rigid, hand-coded pipelines toward data-driven, interpretable, and context-aware robotic reasoning, where VLMs provide the semantic bridge between perception, planning, and action.

5.3 Scene Understanding and Goal Inference

Scene understanding is crucial for autonomous robots operating in unstructured environments, as it provides the semantic context needed for decision-making. VLMs facilitate this process by aligning visual inputs and language prompts in a joint embedding space [124, 138], enabling zero-shot recognition, object localization, and scene interpretation (see Section 5.1).

Models such as CLIP [124] and ALIGN [138] can match textual queries to visual regions and generalize across tasks like classification, segmentation, and captioning [139]. In robotics, this ability allows agents to extract task-relevant cues from cluttered or ambiguous environments, supporting both action selection and planning. Systems like SayCan [133] and VIMA [136] leverage this capability to score the feasibility of actions and ground plans in perceptual context, effectively linking what the robot sees to what it can do.

Recent work extends VLM outputs into structured semantic representations such as scene graphs or object–affordance maps [137]. These representations capture which objects can be manipulated, what actions are plausible, and which goals are implicitly supported by the current environment, enabling downstream planners to operate without handcrafted perception modules. This approach aligns with this thesis's focus on integrating symbolic planning with VLM-informed scene understanding.

By transforming raw multimodal inputs into high-level semantic context, VLMs provide the foundation for both reactive control and deliberative goal inference, bridging perception and reasoning in real-world robotics.

5.4 Failure Detection and Explanation with VLMs

VLMs provide a flexible mechanism for high-level introspection in robotic systems by identifying and explaining task failures directly from visual and contextual input. Unlike traditional rule-based monitors that rely on predefined conditions, VLMs can reason over open-ended observations to detect unexpected deviations and offer interpretable explanations. Building on the failure models introduced in Section 4.1, this subsection examines

how VLMs enable detection and explanation through perceptual queries and symbolic reasoning.

A common strategy is to compare expected and observed scene states using open-vocabulary object recognition. Models like CLIP [124] can detect missing or misplaced objects without requiring explicit class labels, identifying discrepancies such as "cup not found on tray" or "object grasped incorrectly" via prompt-based queries [139]. This capability allows robots to flag execution errors even in previously unseen scenarios.

Beyond simple visual checks, VLMs can participate in goal verification and policy monitoring loops. Systems such as SayCan [133] and VIMA [136] prompt VLMs with templates like "Is the target object present?" or "Was the goal achieved?" to detect precondition violations and assess task progress. This enables structured failure reporting and supports the triggering of fallback actions when tasks deviate from their intended plan.

Our framework [8] extends this idea to real-time detection of unknown failures. It continuously monitors preconditions and postconditions during task execution, using VLM outputs to interpret the cause of anomalies and to guide corrective actions, including dynamic insertion of missing skills. This approach combines visual cues with symbolic context to recover from both anticipated and unanticipated execution errors.

By transforming multimodal observations into actionable diagnostic insights, VLMs enable generalizable and interpretable failure detection. Their ability to detect discrepancies without exhaustive supervision makes them a powerful tool for robust execution in dynamic and unpredictable environments.

Part I – Adaptive Skill Learning for Robot Autonomy

This part of the thesis addresses how to design robot policies that are easy to configure, robust in performance, and suitable for long-term deployment. This includes three research directions: (i) structuring behavior policies so they remain modular, interpretable, and tunable (RQ1.1), (ii) adapting these policies to new task variations without retraining by learning task-parameter mappings (RQ1.2), and (iii) efficiently predicting optimal parameters for unseen tasks on-the-fly to enable responsiveness in real-world settings (RQ1.3).

6 Structuring Robot Policies for Modularity, Interpretability, and Data-Efficient Learning

This chapter addresses **RQ1.1**: *How can robot behavior policies be structured to remain modular, interpretable, and tunable for varying tasks?* We use the BTMG policy formulation, which combines the symbolic structure of behavior trees with parameterized motion generators, to design policies that are both interpretable and adaptable. **Paper I: Skill-based Multi-objective Reinforcement Learning of Industrial Robot Tasks with Planning and Knowledge Integration** supports our answer to this question.

6.1 Motivation and Positioning within Existing Work

Robotic control can be analyzed at multiple abstraction levels, ranging from motion trajectories to symbolic skills and task-level plans (see Section 1). Low-level controllers (e.g., policies in RL [50]) generate trajectories by mapping states to actions, while symbolic policies (e.g., skill sequences [6]) operate over semantically meaningful actions. Many existing robot systems are either hard-coded at the symbolic level (offering no adaptability) or learned directly at the trajectory level through end-to-end methods such as deep neural

policies, which typically require large amounts of data, are hard to debug, and produce opaque behavior [140]. Data-driven motion primitives such as DMPs [141, 142, 143, 20, 144] offer interpretable, modular trajectory representations. However, they operate at the motion level and do not natively support symbolic composition or task-level reasoning.

Industrial robots, in particular, must balance safety, predictability, and adaptability. They need to be quick to reconfigure for new tasks, easy to understand, and able to exploit digital twins and symbolic knowledge about their environment [43, 28, 145]. These requirements are difficult to meet simultaneously. Conversely, manually engineered control strategies are brittle and inflexible when the task changes [146].

Several prior works attempt to bridge symbolic planning with learning. Approaches like PLANQ-learning [147], PEORL [148], SPOTTER [149], and HIP-RL [150] combine symbolic planners with model-free reinforcement learning but are mostly applied in discrete or simulated environments. Other methods such as [46] evolve behavior trees using genetic programming. In contrast, in Paper I, we retain a fixed BT structure and focus on optimizing a small number of interpretable parameters in leaf-level motion generators. Later works (Papers V and VI) extend this approach by allowing automatic generation and extension of the BT structure itself.

Our work also contrasts with black-box policy representations [59, 140], offering instead a human-readable symbolic scaffold with semantic parameter labels to support policy selection. Compared to DMPs, which encode point-to-point motions using attractor dynamics, our method operates at a higher level of abstraction. While DMPs have proven effective for trajectory generalization, they remain at the motion level and do not support symbolic reasoning or task-level composition. In contrast, we wrap MGs inside symbolic skills and compose them using BTs, enabling structured, explainable, and reactive execution across multi-step tasks.

RQ1.1 asks how to design robot behavior policies that are both interpretable and adaptable to specific task contexts. Interpretable policies benefit from structured representations that humans can read and reason about. Adaptability requires tuning motion behavior to match the demands of a task instance. We adopt a layered representation where a behavior is composed of symbolic skill sequences with tunable parameters. These parameters often need to satisfy multiple objectives, such as maximizing success while minimizing force, which motivates the use of data-efficient optimization. We use symbolic planning to generate parameterized skill sequences and adapt those parameters using multi-objective bayesian optimization. Before presenting our approach in detail, we discuss how this structure supports interpretability, tunability, and modularity in line with RQ1.1.

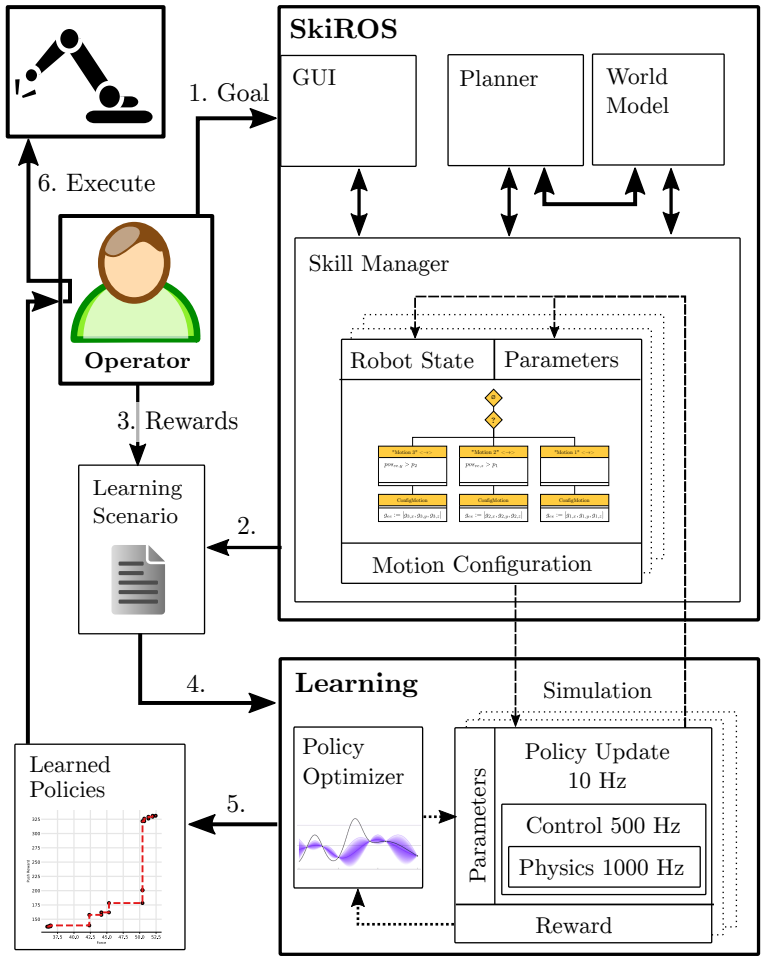


Figure 11: Overview of the skill–planning–learning pipeline (adapted from Paper I, Fig. 2). A symbolic goal is translated into a sequence of skills, each represented using the BTMG format. Tunable parameters of motion generators are optimized using MOBO, and the resulting pareto-optimal policies are deployed on the robot.

6.2 Structuring Policies with BTMG and Parameter Optimization

Our solution builds on symbolic BTs, where some leaf nodes invoke MGs with tunable parameters (See Section 2.4). This structure provides a natural separation between symbolic logic and continuous control. The skill parameters exposed for learning such as insertion force, spiral radius, or push offset are referred to as *extrinsic parameters* of the BTMG representation.

These parameters are tuned using bayesian optimization in a multi-objective setting, where

multiple competing goals (e.g., task success, speed, force) must be balanced. We use BO due to its sample efficiency and ability to handle noisy and expensive evaluations, especially in robotics. This learning layer complements the symbolic BT structure, enabling data-efficient and targeted adaptation of specific behaviors.

The key characteristics of the approach are:

- **Modularity:** The BT encodes the policy as a sequence of reusable symbolic skills.
- **Interpretability:** The BT structure is transparent and readable, while each parameter has a semantic label.
- **Tunability:** Parameters of motion generators can be adjusted to better match task-specific dynamics.
- **Data-efficient learning:** A small number of parameters are optimized using Bayesian Optimization.

This modular and structured formulation directly supports the goals in RQ1.1.

6.3 Method Overview

Our method integrates symbolic task planning with parameter optimization through learning. This optimization may be performed in simulation or on the physical robot. However, simulation is typically preferred due to safety, time, and other cost considerations.

The workflow is as follows (see Fig. 11):

1. **Goal specification:** The operator defines a symbolic goal using PDDL. For instance, for the peg-in-hole task, the goal is: `(skiros:atskiros:Peg-1skiros:BoxWithHole-1)`
2. **Plan generation and parameter identification:** A PDDL planner is used to generate a sequence of skills for the goal. Each skill is represented using the BTMG representation. The BT nodes that involve MGs are identified along with the parameters that require tuning. This information is specified in the SkiROS2 skill models, which define the motion generator used by each skill and list the tunable parameters exposed for optimization. (See Section 2.5).
3. **Learning scenario specification:** The user defines a JSON configuration file specifying which parameters to optimize, their value ranges, and the associated reward functions for each objective. An example configuration for the peg-in-hole task is shown in Listing 1.

4. **Reward assignment:** Each objective (e.g., success, force) is defined using one or more reward functions to guide policy learning.

5. **Learning phase:** Bayesian Optimization is used in a multi-objective setting to evaluate policies and learn optimal parameters (see Sections 3.3 and 4.4).

6. **Pareto front and deployment:** The learned policies are shown on a Pareto front, where each axis corresponds to an objective (e.g., success vs. force). Non-dominated policies (those not worse on all objectives) are retained. These policies can then be selected by the user and deployed directly on the robot.

```
{
   application_name :  peg_insertion_task ,
   optimizer :  hypermapper ,
   optimizer_config : {
     input_parameters : {
       PathVelocity : {  values : [0.0, 0.3],  parameter_type :  real  },
       PathDistance : {  values : [0.0, 0.03],  parameter_type :  real  },
       Radius :       {  values : [0.0, 0.07],  parameter_type :  real  },
       Force :        {  values : [-10, 0.0],   parameter_type :  real  }
    }
  },
   rewards : {
     FixedSuccessReward : {
       objective :  insertion_reward ,  type :  FixedSuccessReward ,  value :
          8.0
    },
     force_application : {
       objective :  force ,  type :  ForceApplicationReward ,  negative : true
    },
     GoalDistanceTranslationReward : {
       objective :  insertion_reward ,  type :  GoalDistanceTranslationReward
          , ...
    },
     linear_distance_to_box : {
       objective :  insertion_reward ,  type :  LinearDistanceToBoxReward ,
          ...
    }
  },
  ...
}
```

Listing 1: Excerpt from the JSON configuration used to define the peg-in-hole task. It specifies the parameters to be optimized (e.g., Force, Radius), their ranges, the optimization backend (in this case, HyperMapper), and the reward functions for each task objective. Each reward is linked to an objective such as insertion success or applied force.

6.4 Experimental Evaluation

To evaluate our approach, we consider two contact-rich tasks commonly encountered in industrial settings.

Peg-in-hole Task. The goal is to insert a peg into a box with a hole (Figure 12). The tunable parameters include *downward insertion force*, *path velocity*, *path distance*, and *spiral radius*. The last three belong to an Archimedean spiral strategy used to search for the hole when contact is uncertain. The objectives of the peg-in-hole task are: (i) maximize insertion success and (ii) minimize applied force to reduce damage risk.

Figure 12: Experimental setup for the peg-in-hole task, where a peg (black) is inserted into a box (orange) with a hole.

The first objective is captured using three rewards: a fixed success reward for successful insertion, a goal distance translation reward that increases with proximity to the hole, and a linear distance to box reward that encourages approaching the box. The second objective

(force minimization) uses a single reward that measures how much force is applied over time during the execution.

Push Task. The goal is to push a rectangular object to a specified goal (Figure 13). The learnable parameters include *start* and *goal offsets* in the x and y directions. The objectives of the push task are: (i) maximize push success and (ii) minimize applied force, to ensure effective and safe completion.

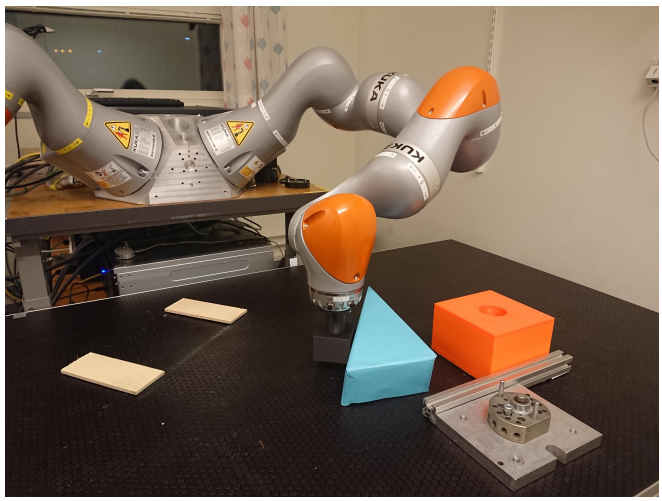


Figure 13: Experimental setup for the push task, where a rectangular block (cyan) is moved to a designated goal position.

The first objective uses three reward terms: goal distance translation (distance from object to goal), object position reward (alignment with the target), and object orientation reward (correct final orientation). The force objective uses the same force applied over time during execution metric as in the peg-in-hole task.

To benchmark the learning procedure, we compared against three baselines, chosen to reflect common industrial and research practices:

- **Planning-only:** symbolic plan with default skill parameters. This reflects the performance of purely deliberative planning without adaptation.
- **Random sampling:** parameter values sampled randomly within valid bounds. This serves as a sanity check for the difficulty of the search space and establishes a naive lower bound.

- **Robot operators:** parameters manually tuned by experienced roboticists. This represents expert-guided tuning, which is the current practice in many industrial setups [146].

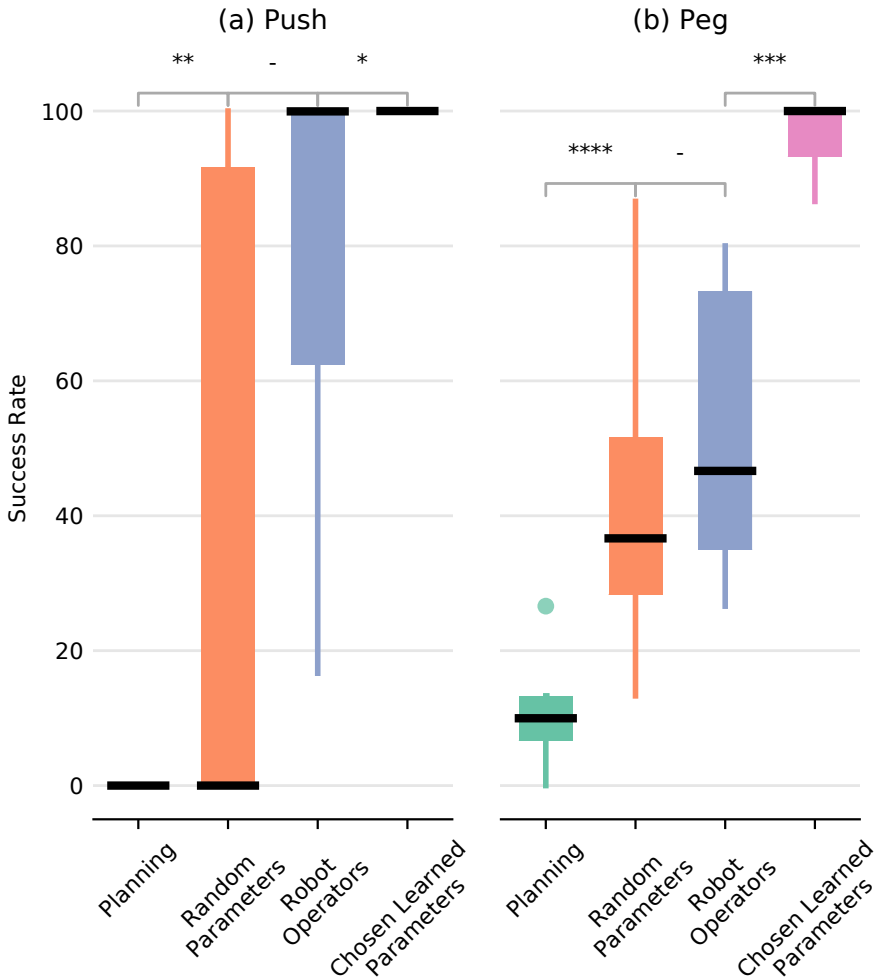


Figure 14: Success rate comparison against baselines for both tasks. Box-and-whisker plots show performance variability.

Our approach outperformed all three baselines (Figure 14). In the peg-in-hole task, learned policies achieved 96% success on unseen starts (versus 52% for operators), while using 16.6% less force and completing task 18.1% faster. In the push task, learned policies achieved 100% success across all test starts, including previously unseen configurations, whereas human operators only reached 50–100% success depending on their parameter set. Moreover, our

method eliminated the need for manual tuning, which took operators on average over 16 minutes and 11 trials per task. Full quantitative analysis and plots can be found in Paper I.

6.5 Discussion

Our approach answers to RQ1.1 by demonstrating how modular and interpretable BT-based policies can be automatically tuned using a small number of parameters, resulting in high task performance. The framework bridges symbolic planning with parameter optimization, providing both structure and adaptability.

One key insight is that semantic parameter names such as 'downward force' or 'spiral radius' enabled operators to interpret learned behaviors and meaningfully assess trade-offs when selecting between candidate solutions. This interpretability proved valuable in selecting solutions from the Pareto front without needing to inspect raw policy behavior. Another insight is that symbolic planning significantly constrained the search space by eliminating irrelevant or fixed parameters, thereby improving the efficiency and safety of the learning process.

At the same time, the approach has limitations. Reward functions still require expert knowledge to define and balance, particularly when multiple objectives are involved. Learning was performed at the level of individual skills, without modeling interactions or dependencies between them. Finally, the simulation-based learning relied on accurate contact dynamics, which may limit the transferability of policies to real-world settings in some scenarios.

Despite these limitations, the overall framework offers a practical and data-efficient path to producing modular, interpretable, and tunable behavior policies. In the next chapter, we extend this foundation to address RQ1.2 by learning to generalize such policies across varying task conditions.

7 Generalizing Modular Policies

This chapter addresses **RQ1.2**: *How can modular, interpretable robot policies be made to generalize over task variations without retraining?* Building on the BTMG policy formulation introduced in Chapter 6, we now extend this structure to adapt across multiple varying instances of a task by learning mappings from scenario parameters to optimal policy parameters. **Paper II: Generalizing Behavior Trees and Motion-Generator (BTMG) Policy Representation for Robotic Tasks Over Scenario Parameters** supports our answer to this question.

7.1 Motivation and Research Framing

While structured BTMG policies allow skill-level modularity and tuning (Chapter 6), they require optimizing parameters for each new variation of a task. For instance, if a robot learns to push an object to one location, it must relearn suitable parameters for each new goal. In real-world deployments, such as flexible manufacturing, this re-optimization is impractical because it incurs time and safety costs, disrupts workflow, and often requires repeated human involvement[145]. When task configurations frequently change, such overhead makes online adaptation infeasible. RQ1.2 asks: *Can we reuse the structure of a learned BTMG policy and generalize its parameters to new task variations without retraining?*

Our core idea is to model the relationship between observable task features (e.g., object pose) and behavior parameters (e.g., push offsets). Observable task features are variables describing the external task context, such as target object positions or goal poses, which we denote as *scenario parameters*. Behavior parameters, in contrast, are values within the BTMG policy (e.g., velocities, offsets) that determine how a skill is executed for that context. These are referred to as *extrinsic parameters* in our formulation (see Section 2.4).

Learning a mapping between scenario parameters and extrinsic BTMG parameters allows the robot to generalize its behavior across varying task instances by adapting motion execution without modifying symbolic structure. This approach draws inspiration from similar efforts in DMPs, where gaussian processes have been successfully used to generalize motion parameters over varying conditions [151, 152, 153, 78]. Since MGs and DMPs both operate at the trajectory level (see Chapter 1), we extend this idea to generalize parameterized BTMG policies.

Note: In this and subsequent chapters, we use the terms *scenario parameters* and *task variations* interchangeably to describe the same concept: observable variables defining the task instance.

7.2 Approach: Gaussian Process-Based Parameter Generalization

We use GP [78] regression to map scenario parameters to BTMG parameters. Each $\mathbf{s}$ defines a different task instance. The challenge is to find a function $f : \mathbf{s} \mapsto \boldsymbol{\theta}$ that maps from $\mathbf{s}$ to extrinsic BTMG parameters $\boldsymbol{\theta} \in \mathbb{R}^d$. The policy structure itself, represented as a behavior tree, remains fixed across task instances. Figure 15 illustrates this mapping.

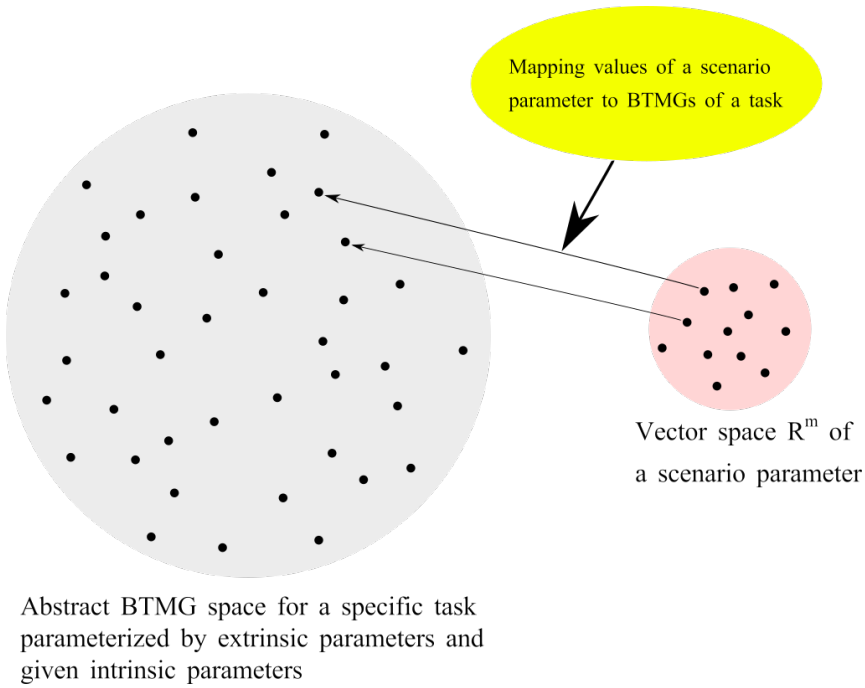


Figure 15: Mapping scenario parameter space (pink) to BTMG parameters (gray) using gaussian processes (yellow).

The training pipeline involves:

1. Collecting training pairs $(\mathbf{s}, \boldsymbol{\theta})$ using the same data collection strategy as in Chapter 6: BO is used to learn extrinsinc BTMG parameters for each training task instance.
2. Training a GP in a supervised learning fashion with scenario parameters as input and corresponding extrinsic BTMG parameters as output. This realizes the mapping function $f : \mathbf{s} \mapsto \boldsymbol{\theta}$.
3. At runtime, querying the GP with a new scenario parameter vector $\mathbf{s}$ to infer tuned extrinsic parameters $\boldsymbol{\theta}$ for the BTMG policy.

This allows one-shot parameter inference for novel task instances while preserving symbolic interpretability.

7.3 Experimental Setup

We evaluated this method on the same pushing task used in Chapter 6, where a triangular object must be moved to various 2D goal positions using a push skill.

However, we made one key modification: the goal corner used previously in Figure 13 was removed. This was done for two reasons. First, the corner simplified execution by guiding the object into the goal area, which could mask differences between parameter values. Second, in this chapter we introduce variable goal locations as scenario parameters, which made the corner unreliable and inappropriate for generalization.

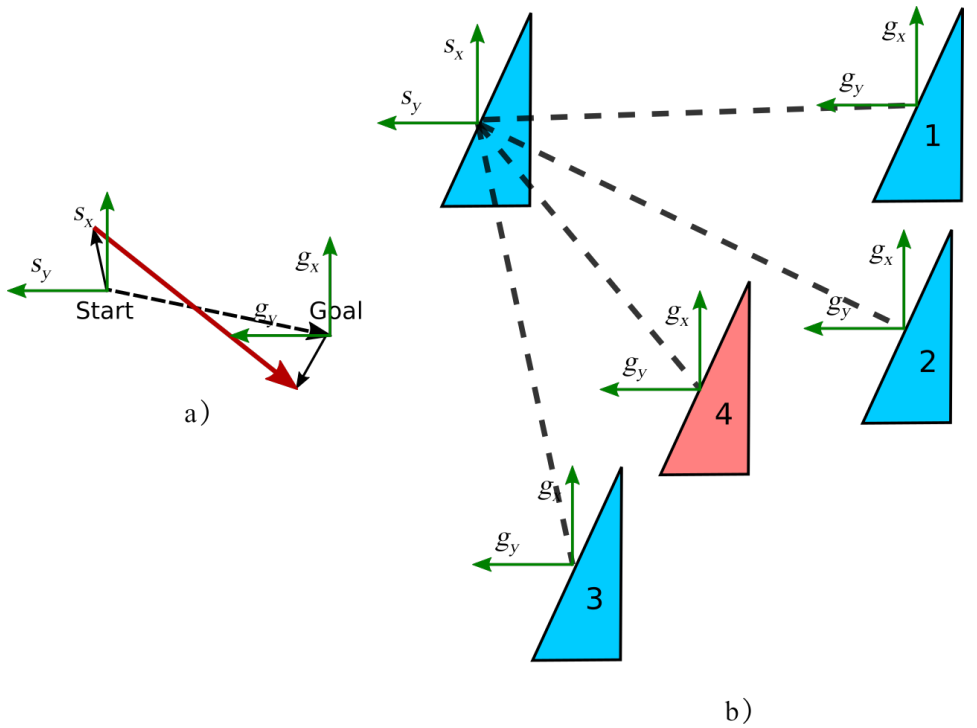


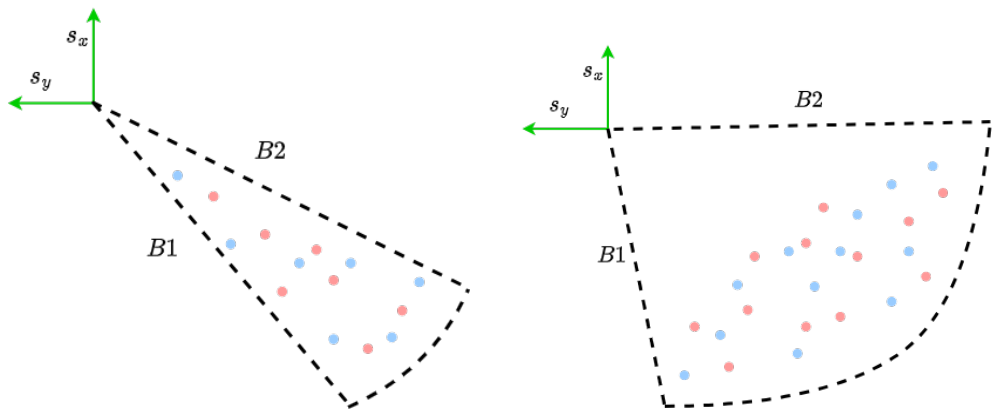
Figure 16: a) Red push vector defined by extrinsic offsets. b) Setup showing training and test goal locations.

To simplify the setup and focus on understanding the effect of task variation, we fixed the initial object position and varied only the 2D goal location. This goal position forms the scenario parameter vector $\mathbf{s}$. The push behavior is parameterized using four extrinsic

parameters, start and goal offsets in x and y directions (s_x, s_y, g_x, g_y) that together define the push vector (Figure 16). These offsets determine how the robot approaches and moves the object.

Training and Generalization: We generated training and test configurations using Latin Hypercube Sampling (LHS), a space-filling method that ensures efficient coverage of continuous domains [154]. Each sample corresponds to a unique 2D goal pose. For every configuration, we used BO (as in Chapter 6) to learn the BTMG extrinsic parameters. The resulting $(\mathbf{s}, \boldsymbol{\theta})$ pairs were used to train the GP model.

The GP was then queried with unseen goal poses to produce parameter predictions.



(a) Original space (GP performs well) (b) Expanded space (GP underfits)

Figure 17: Effect of scenario space size on generalization. a) GP learns smooth mappings in small space. b) GP underfits when scenario space is expanded and training data is insufficient.

7.4 Discussion

This chapter shows that modular BTMG policies can be generalized to new task instances using GP-based mappings from scenario parameters to extrinsic motion parameters. The learned mappings allowed the same symbolic policy structure to adapt across different goal configurations without retraining.

Our results suggest that GPs offer reliable generalization in low-dimensional scenario spaces with limited training data. This is beneficial for practical deployment, where collecting new data is costly. However, we observed that as the scenario space grows larger or becomes higher-dimensional, the GP begins to underfit, predicting similar outputs for distinct inputs (Figure 17). This limitation arises from sparse training data and the fact that only the

final output from each optimization run was used for GP training, discarding all intermediate samples.

To address these issues, we propose two directions: (1) modifying the GP input–output structure to incorporate more task features and (2) using all data collected during optimization, not just the final solutions. These improvements are explored in the next chapter as part of answering RQ1.3.

8 Real-Time Prediction of Policy Parameters

This chapter addresses **RQ1.3**: *Can we design a predictive model that enables real-time inference of optimal policy parameters for unseen task variations?* Building on Chapters 6 and 7, we now aim to eliminate the latency associated with re-optimization at deployment. The goal is to infer parameters on-the-fly with minimal computation and no retraining. **Paper III: Learning to Adapt the Parameters of Behavior Trees and Motion Generators (BTMGs) to Task Variations** supports our answer to this question.

8.1 Motivation and Research Framing

In Chapter 7, we showed that GP [78] models could generalize extrinsic BTMG parameters to new task variations, but only using the final optimized sample from each BO run. This discarded the intermediate samples, limiting data use and weakening performance as the variation space grew more complex.

RQ1.3 addresses this limitation by asking whether optimal parameters can be efficiently predicted for unseen task variations on-the-fly, enabling fast and robust real-world deployment. This is especially critical in flexible manufacturing or logistics, where robot tasks frequently change and must be handled reliably without retraining.

Prior work in trajectory-level control has explored similar ideas. GP-based models have been used to adapt dynamic movement primitives (DMPs) to new goals [155, 156, 157], while ProMPs [158] and MoMPs [159, 160] allow trajectory adaptation using probabilistic inference or weighted combinations of primitives. These approaches demonstrate that surrogate prediction, probabilistic inference, and library-based adaptation can generalize motion behavior effectively.

Inspired by this line of work, we extend these ideas to the symbolic skill level. Rather than directly regressing parameters for each new task variation, we predict reward and feasibility of candidate parameterizations and use lightweight optimization to select the best one. This approach forms the basis for our model, **PerF**, described next.

8.2 Approach: Surrogate Inference for Real-Time Parameter Prediction

To address RQ1.3, we propose **PerF** (Performance and Feasibility), a surrogate-based model for real-time inference of BTMG parameters. PerF uses all training data from earlier BO runs (Chapter 6), repurposing it to train a surrogate model that can be efficiently queried during deployment.

Surrogate-Based Reformulation

Given a task variation $\mathbf{v}$ and candidate BTMG parameters $\boldsymbol{\theta}_e$, PerF predicts the expected performance and feasibility of executing the policy and optimizes over this surrogate. It comprises:

- **Performance model** $\hat{J}(\boldsymbol{\theta}_e, \mathbf{v})$: A gaussian process (GP) regression model that estimates the task reward.
- **Feasibility model** $\hat{F}(\boldsymbol{\theta}_e, \mathbf{v})$: A weighted support vector machine (SVM) that classifies whether the parameters satisfy task-specific feasibility constraints.

Both models are trained using the full set of intermediate samples from BO runs. For the performance model, we collect all (task variation, parameter, reward) tuples. For the feasibility model, we evaluate each parameterization by executing the corresponding BTMG policy, determining feasibility based on task-specific criteria such as collisions or failure to reach the goal. Alternatively, feasibility information can be obtained directly from the optimization framework used during BO. In our case, we use HyperMapper 2.0 [161], which supports modeling unknown feasibility constraints through classification models as part of its surrogate-assisted search process.

GPs are chosen for performance modeling due to their ability to generalize under limited data [78]. Weighted SVMs [162] are selected for feasibility classification, as they handle class imbalance and sparse feasible regions effectively.

This formulation offers several benefits: it reuses existing data without new trials, captures reward structure more accurately than single-point models, and integrates empirical reward functions with task-specific feasibility checks.

Surrogate Optimization for Real-Time Inference

To predict parameters for a new variation $\mathbf{v}$, we define a surrogate reward:

$$\hat{r}_{\text{sur}} = \hat{J}(\boldsymbol{\theta}_e, \mathbf{v}) - (1 - \hat{F}(\boldsymbol{\theta}_e, \mathbf{v})) \cdot \mu$$

where μ is a penalty for infeasible configurations. This function is optimized using Limited-memory Broyden–Fletcher–Goldfarb–Shanno (L-BFGS) algorithm [163], a memory-efficient quasi-Newton method well-suited for problems with moderate dimensional inputs. It uses low-rank updates to approximate second-order derivatives without storing the full Hessian.

To obtain the BTMG parameters for a new variation, we solve the following equation:

$$\boldsymbol{\theta}_e^* = \arg\max_{\boldsymbol{\theta}_e} \hat{r}_{\text{sur}}$$

where the optimization is performed entirely in surrogate space, enabling real-time inference without further interaction with the robot. The model and workflow are summarized in Figure 18.

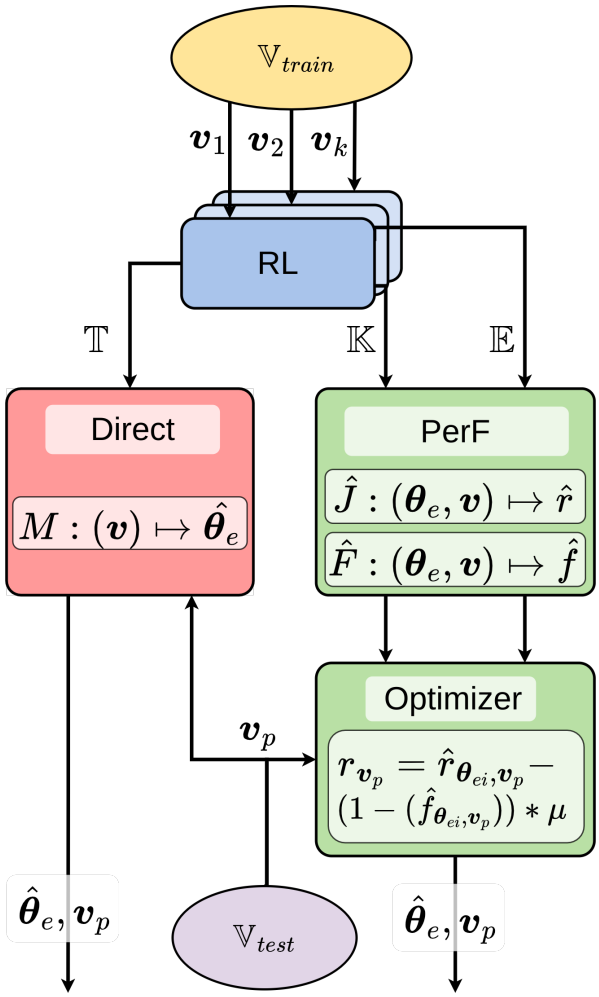


Figure 18: GP-based direct model (Chapter 7) shown in red. Right: PerF model with GP reward and weighted SVM feasibility shown in green. An optimizer infers parameters maximizing the surrogate reward.

8.3 Experimental Setup and Evaluation

We tested PerF on two tasks: push and obstacle avoidance. The push task follows the setup used previously (Chapters 6–7), where a KUKA iiwa robot pushes a triangular object using a cylindrical peg (Figure 19). Here, we additionally varied the object's initial position across test cases. For details on the obstacle avoidance task, refer to Ahmad et al. [7].

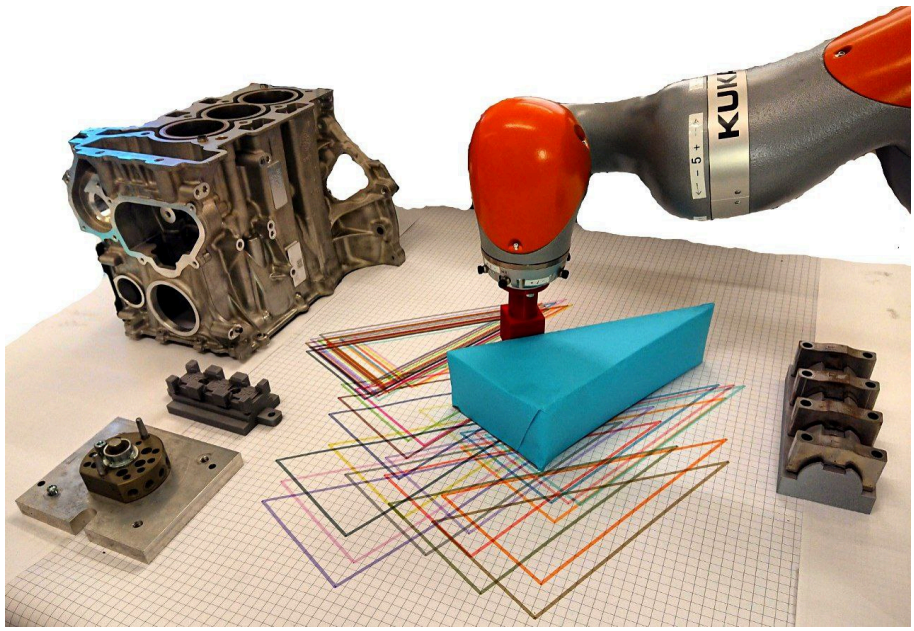


Figure 19: Experimental push setup on a KUKA iiwa. Colored boards mark start and goal poses. A triangular object is pushed with a cylindrical peg.

We evaluated PerF against four baselines:

- **Learned**: Policies retrained from scratch using RL (upper-bound performance).
- **Direct**: GP model from Chapter 7, trained using only final BO outputs.
- **Nearest Neighbor**: Applies parameters from the closest training variation based on Euclidean distance.
- **Single Policy**: Reuses all training policies across test cases to check for a universal solution.

In simulation, PerF achieved an 86% success rate, closely matching Learned (97%) and significantly outperforming Direct (65%), Nearest Neighbor (52%), and Single Policy (39%)(Fig-

ure 20). On the real robot, PerF slightly exceeded the Learned baseline, indicating better generalization and smoother physical execution (Figure 21). Inference time averaged under 6 seconds, compared to over 20 minutes for RL-based training. For further results, see Ahmad et al. [7].

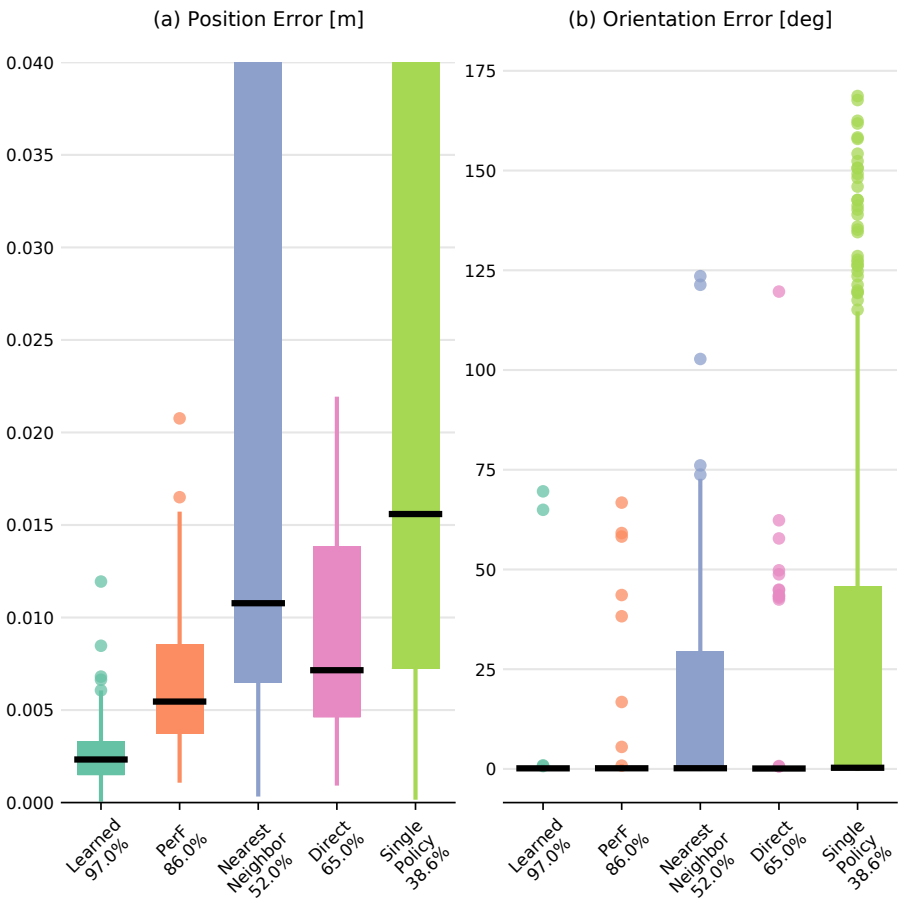


Figure 20: Simulation performance: position and orientation errors across baselines. PerF matches Learned and outperforms others.

8.4 Discussion

PerF shows that real-time parameter inference is possible without retraining by evaluating candidate parameters through surrogate modeling. By decoupling reward prediction from feasibility and training both models on full BO data, PerF enables high-quality inference with minimal compute cost. Its data reuse strategy leads to faster, more robust adaptation

than prior approaches.

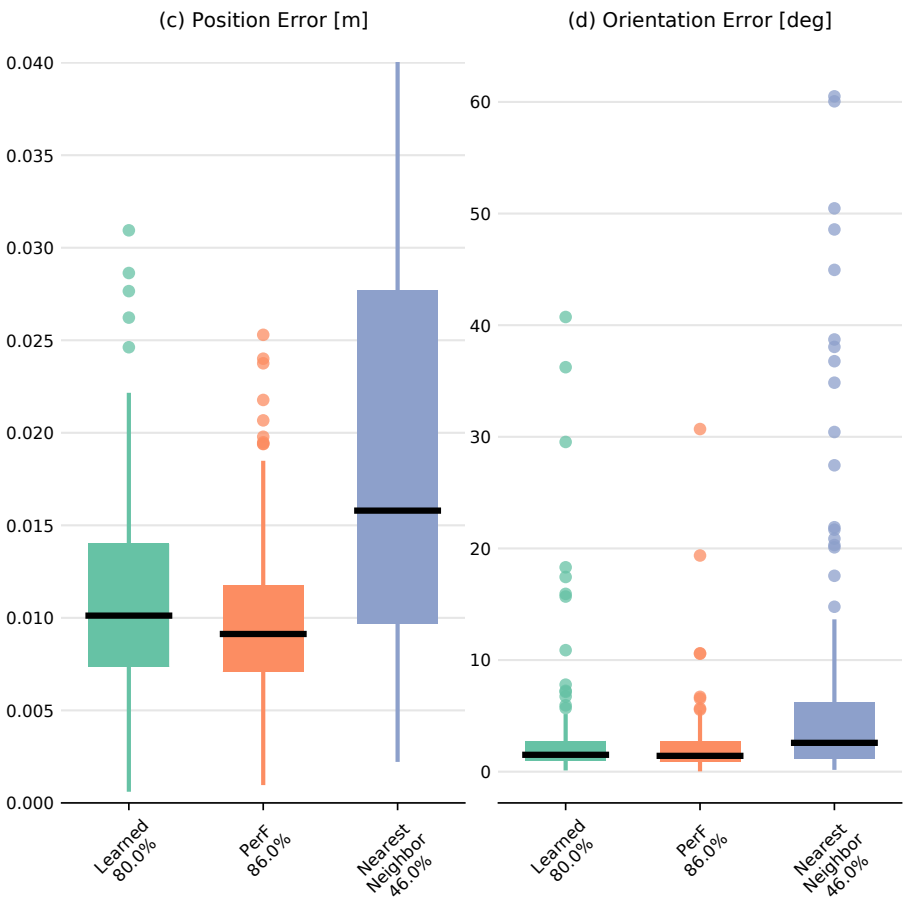


Figure 21: Real-robot results: PerF slightly outperforms retrained Learned policies.

Key insights include the importance of using intermediate learning samples to improve generalization, and the role of feasibility classification in constraining exploration within safe bounds. The use of surrogate optimization also allows principled integration of task performance and safety into a unified objective.

Nevertheless, PerF assumes a fixed BT structure and known parameter bounds. It cannot generalize across different symbolic plans or adapt to new tasks. Moreover, feasibility labels require manual definition and may not cover all failure modes. These limitations motivate future work on runtime monitoring, adaptive BT structures, and integration with failure recovery systems.

Concluding Remarks on Adaptive Skill Learning (RQ1)

This part of the thesis addressed **RQ1**: *How can we design modular, interpretable, and adaptable robot policies that generalize across task variations and enable real-time adaptation?* The investigation unfolded through three stages.

First, Chapter 6 demonstrated how robot policies can be structured using BTs and MGs, forming BTMGs that offer a clear separation between symbolic control flow and low-level motion adaptation. This modular design enabled interpretable and tunable policy representations that retained robustness across skill compositions.

Next, Chapter 7 showed that generalization across task variations can be achieved by learning mappings from task descriptors to policy parameters using GPs. This allowed BTMG policies to adapt without re-optimization, supporting reuse and flexibility.

Finally, Chapter 8 proposed PerF, a model for real-time parameter inference. By evaluating candidate policies through a surrogate composed of performance and feasibility models, PerF eliminated retraining and delivered high-quality parameters in seconds, enabling practical deployment.

Together, these contributions show that modular skill representations, data-efficient generalization, and lightweight inference methods can be combined to support adaptive robot autonomy. While structural adaptation and failure recovery remain open challenges, the methods developed here form a foundation for reliable behavior adaptation under known task variations. These results also motivate the next part of the thesis, which focuses on handling failure cases that lie beyond the training distribution.

Part II – Failure Detection, Explanation, and Recovery

9 Structuring Modular and Adaptive Recovery Behaviors

This chapter addresses **RQ2.1**: *How can recovery behaviors be designed to remain modular, interpretable, and reusable when embedded into structured robot policies?* The central challenge is to augment structured robot policies with failure handling while preserving the modularity and tunability established in Part I. Our solution, detailed in **Paper IV**: *Adaptable Recovery Behaviors in Robotics: A Behavior Trees and Motion Generators (BTMG) Approach for Failure Management*, is to represent recovery behaviors using the same BTMG abstraction as nominal skills. This enables unified learning, execution, and integration.

9.1 Motivation and Positioning within Existing Work

Industrial robotic tasks such as peg insertion, screwing, and part fitting often involve contact interactions and tight tolerances, which make them susceptible to execution failures even under nominal sensory and actuation conditions [9]. These failures are not due to hardware breakdown or sensor noise but stem from environmental uncertainties or incorrect assumptions about the task state. As categorized in Chapter 4, we focus on *execution-level failures*, which can be identified either during skill execution (e.g., peg slips mid-insertion) or beforehand, *pre-execution*, when symbolic preconditions are violated (e.g., obstacle blocking the hole before action starts). In this chapter, and specifically in Paper IV, we consider a restricted class of predefined execution failures, where failure types are anticipated and modeled offline. The recovery behavior is selected accordingly, and we begin from the assumption that the failure has already occurred. The goal is to encode both the nominal and recovery components into a single symbolic skill that is modular, interpretable, and adaptable.

Conventional failure handling in BTs relies on static fallback branches or recovery subtrees handcrafted for specific scenarios. For instance, Wu et al. [97] design fixed recovery nodes embedded in BTs, while De Luca et al. [164] construct explicit failure recovery routes in navigation trees. While modular, these designs assume known failure types and lack support for tuning behavior parameters to task conditions.

Recent research has explored hybrid methods that combine symbolic reasoning with learning. Pezzato et al. [165] integrate BTs with active inference to select recovery behaviors based on belief updates, though the symbolic and learning layers remain loosely coupled. Paxton et al. [37] introduce logical-dynamical BTs that embed robustness at the symbolic level, yet these are statically compiled and lack runtime adaptation.

At the control level, researchers have developed reactive motion strategies to increase physical robustness, such as push recovery for quadrupeds [98], regrasping in manipulation [166], or contact-force corrections [167]. These techniques enhance motion-level durability but do not support symbolic planning or recovery reuse across different tasks.

In contrast, our approach encodes recovery behaviors within the same BTMG structure used for nominal task skills [13, 9]. This unified representation enables symbolic planning, modular reuse, and parameter learning. A symbolic PDDL planner is used to generate recovery-augmented plans by composing production and recovery skills for each known failure. Importantly, the approach adopts a selective tuning strategy, only the parameters associated with the recovery action are learned (e.g., push force or grasp offset), leading to adaptive yet data-efficient behavior synthesis. The remainder of this chapter details the structure and evaluation of this framework.

9.2 Modeling Recovery in the BTMG Framework

Our approach to RQ2.1 models both production and recovery behaviors using the BTMG framework. Production skills achieve task objectives (e.g., peg insertion), while recovery behaviors address known failure types (e.g., obstacle removal or regrasping). Both are represented using the same BTMG structure and parameterized by extrinsic values such as force magnitude, pose offsets, or compliance gains.

We operate under two core assumptions: (i) failure types are known and expected based on prior task knowledge, and (ii) recovery behaviors can be constructed by composing and tuning existing skills from the BTMG library. These assumptions enable the use of a fixed BT structure combined with parameter learning of the MG part.

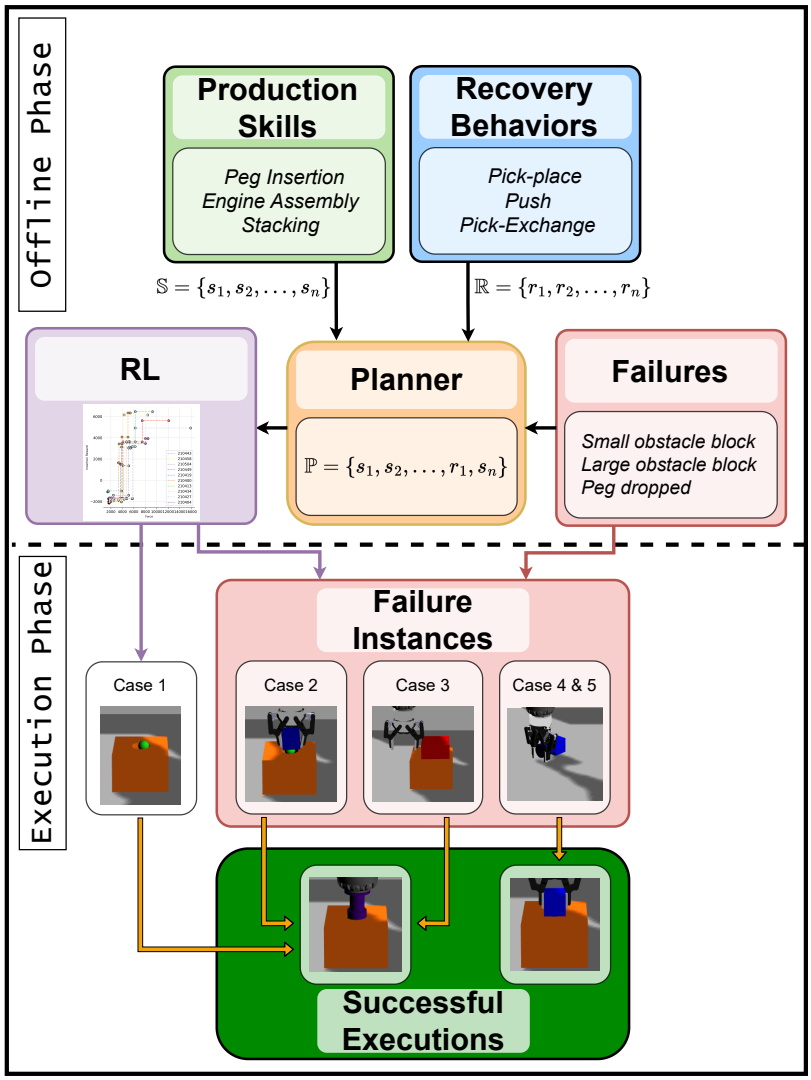


Figure 22: Illustration of our approach: for a known failure, a symbolic planner composes a plan using production and recovery skills which are represented in the BTMG format. BO is used to tune policy parameters, after which the policy is deployed.

The method proceeds in two phases. In the **offline phase**, for a given failure scenario, a symbolic PDDL planner generates a plan (BTMG-based policy) by composing production and recovery skills. This policy is passed to a learning module, where tunable parameters are optimized using the multi-objective bayesian optimization techniques discussed in Chapter 6.

In the **execution phase**, the optimized policy is executed without further modification.

This design supports reuse of the same planning and learning framework for both nominal task execution and recovery handling. Rather than relying on fixed fallback structures, the system synthesizes symbolic recovery plans at design time and adapts them to specific failure conditions using previously tuned parameters.

9.3 Failure Cases and Recovery Behavior Design

We evaluate our approach using the peg-in-hole task introduced in Chapter 6, where the goal is to insert a peg into a hole. This task serves as a benchmark for assessing how recovery behaviors can be modularly designed and integrated using the BTMG framework. In line with the assumptions of Paper IV, we consider a set of *known and predefined* execution failures that were modeled in advance.

We consider the following failure instances:

- **F1 – Small obstacle blocking the hole** (**Figure 23b**): A small object blocks the hole, preventing peg insertion.
- **F2 – Large obstacle blocking the hole** (**Figure 23c**): The hole is obstructed by a large object beyond the grasping capability of the gripper.
- **F3 – Peg dropped from gripper** (**Figure 23d**): The peg slips from the gripper before insertion is completed.

Each failure is addressed by a corresponding recovery behavior, represented as a BTMG policy composed of skill primitives:

- `Pick-Place` (used for F1): Moves the gripper to the obstacle, grasps it, lifts it, and places it aside. This behavior is manually defined and contains **no learnable parameters**.
- `Push` (used for F2): Applies a lateral force to move the obstructing object. It includes **one learnable parameter**: the push force magnitude.
- `Pick-Exchange` (used for F3): Regrasps the dropped peg and restores it to the insertion pose. It involves **two learnable parameters**: the x- and y-offsets of the grasp location, which may be fixed or learned depending on the scenario.

All recovery behaviors are constructed from a shared set of reusable skill primitives:

- `GripperOpen`: opens the gripper
- `GripperClose`: closes the gripper to secure an object
- `GoToLinear`: performs linear motion to a specified pose
- `ChangeStiffness`: adjusts impedance control gains for compliant motion
- `ApplyForce`: applies a Cartesian force vector at the end-effector

These primitives are composed into BTMG representation to define recovery behaviors. For example, `Pick-Place` sequences a linear approach, grasp, lift, and release; `Push` combines compliance adjustment with a lateral force application; and `Pick-Exchange` includes grasp offset correction followed by re-alignment to the insertion pose.

9.4 Execution Scenarios and Learning Setup

To evaluate how well the proposed framework supports modular and adaptive recovery behaviors, we define five representative scenarios. These scenarios vary in both the type of execution failure and whether the associated recovery behavior includes learnable parameters. Across all scenarios, the insertion skill is optimized using the same four extrinsic parameters introduced in Chapter 6: *downward insertion force, spiral path velocity, spiral radius, and path distance.*

We evaluate the following five scenarios:

1. **Baseline:** No failures or recovery behaviors are present. Only the insertion skill is executed and optimized. This serves as a reference for normal task execution (Figure 23a).
2. **Static Recovery (F1):** Failure F1 (small obstacle blocking the hole) is introduced and resolved using a fixed `Pick-Place` behavior with no learnable parameters. Only the insertion skill is optimized (Figure 23b).
3. **Dynamic Recovery (F2):** Failure F2 (large obstacle blocking the hole) is resolved using the `Push` behavior, which includes a tunable push force. Insertion parameters and push force are optimized jointly (total of five parameters) (Figure 23c).
4. **Static Recovery + Behavior Change (F3):** Failure F3 (peg dropped from gripper) is resolved using a manually defined `Pick-Exchange` behavior with fixed grasp offsets. Only the insertion skill is optimized (Figure 23d).

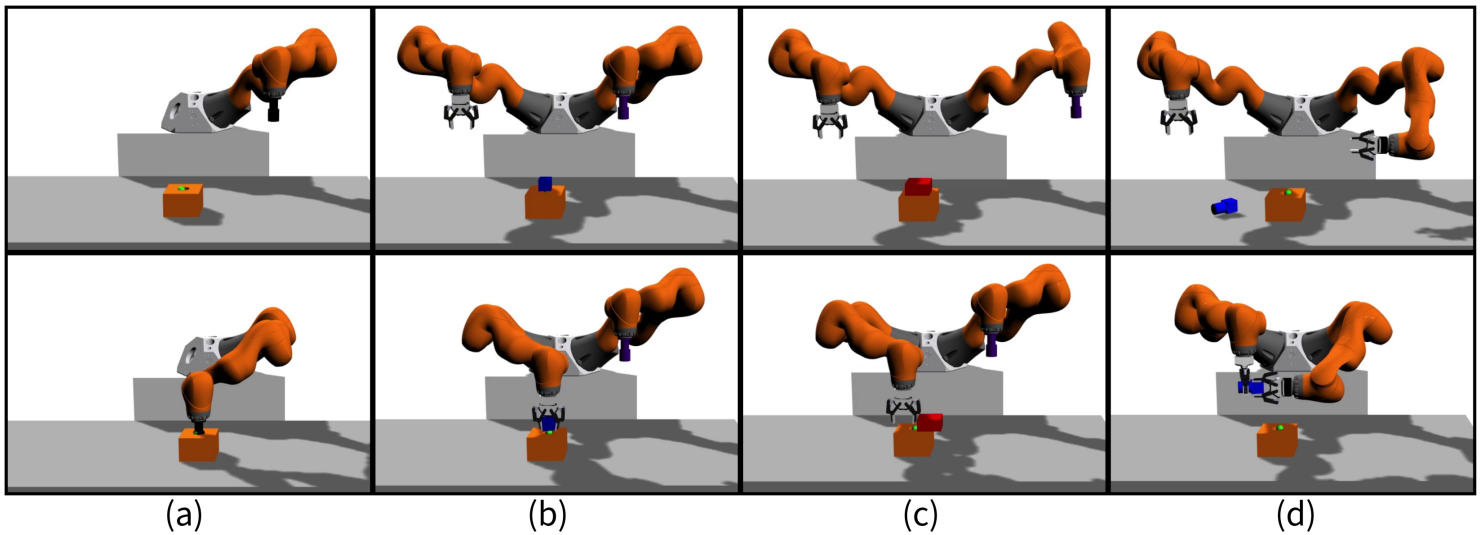


Figure 23: Peg insertion task and corresponding recovery behaviors. (a) Nominal execution. (b) F1: chip in hole removed by `Pick-Place`. (c) F2: access blocked, resolved by `Push`. (d) F3: dropped peg recovered by `Pick-Exchange`.

5. **Dynamic Recovery + Behavior Change (F3):** Uses the same failure as Scenario 4 but includes learning of grasp x- and y-offsets in addition to the insertion parameters (total of six parameters) (Figure 23d).

For each scenario, we perform multi-objective Bayesian Optimization over 40 iterations with 10 random seeds. Candidate policies are evaluated in five randomized simulation environments that vary the block position and initial peg pose. The reward function balances two competing objectives: insertion success and contact force minimization.

All five scenarios yielded at least one successful policy capable of achieving 5/5 successful insertions across random world instances. This validates the framework's ability to accommodate fixed and learnable recovery behaviors while preserving task performance.

9.5 Discussion

The experiments confirm that recovery behaviors modeled as symbolic BTMG skills can be effectively integrated into structured robot policies without degrading task performance. Across all five scenarios, including both static and dynamic recovery, the framework produced successful policies that handled different failure modes under randomized conditions. This supports the claim in RQ2.1 that modeling recovery and production behaviors within the same BTMG structure enables modular reuse, parameter adaptation, and unified planning. The results also highlight the value of selective learning: in cases where parameter tuning was necessary (e.g., push force, grasp offsets), the optimizer efficiently adjusted recovery parameters without affecting unrelated parts of the policy.

This work focuses strictly on expected and predefined execution failures. All failure types are known ahead of time, and recovery behaviors are composed from existing skills that are modeled offline. The framework assumes that failure detection occurs before execution, and that symbolic planning (via PDDL) can be performed before runtime. As such, unanticipated failures or mid-execution recovery are outside the scope of this study.

These limitations are addressed in the next chapter (Chapter 10), which investigates **RQ2.2**. There, we extend the framework to handle pre-execution failures, such as unmet symbolic preconditions, by enabling failure-aware planning before execution begins. A reactive planner is introduced that can dynamically modify or extend the behavior tree based on detected conditions at planning time. Recovery from runtime failures, including unanticipated disturbances during execution, is tackled in Chapter 11, where the system adapts online through monitoring and BT modification. Together, these extensions move toward policies that can adapt both before and during execution, improving robustness in dynamic settings.

10 Pre-Execution Failure Handling with Vision–Language Models

This chapter addresses **RQ2.2**: *Can a robot detect and explain failures before executing a skill, enabling preemptive intervention using environmental feedback?* We focus on failures that arise not during execution, but beforehand, when symbolic preconditions are violated or task assumptions do not hold. The challenge is to detect such failures automatically and adapt the robot's policy before acting. Our approach, developed in **Paper V: Addressing Failures in Robotics using Vision-Based Language Models (VLMs) and Behavior Trees (BT)**, integrates VLMs into the symbolic planning loop. VLMs provide a flexible interface to inspect the current environment and predict task feasibility. If a problem is identified, the system adapts its policy by modifying or extending the current BT before execution begins.

10.1 Motivation and Positioning within Existing Work

In symbolic robot planning, task failures are often discovered only during execution, when it is already too late. Studies have shown that classical planners assume perfect action execution and only detect errors after execution fails, requiring mid-course replanning [168]. Similarly, task & motion planning systems rely on predefined symbolic models and only detect precondition violations when grounding fails during execution [169]. More recent methods have begun augmenting planning pipelines with LLM-based plan verification to anticipate such issues before they occur [170].

Prior work focuses primarily on reacting to execution failures. For instance, Wu et al. [97] designed BT based recovery strategies for known errors, while Ahmad et al. [9] compose symbolic recovery skills for anticipated execution-level failures. These methods require predefined failures and do not generalize to novel scenarios. Some systems use VLMs to generate recovery actions post-facto, such as Chen et al. [137] and AHA [93], but these approaches handle symbolic correction only indirectly and after plan execution has begun. ReplanVLM [100] rebuilds the plan entirely upon failure, maintaining symbolic structure but relying on re-planning rather than incremental correction.

A key limitation across these methods is the assumption of perfect symbolic knowledge and the lack of proactive plan inspection before execution. To address this gap, our method (Paper V) injects a VLM before execution. The VLM reviews the current scene, symbolic state, and BT to predict feasibility. If a symbolic precondition is violated, it identifies the missing condition or suggests a new skill.

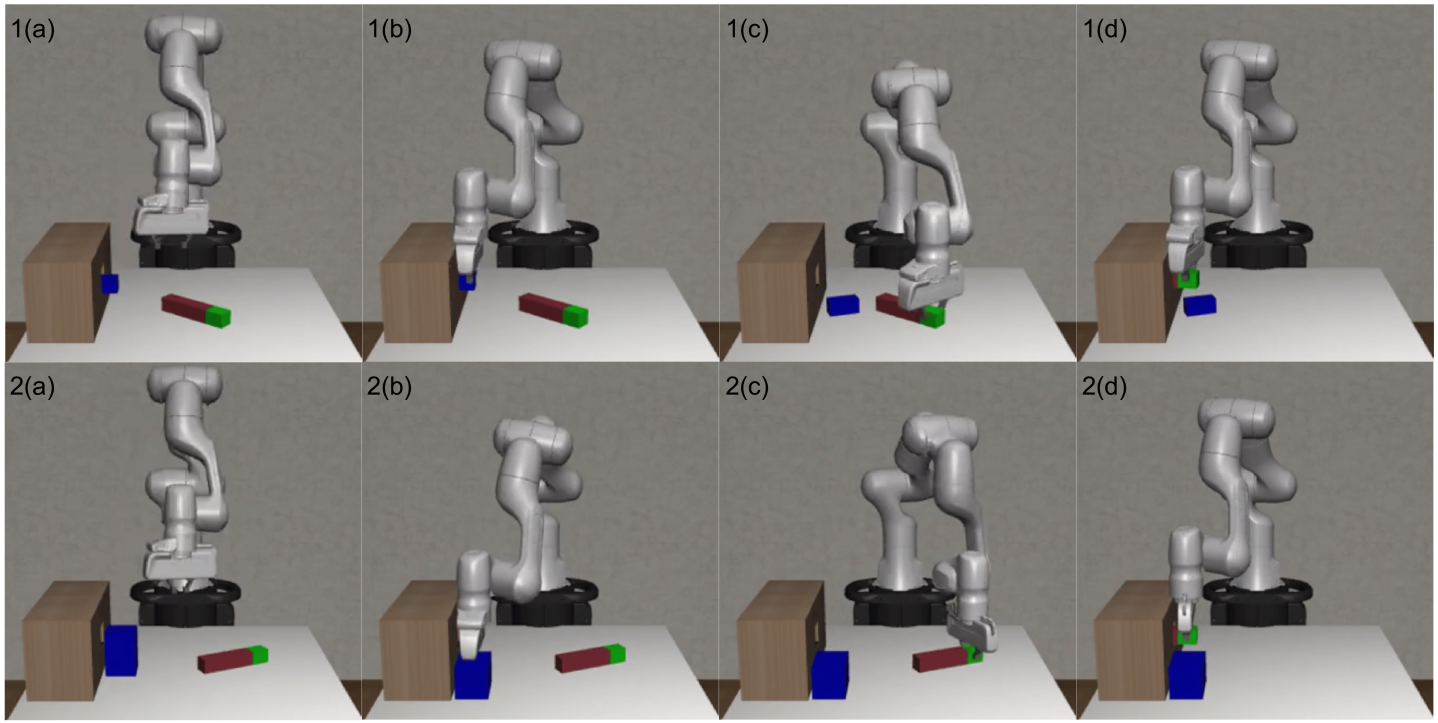


Figure 24: System overview: the VLM inspects the current environment and symbolic BT, identifies missing preconditions or skills, and guides the reactive planner to generate a corrected BT before execution.

A reactive planner then incrementally extends or patches the BT, unlike PDDL replanning, this reactive planner retains existing plan structure and enables targeted BT extension [107, 38]. This results in truly proactive failure handling: the robot adapts its symbolic policy using multi-modal feedback, before execution begins.

This chapter evaluates whether pre-execution symbolic reasoning is feasible and beneficial. Drawing on Paper V, we explore how VLMs can detect potential failures early, articulate them in symbolic terms, and enable corrective BTs that are both interpretable and reusable.

10.2 Approach Overview

To address **RQ2.2**, we propose a framework that augments symbolic robot policies with visual and language-based reasoning *before* execution begins. The goal is to detect potential failures, such as violated symbolic preconditions or the absence of a needed skill, and proactively correct the BT to ensure feasibility. Rather than relying on post-failure recovery, the robot evaluates whether the current plan is valid in its actual environment and, if not, amends it accordingly. This is achieved by integrating a VLM with a reactive planner and symbolic BTs. The VLM acts as a reasoning module that interprets both visual observations and symbolic structure, while the reactive planner enables dynamic modification of BTs based on preconditions.

Our approach decomposes pre-execution failure handling into three abstract reasoning stages:

- **Failure detection**: Is the current BT likely to fail in this environment? (binary output from the VLM)
- **Failure identification**: Which skill is expected to fail, and which precondition is violated?
- **Failure correction**: How can the symbolic plan be augmented to restore feasibility?

This chain-of-thought [171] style reasoning makes the internal logic of the system transparent and auditable. The different steps of th approach are shown in Figure 25.

To perform this reasoning, the VLM is prompted with four types of input:

- **RGB images** from multiple viewpoints, which provide visual context and help resolve occlusions in cluttered scenes.

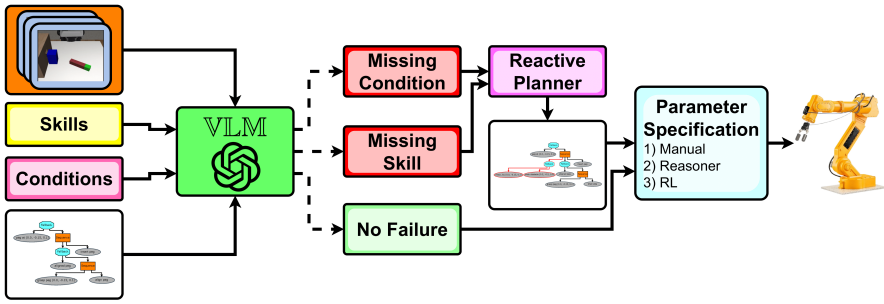


Figure 25: Overview of the proposed VLM-BT integration: the VLM receives multi-view RGB images, symbolic BTs, skill descriptions, and known conditions. It predicts missing conditions or skill templates to correct pre-execution failures. The reactive planner then synthesizes an updated BT and lastly the parameters are tuned either manually, via a reasoner or using RL before execution on the robot.

- **The current BT**, represented in plain text, which describes the robot's intended sequence of actions.
- **The skill library**, containing symbolic skills with natural language descriptions, which defines available actions and their symbolic semantics.
- **The known symbolic conditions**, which encode the system's current knowledge about the world state.

Given this prompt, the VLM assesses the feasibility of the BT. If a failure is likely, it returns either:

- A `missing_condition`, e.g., `hole free`, representing a symbolic requirement that is not currently satisfied.
- A `suggested_skill`, such as a `Push` action, encoded as a skill template with a symbolic name, parameters, and postconditions, when the `missing_condition` alone is insufficient to resolve the failure.

The use of `missing_condition` as the primary output is motivated by how the reactive planner operates. Once a missing precondition is added to a skill node in the BT, the reactive planner automatically expands the tree by backchaining through the skill library to find a skill (or sequence of skills) whose postconditions satisfy it. This makes the correction process modular, symbolic, and grounded in the planner's existing action model. For example, appending the condition `hole free` to an insertion skill causes the planner to insert a `Pick-Place` subtree if it has matching postconditions or suggest a new `Push` skill if the obstacle is larger than the gripper affordance.

If no known skill satisfies the missing condition, the VLM proposes a new skill template. A human expert then reviews and completes this template. Once added to the library, the skill becomes available for future expansions. (In such cases, its symbolic cost is adjusted to prioritize it appropriately over alternatives that produce the same effect.)

Once the BT structure is finalized, skill parameters must still be specified. These include continuous values such as force, pose offsets, or velocities. Depending on the task, these parameters can be selected manually, inferred through symbolic reasoning, or optimized using learning techniques such as bayesian optimization, as introduced in Chapter 6.

Throughout this process, the VLM never directly controls execution. Instead, it serves as a pre-execution filter that inspects planned actions against the current environment and suggests symbolic corrections. The reactive planner then synthesizes a new plan that satisfies the goal under current conditions. This combination enables the system to proactively detect and resolve pre-execution failures in a way that is general, interpretable, and reusable.

10.3 Illustrative Scenarios

We evaluate the proposed framework using `robosuite` [**robosuite2023**] simulations and GPT-4o as the VLM. Each scenario begins with a nominal BT that would succeed in an ideal environment. An unforeseen precondition violation is then introduced, and we assess whether the VLM and reactive planner can detect and correct the plan before execution. Execution stages for the first two scenarios (small and large obstacle) are illustrated in Figure 24, where each row shows the visual progression from failure detection to task success. The corresponding symbolic plan corrections, before and after VLM-guided updates, are shown in Figures 26 and 27.

Peg-in-hole with small obstacle: A small obstacle blocks the hole. The VLM identifies this as a potential failure and suggests the symbolic precondition `hole free` (internally represented as `check_loc`). The reactive planner responds by appending a recovery subtree composed of `Grasp`, `Reach`, and `Open` actions, forming a `Pick-Place` behavior that clears the hole. This example demonstrates successful symbolic correction through known skill reuse. The original BT is shown is Figure 1.

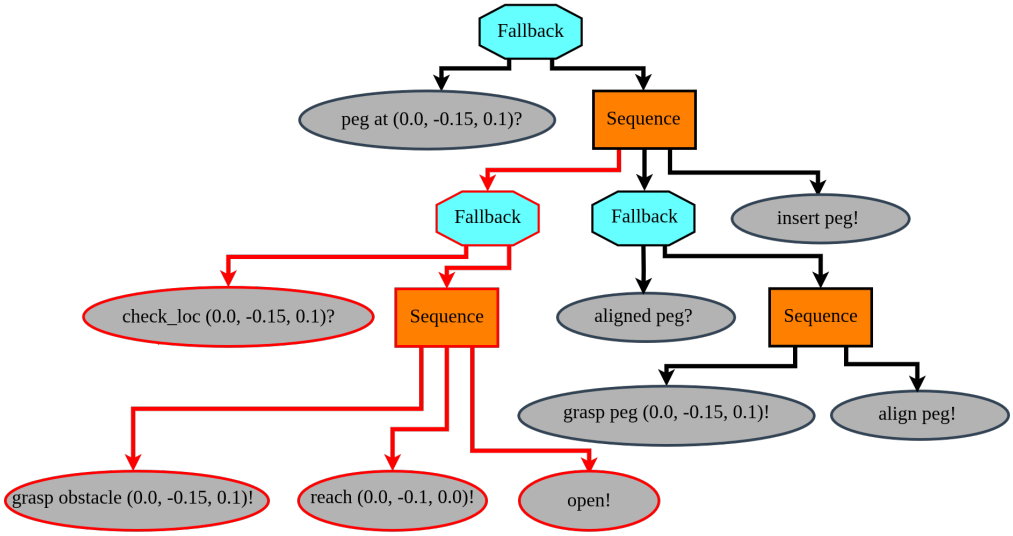


Figure 26: BT correction for small obstacle scenario: the VLM proposes a missing condition, which the reactive planner satisfies by inserting a known `Pick-Place` subtree.

Peg-in-hole with large obstacle: A large object blocks the hole and cannot be grasped. The VLM detects this condition and determines that no existing skill can satisfy the missing precondition. It then suggests a new `Push` skill template, which is later completed by a human expert and added to the skill library. In subsequent runs, this new skill is automatically selected by the planner when the same condition arises. The original BT is shown is Figure 1.

Lift task: An obstacle rests on the target object. The VLM suggests the symbolic condition `target clear` as a missing precondition for the grasping skill. The planner resolves this by inserting a known `Pick-Place` subtree that clears the obstacle from the grasp target.

Door opening: The robot attempts to pull a door without turning the handle. The VLM proposes `handle turned` as a missing precondition, which is satisfied by inserting the known `TurnHandle` skill as a prerequisite step.

For detailed BT structures and step-by-step execution sequences corresponding to each scenario, please refer to the supplementary material provided in Paper V.

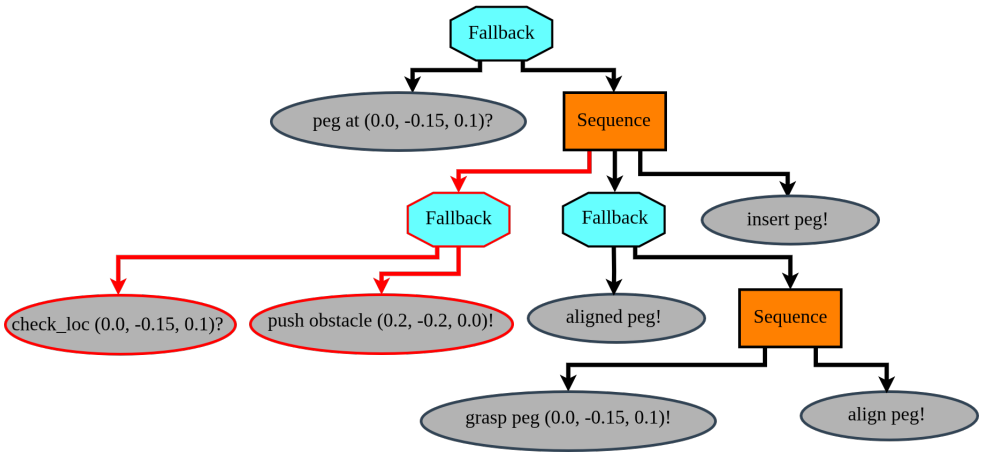


Figure 27: BT correction for large obstacle scenario: the VLM detects a precondition violation and proposes a new `Push` skill, which is later reused in future plan expansions.

10.4 Evaluation and Findings

We conducted 10 trials per scenario using GPT-4o as the Vision–Language Model. To ensure reproducibility and stable outputs, we used a deterministic decoding configuration with `temperature = 0.1` and `top_p = 0.1`. This setting reduces the randomness in generation, helping the model consistently return valid symbolic conditions or skill suggestions.

- **Consistent failure detection:** The VLM successfully detected and corrected symbolic failures in all tasks across all trials, achieving 100% consistency.
- **Role of vision:** Removing image input severely degraded performance. In the peg-in-hole with large obstacle scenario, success dropped from 100% with vision to 30% using only text-based prompts. Even when feasibility constraints were manually included in the prompt, success reached only 60%, with inconsistent and unreliable outputs. This reinforces that visual grounding is essential for physically meaningful reasoning, LLMs alone cannot substitute for perception in geometry sensitive tasks.
- **Skill feasibility:** All VLM-generated conditions and skill suggestions were executable and matched the physical constraints of the robot and scene. In our setup, skill feasibility refers to whether a suggested action aligns with the robot's capabilities (e.g., grasping only if the object is within gripper size limits or not occluded). This ensures that the recovery suggestions are practical and do not introduce additional failure risk.

These results indicate that VLM-based pre-execution reasoning can scalably extend symbolic policies while maintaining consistency and physical realism. The ability to correct plans before execution, and ensure that proposed corrections are executable, makes this approach promising for reducing failure rates in real-world robot deployments.

10.5 Discussion

This work demonstrates that VLMs can be effectively integrated into symbolic planning pipelines to anticipate failures before execution. By inspecting both the visual context and the symbolic policy, the VLM enables proactive correction of missing preconditions or capabilities. This allows the robot to avoid futile actions and instead begin execution from a policy that has already been adapted to its current environment.

One notable insight is the VLM's capacity to reason about physical feasibility, for instance, recognizing that an obstacle is too large for the gripper to grasp and instead proposing a new action like pushing. This suggests that the VLM is not only grounded in visual and symbolic input but can align its suggestions with task and hardware constraints, enabling physically valid plan corrections.

Once a correction is proposed and accepted, it becomes reusable: the updated BT and skill library entries persist, allowing future plans to inherit robustness without repeating the correction step. Over time, this allows the system to incrementally build resilience to disturbances without relying on hardcoded recovery branches or exhaustive planning.

However, there are still limitations. The system depends on the VLM's ability to generalize from minimal demonstrations or examples. While the reactive planner ensures symbolic consistency, the feasibility of entirely new skills still requires human validation. This means that the approach, while reducing manual effort, is not yet fully autonomous. Moreover, pre-execution reasoning assumes the failure is detectable from the visual and symbolic prompt; failures that emerge only during execution (e.g., slipping, deformation) remain outside the current scope.

In the broader context of **RQ2.2**, this chapter shows that symbolic failure detection before execution is both feasible and beneficial when guided by a vision–language model. The integration of learned multimodal reasoning with reactive symbolic planning offers a scalable path toward adaptable, interpretable robot behavior. The next chapter (Chapter 11) builds on this foundation by addressing **RQ2.3**, extending the framework to handle runtime failures through monitoring and dynamic BT modification.

11 Real-Time Monitoring and Reactive Recovery

This chapter addresses **RQ2.3**: *How can a robot detect, explain, and correct failures* during *execution, not just beforehand, while remaining modular and interpretable?*

Whereas Chapter 10 focused on detecting failures before execution begins, many realistic failures only manifest at runtime due to dynamic disturbances, occlusions, or human interference. In such cases, pre-execution inspection is insufficient. RQ2.3 therefore asks whether robots can handle such failures in a modular and interpretable manner as they occur, rather than relying solely on proactive planning or full replanning.

Our approach, developed in **Paper VI: A Unified Framework for Real-Time Failure Handling in Robotics Using Vision–Language Models, Reactive Planner and Behavior Trees**, builds on the pre-execution verification framework introduced in Chapter 10, and extends it with a real-time monitoring loop. The robot checks symbolic preconditions and postconditions during execution, invokes a VLM to reason about missing knowledge, and uses a reactive planner to dynamically extend the running BT. The result is a unified, two-phase framework for failure handling that supports both proactive and reactive intervention, using shared symbolic and visual representations.

11.1 Motivation and Positioning within Existing Work

Robots deployed in unstructured environments, from assistive homes to cluttered factories, must contend with dynamic and partially observable surroundings. Failures can emerge not just from initial planning errors, but from unexpected runtime events such as occlusions, object displacements, or human interference. As described in Chapter 4, proactive failure recovery attempts to prevent such issues by inspecting the environment and symbolic policy before execution. However, this strategy cannot catch failures that depend on action outcomes or newly revealed conditions. Reactive failure recovery instead responds during execution, typically by verifying symbolic conditions and inserting repairs as needed.

Pre-execution verification, discussed in Chapter 10, is effective in catching incomplete plans or known symbolic violations. Yet, consider a drawer placement task: the robot may correctly plan to open a drawer and place an object, only to discover at runtime that the drawer already contains an item. This postcondition violation (`occupied(drawer)`) is invisible until the drawer is opened. The robot must reactively insert a subgoal to clear the drawer, a correction that was not known or relevant at planning time.

Prior work has explored related ideas across different failure handling modes. Reactive symbolic recovery methods such as Wu et al. [97] use BTs with hand-coded fallback branches for each known failure type. Similarly, RACER [102] integrates recovery strategies into

imitation learning pipelines. While interpretable, both approaches require predefining the types of failures and do not generalize to unseen ones. RECOVER [88] introduces symbolic monitors to detect failures and supports plan repair, but relies heavily on fixed relational graphs and pre-specified symbolic descriptions, limiting applicability in changing environments.

Large language and vision–language models have also been explored for failure reasoning. REFLECT [91] summarises past experience and queries an LLM for failure explanations, but only after execution. AHA [93] and DoReMi [92] detect and explain failures using VLMs at selected checkpoints, but do not operate over structured symbolic policies. Similarly, Code-as-Monitor [95] monitors execution against pseudo-code representations, but lacks symbolic correction capabilities. While these methods support failure understanding, they do not permit real-time symbolic repair within ongoing plans.

Other works improve real-time perception and affordance grounding. Chen et al. [137] optimize prompts to detect spatial violations using VLMs, but treat the model as a standalone policy selector rather than part of a symbolic system. Causal-based reasoning [105] and CurricuLLM [106] use LLMs to learn causally valid curricula or identify action dependencies, but require posthoc plan re-generation or retraining, rather than direct symbolic correction during execution.

In contrast to these works, our approach in Paper VI offers three distinctive contributions. First, it performs real-time symbolic monitoring by continuously verifying preconditions and postconditions at every skill tick and detecting violations through a combination of visual and symbolic context. Second, it uses a reactive planner to insert missing preconditions or new skills directly into the currently running BT, avoiding full replanning and preserving the interpretability of the symbolic structure. Third, it employs a unified reasoning pipeline for both pre-execution and real-time recovery, built around a shared VLM, incremental scene graph, and execution log. This enables consistent, modular failure detection and correction throughout the entire task lifecycle.

The next subsection describes the architecture and key components of this unified framework in detail.

11.2 Approach Overview

To address **RQ2.3**, we extend the pre-execution failure detection framework introduced in Chapter 10 by enabling robots to also detect and correct failures *during* execution. While pre-execution checks can catch infeasible plans before any movement begins, many failures arise only at runtime due to occlusions, sensor drift, or environmental changes. Our approach combines a BT execution policy, a reactive planner, and a VLM into a unified

two-phase loop for symbolic failure detection and repair.

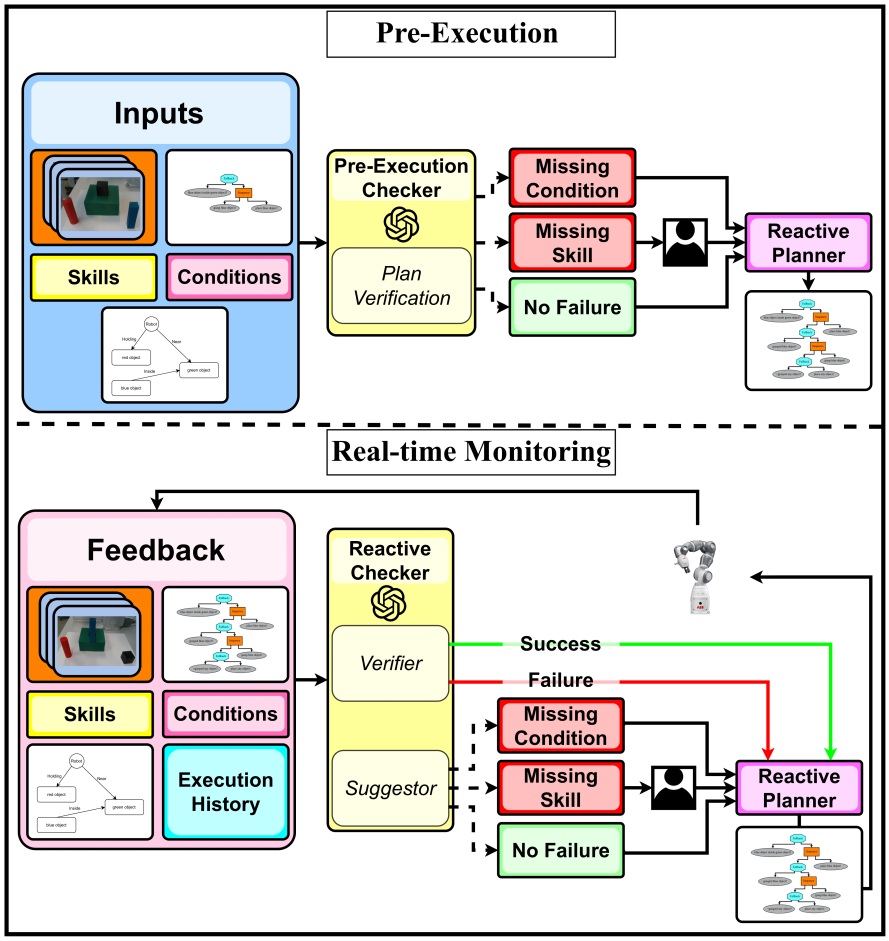


Figure 28: Overview of the two-phase system. The *Pre-execution phase* verifies the planned BT before execution by checking for missing conditions or skills to address the failure. The *Real-time monitoring phase* uses Verifier and the Suggestor module to monitor pre and post conditions, execution, detect failures, and propose corrections.

The framework consists of two distinct but integrated phases:

(1) **Pre-execution verification**, performed once immediately after planning, uses the same method introduced in Chapter 10 (Paper V). The VLM analyzes the generated BT and the current environment to suggest either a missing symbolic precondition or a new skill template. The reactive planner integrates this correction by expanding the BT accordingly. This helps anticipate and fix symbolic gaps before execution starts.

(2) **Real-time monitoring**, applied at every tick of BT execution, performs continuous failure detection and correction. Two modules operate in this phase:

- The *Verifier* checks that skill preconditions are satisfied before execution and verifies postconditions afterwards. Failures are handled reactively by the BT structure (e.g., retrying or reselecting).
- The *Suggestor* is invoked when a failure is detected. It reuses the same prompting strategy as the pre-execution phase but is applied during runtime. It returns missing conditions or new skill templates based on current multimodal context, enabling incremental repairs.

The key idea is to preserve the modular, interpretable structure of symbolic BTs while augmenting them with real-time visual reasoning and language-based correction. We reuse the same chain-of-thought reasoning process as in Chapter 10, *detection*, *identification*, and *correction*, but now apply it both before and during execution.

Figure 28 shows the overall system, including both monitoring phases and shared inputs.

Both phases rely on a common set of structured inputs:

- **RGB-D images** (front and side views) provide real-time visual context and help resolve occlusions in cluttered scenes.
- **Skill library** contains the set of symbolic skills available to the robot, each annotated with pre- and post-conditions.
- **Conditions** represent a fixed set of boolean predicates. For instance, `on(obj1,obj2)` and `grasped(obj)` define the symbolic world state.
- **Scene graph** encodes symbolic object–object and robot–object relations using the same condition predicates. This structure complements image input by making world knowledge accessible to symbolic reasoning and the VLM.
- **Behavior tree** represents the current execution policy. It is dynamically updated by the reactive planner when symbolic fixes are needed.
- **Execution history**, used only during runtime, logs previous skill ticks, their outcomes, and changes in the scene graph. This is particularly useful when failures depend on subtle visual shifts that happen during skill execution but are only detected afterward. For example, in a stacking task, if a human removes an object from the table while the robot is approaching it, the history allows the system to recognize the failure retrospectively even though it was not observed during skill execution.

The two-phase architecture ensures that known symbolic gaps are caught early (pre-execution), while runtime failures, those that emerge through interaction, are detected and resolved on-the-fly. The VLM never controls the robot directly; it only suggests symbolic additions based on multimodal inputs. The reactive planner handles policy expansion, ensuring that all corrections remain interpretable, grounded, and consistent with the symbolic model.

The next subsection introduces the visual perception pipeline that enables real-time scene understanding and maintains the scene graph required for symbolic monitoring.

11.3 Visual Pipeline and Scene Graph Maintenance

Reliable real-time monitoring depends on accurate, low-latency perception of the robot's environment. To support symbolic reasoning over object relations and skill outcomes, we maintain a continuously updated scene graph using a structured visual pipeline that operates at 15 Hz. This scene graph encodes object–object and robot–object relationships such as `on`, `inside`, or `holding`, which are used by the verifier and suggestor during both pre-execution and runtime monitoring.

The pipeline combines object detection, segmentation, pose estimation, and relation inference. Object detection is performed using Grounding DINO [172], which produces 2D bounding boxes conditioned on language prompts. These detections are passed to SAM2 [173] for instance segmentation and tracking, enabling mask-level precision and identity maintenance over time. Each segmented region is projected into a point cloud using RGB-D depth information. We estimate 6-DoF object poses by aligning these point clouds to pre-scanned object models using RANSAC for initial alignment and PCA for pose refinement.

Given the object poses and geometries, spatial relations are computed through simple geometric tests. For instance, `on(A, B)` is inferred if the bottom of object A lies slightly above the top surface of B within a fixed distance threshold. Similarly, `inside(A, B)` requires object A's bounding volume to be fully enclosed within B's cavity.

Unlike prior work such as REFLECT [91], which reconstructs a new scene graph from scratch after each action, we incrementally update the scene graph in-place. Each frame triggers localized edits: new objects are added, vanished objects are removed, and relations are re-evaluated only for changed elements. This improves runtime efficiency and consistency, enabling the verifier to correlate current observations with symbolic expectations.

11.4 Demonstrated Scenarios

We validate our unified failure handling framework across both simulated and physical robotic setups, demonstrating its effectiveness in real-time failure detection, identification, and correction. Unlike prior work such as REFLECT [91], which generates hierarchical summaries post-execution for language-guided replanning, our system continuously monitors execution and applies symbolic corrections on-the-fly. This allows us to address both pre-execution and unforeseen runtime failures, ensuring robust performance across varying environments.

Simulation (AI2-THOR): We reproduce all failure scenarios in the REFLECT benchmark [91] using our framework. Unlike REFLECT, which reconstructs the scene graph and performs reasoning only after execution completes, our system maintains an incrementally updated scene graph and performs real-time plan repair. Across all 50 randomly seeded trials, our system achieved 100% task success without requiring post-execution intervention. Failures such as blocked receptacles, incorrect object states, or occlusions were detected and corrected during execution via the Verifier and Suggestor modules (see Section 11.2). These results show that structured symbolic reasoning, combined with proactive VLM checks and reactive BT rewrites, can fully eliminate the need for retrospective correction.

Physical robot (ABB YuMi): We deploy our system on an ABB YuMi robot equipped with an RGB-D camera, testing its performance across three tabletop manipulation tasks, each under dynamic disturbances:

- **Peg-in-hole:** Execution perturbations include occlusions of the hole and dropped pegs, leading to symbolic violations such as `hole free` or `grasped(object)`.
- **Drawer opening:** Failures are introduced through drawer jamming or human interference, requiring skill reordering or insertion of precondition restoring actions.
- **Object sorting:** External bin movements and object displacement test the system's ability to detect spatial inconsistencies like incorrect `inside(obj, bin)` relations.

We conducted 30 real-world trials (10 per task) with perturbations at varying stages. Our full framework (pre-execution + reactive monitoring) achieved 100% success. Using only reactive monitoring also succeeded but required 30% longer execution due to repeated skill reattempts. Pre-execution-only verification failed in 69% of trials, unable to anticipate runtime disturbances. These results show that unified failure handling, combining proactive and reactive reasoning, is essential for robust execution in dynamic settings. Demonstrations on the real robot are shown in Figures 29 and 30.

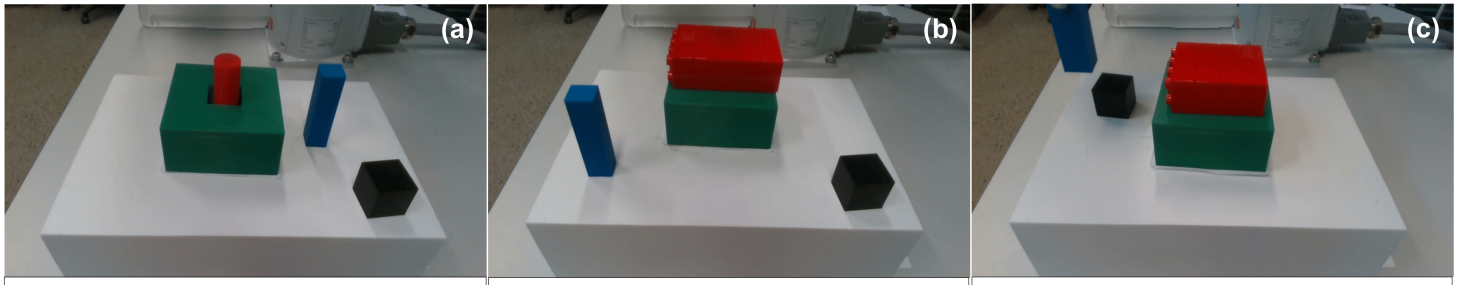


Figure 29: Runtime correction loop. Left: Skill fails due to unmet precondition. Center: VLM suggests missing symbolic knowledge. Right: Reactive planner integrates repair into current BT.

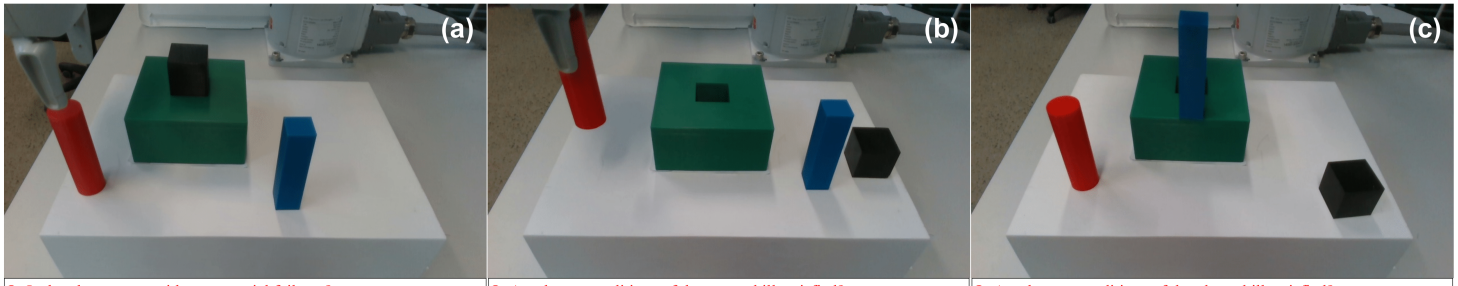


Figure 30: Visual verification loop. (a) Pre-execution check detects a missing condition. (b) PreVerifier finds an occlusion-related failure. (c) PostVerifier identifies incorrect placement. Each case triggers symbolic repair.

11.5 Discussion

This chapter addressed **RQ2.3** by presenting a unified framework that enables robots to detect, explain, and correct failures during execution through real-time symbolic monitoring and reactive plan repair. The proposed system extends the pre-execution framework of Chapter 10 with runtime modules, PreVerifier, PostVerifier, and Suggestor, that interface with a VLM and a reactive planner. This integration allows failure detection at every skill tick and correction through symbolic insertions, preserving modularity and interpretability.

Our evaluation confirms that combining proactive and reactive monitoring yields robust behavior across varied domains. In simulation, the framework succeeded on all REFLECT benchmark tasks without requiring post-execution replanning. On the physical YuMi robot, it handled occlusions, disturbances, and unexpected objects without human intervention. Compared to reactive-only or pre-execution-only strategies, our approach maintained high task success while reducing latency and VLM query costs.

Several limitations remain. First, the VLM operates over a fixed symbolic vocabulary; new predicates or concepts must be defined manually, limiting adaptability to unseen domains. Second, the reliance on RGB-D input can degrade under heavy occlusions or lighting changes. Future extensions may include fusing tactile or proprioceptive inputs for more robust scene grounding. Finally, each VLM query incurs non-trivial latency. We are currently exploring local fine-tuned models to improve response time and privacy.

Beyond immediate improvements, our longer-term goal is to enrich runtime reasoning. Future work will integrate holding condition monitoring and predictive checks to catch mid-execution hazards. Additionally, VLA models or diffusion policies could replace skill templates entirely, enabling autonomous skill suggestion with grounded pre/post-conditions. These enhancements aim to move closer to general purpose, self-correcting robots that operate reliably in unstructured, human-centered environments.

Concluding Remarks on Symbolic Failure Handling and Real-Time Adaptation (RQ2)

This part of the thesis addressed **RQ2**: *How can a robot detect, explain, and correct failures that arise before or during execution, while preserving modularity and interpretability?* The investigation was carried out in three stages.

First, Chapter 9 introduced a method for integrating predefined recovery behaviors into symbolic robot policies. Using the BTMG framework, both nominal and recovery skills were modeled uniformly, enabling reuse, modularity, and parameter adaptation. This ap-

proach supported recovery from known failures through structured planning and offline tuning.

Next, Chapter 10 extended this capability by addressing failures that occur before execution begins. The proposed system integrated a VLM into the planning loop to detect missing symbolic preconditions or absent skills using multimodal reasoning. A reactive planner then adapted the Behavior Tree accordingly. This enabled proactive correction before any motion occurred, improving robustness to environment-specific discrepancies.

Finally, Chapter 11 addressed failures that emerge during execution. The unified framework combined pre-execution verification with real-time monitoring of symbolic preconditions and postconditions. A runtime Suggestor module queried the VLM reactively when failures were detected, and the reactive planner modified the BT on-the-fly. This two-phase design allowed robots to adapt dynamically, detecting failures early, explaining them symbolically, and recovering without discarding the existing plan.

Together, these contributions demonstrate that structured symbolic policies can be augmented with visual and language-based reasoning to handle both anticipated and unforeseen failures. The resulting system achieves generality without sacrificing transparency, and supports continuous adaptation before and during execution. These insights lay the foundation for scalable, interpretable robot autonomy in unstructured environments.

Conclusions

This thesis presented a modular and introspective framework for robotic skill learning and failure recovery. It addressed two major challenges: how to represent and generalize interpretable robot policies across task variations, and how to enable autonomous failure detection, explanation, and correction through pre- and post-execution reasoning.

The first part of the thesis focused on representing and generalizing interpretable robot policies. BTMGs were used as a hybrid policy representation that combines symbolic task structure with parametric motion control, enabling modular and interpretable policy execution (RQ1.1). Policy generalization across task variations was achieved by learning smooth mappings from task descriptors to motion generator parameters using gaussian processes (RQ1.2). Real-time adaptation was supported through the PerF model, which dynamically inferred BTMG parameters during execution (RQ1.3). Together, these studies demonstrated that combining interpretable policy structures with data-driven parameter learning enables adaptive and efficient robotic behavior.

The second part of the thesis focused on failure detection and recovery. Recovery behaviors were encoded as BTMG representation, enabling local, reusable corrections (RQ2.1). To prevent execution-time failures, VLMs were integrated with a reactive planner and behavior trees for pre-execution reasoning, allowing the system to detect likely failures and insert missing skills or conditions before execution (RQ2.2). The framework was then extended with real-time monitoring, combining symbolic pre- and post-condition checkers, a Verifier module for continuous skill validation, and a Suggestor module to detect, explain, and correct failures as they occurred (RQ2.3). This unified framework supports continuous task execution without resetting, providing both proactive and reactive failure management.

The six research questions (RQ1.1–RQ2.3) were each addressed through a dedicated study. Table 1 summarizes the mapping between research questions, corresponding contributions, and how they were answered.

Table 1: Summary of research questions and their answers

Research Question	Addressed in	Answer Summary
RQ1.1: How can we design robotic policies that remain interpretable and adaptable?	Papers I	Utilized BTMGs as a hybrid representation separating symbolic structure from motion parameters, supporting modular and interpretable policies.
RQ1.2: How can these policies generalize to new task variations?	Papers II	Achieved parameter generalization via learned mappings from task descriptors to motion generator parameters using gaussian processes.
RQ1.3: How can policy parameters be efficiently inferred at runtime?	Paper III	Introduced PerF, a lightweight model for real-time parameter inference based on performance and feasibility surrogates.
RQ2.1: How can we design modular recovery behaviors?	Paper IV	Modeled recovery behaviors using the BTMG approach, which can be dynamically inserted to handle localized failures.
RQ2.2: How can failures be detected and explained before execution?	Paper V	Enabled pre-execution plan verification with VLMs to identify missing symbolic conditions, skills and suggest corrective steps.
RQ2.3: How can failures be detected and recovered from during execution?	Paper VI	Extended the framework with real-time symbolic monitoring, a Verifier for pre- and post-condition checks, and a Suggestor module to identify missing symbolic conditions, skills and suggest corrective steps.

This thesis demonstrates that symbolic structure and data-driven learning can complement each other effectively. BTMGs provide a transparent policy representation while supporting real-time adaptation. Modular recovery integrated into policy structure becomes a first-class capability rather than an afterthought. Finally, VLM-supported semantic reasoning closes the loop between perception, symbolic understanding, and policy adjustment, enabling robots to adapt and recover autonomously in dynamic environments.

Limitations and Future Work

While this thesis presents a modular and introspective framework for robotic skill learning and failure recovery, several limitations remain, which also suggest directions for future

research.

The current framework has been evaluated on tasks with moderate complexity, such as tabletop and mobile manipulation. Scaling to more challenging domains like humanoid locomotion, dexterous manipulation, or deformable object handling remains an open problem. These tasks require managing high-dimensional state spaces and long skill sequences, which could benefit from hierarchical policies and structured task decomposition.

Another limitation comes from the use of VLMs. Current VLMs are trained on general internet data and are not optimized for robotic scenarios, which can lead to semantic drift and occasional misinterpretation of task-specific queries. They also introduce latency, which makes real-time monitoring more challenging. A promising direction is to fine-tune VLMs or develop smaller, robot-focused models to improve both accuracy and responsiveness.

In addition, the framework currently relies on manually defined skills, which limits scalability for diverse or long-horizon tasks. A natural next step is to replace manually written skills with fine-tuned VLA models or diffusion policies that generate executable trajectories. These trajectories could then be automatically wrapped into skills using VLM-extracted preconditions and postconditions. These skills can then be autonomously generated and embedded within the behavior tree and reactive planning framework. This allows the system to monitor execution, handle failures, and maintain long-horizon task performance. By wrapping behaviors generated by VLA models or diffusion policies in this structured framework, the system overcomes one of the main weaknesses of purely data-driven methods. It gains the ability to supervise execution and recover from errors at the skill level.

Another limitation is that the system does not currently output confidence or feasibility scores for its suggested actions. This makes the framework vulnerable to edge cases where proposed skills might be unsafe or infeasible. Future work could explore integrating static verification and continuous integration checks inspired by software testing, ensuring that foundation model outputs are feasible and safe before execution.

Finally, proactive failure prediction is still limited. While the framework performs pre-execution verification and real-time monitoring, it does not fully anticipate holding condition violations during skill execution. Extending the system with predictive monitoring, simulation based foresight, and probabilistic reasoning would allow robots to prevent failures before they occur, further improving robustness.

Overall, future work lies in scaling to more complex domains, adopting fine-tuned VLMs and VLAs or diffusion policies for automated skill synthesis, and integrating predictive and safety-aware mechanisms. These directions aim to enhance the system's autonomy, reliability, and adaptability in open-ended real-world environments.

References

[1] Bruno Siciliano and Oussama Khatib. *Springer Handbook of Robotics*. Berlin, Heidelberg: Springer-Verlag, 2007. ISBN: 354023957X.

[2] Sebastian Thrun, Wolfram Burgard and Dieter Fox. *Probabilistic Robotics (Intelligent Robotics and Autonomous Agents)*. The MIT Press, 2005. ISBN: 0262201623.

[3] Faseeh Ahmad et al. "A Unified Framework for Real-Time Failure Handling in Robotics Using Vision-Language Models, Reactive Planner and Behavior Trees". In: *arXiv preprint* (2025). arXiv: 2503.15202 [cs.RO]. URL: https://arxiv.org/abs/2503.15202.

[4] Wikipedia contributors. *Autonomous robot*. Accessed: 2025-05-07. 2024. URL: https://en.wikipedia.org/wiki/Autonomous_robot.

[5] Niryo. *What is an autonomous robot?* Accessed: 2025-05-07. 2024. URL: https://niryo.com/what-is-an-autonomous-robot/.

[6] 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)*. 2022, pp. 1995–2002. DOI: 10.1109/ROBIO55434.2022.10011996.

[7] 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)*. 2023, pp. 10133–10140. DOI: 10.1109/IROS55552.2023.10341636.

[8] Faseeh Ahmad, Jonathan Styrud and Volker Krueger. "Addressing failures in robotics using vision-based language models (vlms) and behavior trees (bt)". In: *arXiv preprint arXiv:2411.01568* (2024).

[9] Ahmad, Faseeh and Mayr, Matthias and Suresh-Fazeela, Sulthan and Krueger, Volker. "Adaptable Recovery Behaviors in Robotics : A Behavior Trees and Motion Generators (BTMG) Approach for Failure Management". eng. In: *2024 IEEE 20th International Conference on Automation Science and Engineering (CASE)*. IEEE Computer Society, 2024, 1815–1822. ISBN: 9798350358513. DOI: {10.1109/CASE59546.

2024.10711715}. URL: %7Bhttp://dx.doi.org/10.1109/CASE59546.2024.10711715%7D.

[10] Michele Colledanchise and Petter Ögren. *Behavior Trees in Robotics and AI: An Introduction*. CRC Press, 2018. ISBN: 978-1-138-59373-4. URL: https://www.taylorfrancis.com/books/mono/10.1201/9780429489105/behavior-trees-robotics-ai-michele-colledanchise-petter-%C3%B6gren.

[11] Danny Driess et al. "Palm-e: An embodied multimodal language model". In: (2023).

[12] Michele Colledanchise, Alejandro Marzinotto and Petter Ögren. "Performance analysis of stochastic behavior trees". In: (2014), pp. 3265–3272. DOI: 10.1109/ICRA.2014.6907328.

[13] Francesco 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)*. IEEE. 2018, pp. 5964–5971.

[14] Jean-Baptiste Alayrac et al. "Flamingo: a visual language model for few-shot learning". In: *Advances in neural information processing systems* 35 (2022), pp. 23716–23736.

[15] Anas Awadalla et al. "Openflamingo: An open-source framework for training large autoregressive vision-language models". In: *arXiv preprint arXiv:2308.01390* (2023).

[16] Anthony Brohan et al. "Rt-1: Robotics transformer for real-world control at scale". In: *arXiv preprint arXiv:2212.06817* (2022).

[17] Anthony Brohan et al. "Rt-2: Vision-language-action models transfer web knowledge to robotic control". In: *arXiv preprint arXiv:2307.15818* (2023).

[18] Cheng Chi et al. "Diffusion policy: Visuomotor policy learning via action diffusion". In: *The International Journal of Robotics Research* (2023), p. 02783649241273668.

[19] 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.

[20] Auke Jan Ijspeert et al. "Dynamical movement primitives: learning attractor models for motor behaviors". In: *Neural computation* 25.2 (2013), pp. 328–373.

[21] Damian Isla. "Handling Complexity in the Halo 2 AI". In: *Game Developers Conference (GDC)*. Available at: https://www.gamedeveloper.com/programming/gdc-2005-proceeding-handling-complexity-in-the-i-halo-2-i-ai. 2005.

[22] Alex Champandard. "Understanding behavior trees". In: *AiGameDev. com* 6.328 (2007), p. 19.

[23] Matteo Iovino et al. "A survey of behavior trees in robotics and ai". In: *Robotics and Autonomous Systems* 154 (2022), p. 104096.

[24] Rodney A. Brooks. "A Robust Layered Control System for a Mobile Robot". In: *IEEE Journal of Robotics and Automation* 2.1 (1986), pp. 14–23. DOI: 10.1109/JRA.1986.1087032. URL: https://people.csail.mit.edu/brooks/papers/AIM-864.pdf.

[25] Hiren D Patel and Sandeep K Shukla. "Behavioral hierarchy with hierarchical FSMs (HFSMs)". In: *Ingredients for Successful System Level Design Methodology* (2008), pp. 47–75.

[26] Tadao Murata. "Petri Nets: Properties, Analysis and Applications". In: *Proceedings of the IEEE* 77.4 (1989), pp. 541–580. DOI: 10.1109/5.24143.

[27] Kutluhan Erol, James Hendler and Dana S. Nau. "HTN Planning: Complexity and Expressivity". In: *Proceedings of the 12th National Conference on Artificial Intelligence (AAAI)*. 1994, pp. 1123–1128.

[28] Francesco Rovida, Bjarne Grossmann and Volker Krüger. "Extended behavior trees for quick definition of flexible robotic tasks". In: *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2017, pp. 6793–6800.

[29] Alejandro Marzinotto et al. "Towards a unified behavior trees framework for robot control". In: *2014 IEEE International Conference on Robotics and Automation (ICRA)*. 2014, pp. 5420–5427. DOI: 10.1109/ICRA.2014.6907656.

[30] Petter Ogren. "Increasing modularity of UAV control systems using computer game behavior trees". In: *Aiaa guidance, navigation, and control conference*. 2012, p. 4458.

[31] 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 (2017), pp. 372–389. DOI: 10.1109/TRO.2016.2633567.

[32] 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*. 2020, pp. 196–209.

[33] 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.

[34] 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)*. IEEE. 2022, pp. 1153–1160.

[35] Matteo Iovino et al. "A framework for learning behavior trees in collaborative robotic applications". In: *2023 IEEE 19th International Conference on Automation Science and Engineering (CASE)*. IEEE. 2023, pp. 1–8.

[36] Kevin French et al. "Learning behavior trees from demonstration". In: *2019 International Conference on Robotics and Automation (ICRA)*. IEEE. 2019, pp. 7791–7797.

[37] Chris Paxton et al. “Representing robot task plans as robust logical-dynamical systems”. In: *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2019, pp. 5588–5595.

[38] Jonathan Styrud et al. “Bebop-combining reactive planning and bayesian optimization to solve robotic manipulation tasks”. In: *2024 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2024, pp. 16459–16466.

[39] Matteo Iovino et al. “A Framework for Learning Behavior Trees in Collaborative Robotic Applications”. In: *2023 IEEE 19th International Conference on Automation Science and Engineering (CASE)*. 2023, pp. 1–8. DOI: 10.1109/CASE56687.2023.10260363.

[40] Matteo Iovino. “Learning Behavior Trees for Collaborative Robotics”. PhD thesis. KTH Royal Institute of Technology, 2023.

[41] Christian Fritz and Sheila McIlraith. “Generating optimal plans in highly-dynamic domains”. In: *arXiv preprint arXiv:1205.2647* (2012).

[42] Auke Ijspeert, Jun Nakanishi and Stefan Schaal. “Learning attractor landscapes for learning motor primitives”. In: *Advances in neural information processing systems* 15 (2002).

[43] Mayr, Matthias and Rovida, Francesco and Krueger, Volker. “SkiROS2: A Skill-Based Robot Control Platform for ROS”. eng. In: *2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE - Institute of Electrical and Electronics Engineers Inc., 12 2023, 6273–6280. ISBN: 978-1-6654-9190-7. DOI: {10.1109/IROS55552.2023.10342216}. URL: %7Bhttp://dx.doi.org/10.1109/IROS55552.2023.10342216%7D.

[44] Malik Ghallab, Dana S. Nau and Paolo Traverso. *Automated Planning: Theory and Practice*. Amsterdam, Netherlands: Elsevier, 2004.

[45] Malte Helmert. “The fast downward planning system”. In: *Journal of Artificial Intelligence Research* 26 (2006), pp. 191–246.

[46] Jonathan Styrud et al. “Combining Planning and Learning of Behavior Trees for Robotic Assembly”. In: *2022 International Conference on Robotics and Automation (ICRA)*. 2022, pp. 11511–11517. DOI: 10.1109/ICRA46639.2022.9812086.

[47] 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)*. 2018, pp. 5672–5678. DOI: 10.1109/IROS.2018.8593566.

[48] 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)*. 2018, pp. 512–519. DOI: 10.1109/ICRA.2018.8460964.

[49] Moritz Tenorth and Michael Beetz. "KnowRob: A knowledge processing infrastructure for cognition-enabled robots". In: *The International Journal of Robotics Research* 32.5 (2013), pp. 566–590. DOI: 10.1177/0278364913481635. eprint: https://doi.org/10.1177/0278364913481635. URL: https://doi.org/10.1177/0278364913481635.

[50] Barto Andrew and Sutton Richard S. "Reinforcement learning: an introduction". In: (2018).

[51] Volodymyr Mnih et al. "Human-level control through deep reinforcement learning". In: *nature* 518.7540 (2015), pp. 529–533.

[52] Chen Tang et al. "Deep reinforcement learning for robotics: A survey of real-world successes". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 39. 27. 2025, pp. 28694–28698.

[53] Wenshuai Zhao, Jorge Peña Queralta and Tomi Westerlund. "Sim-to-real transfer in deep reinforcement learning for robotics: a survey". In: *2020 IEEE symposium series on computational intelligence (SSCI)*. IEEE. 2020, pp. 737–744.

[54] Timothy P Lillicrap et al. "Continuous control with deep reinforcement learning". In: *arXiv preprint arXiv:1509.02971* (2015).

[55] Tuomas Haarnoja et al. "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor". In: *International conference on machine learning*. Pmlr. 2018, pp. 1861–1870.

[56] Marc Deisenroth and Carl E 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.

[57] Josh 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)*. IEEE. 2017, pp. 23–30.

[58] Ofir Nachum et al. "Data-efficient hierarchical reinforcement learning". In: *Advances in neural information processing systems* 31 (2018).

[59] Konstantinos Chatzilygeroudis et al. "Black-box data-efficient policy search for robotics". In: *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2017, pp. 51–58.

[60] Tim Salimans et al. "Evolution strategies as a scalable alternative to reinforcement learning". In: *arXiv preprint arXiv:1703.03864* (2017).

[61] James Bergstra and Yoshua Bengio. "Random search for hyper-parameter optimization". In: *The journal of machine learning research* 13.1 (2012), pp. 281–305.

[62] Andras Kupcsik et al. "Data-efficient generalization of robot skills with contextual policy search". In: *Proceedings of the AAAI conference on artificial intelligence*. Vol. 27. 1. 2013, pp. 1401–1407.

[63] Chelsea Finn et al. "One-shot visual imitation learning via meta-learning". In: *Conference on robot learning*. PMLR. 2017, pp. 357–368.

[64] Peter I. Frazier. *A Tutorial on Bayesian Optimization*. 2018. arXiv: 1807.02811 [stat.ML]. URL: https://arxiv.org/abs/1807.02811.

[65] Donald R Jones, Matthias Schonlau and William J Welch. "Efficient global optimization of expensive black-box functions". In: *Journal of Global optimization* 13 (1998), pp. 455–492.

[66] Niranjan Srinivas et al. "Gaussian process optimization in the bandit setting: No regret and experimental design". In: *arXiv preprint arXiv:0912.3995* (2009).

[67] Harold J Kushner. "A new method of locating the maximum point of an arbitrary multipeak curve in the presence of noise". In: (1964).

[68] Shinya Suzuki et al. "Multi-objective Bayesian optimization using Pareto-frontier entropy". In: *International conference on machine learning*. PMLR. 2020, pp. 9279–9288.

[69] Kaisa Miettinen. *Nonlinear multiobjective optimization*. Vol. 12. Springer Science & Business Media, 1999.

[70] R Timothy Marler and Jasbir S Arora. "Survey of multi-objective optimization methods for engineering". In: *Structural and multidisciplinary optimization* 26 (2004), pp. 369–395.

[71] Roman Garnett. *Bayesian optimization*. Cambridge University Press, 2023.

[72] Samuel Daulton, Maximilian Balandat and Eytan Bakshy. "Differentiable expected hypervolume improvement for parallel multi-objective Bayesian optimization". In: *Advances in Neural Information Processing Systems* 33 (2020), pp. 9851–9864.

[73] Kalyanmoy Deb et al. "A fast and elitist multiobjective genetic algorithm: NSGA-II". In: *IEEE transactions on evolutionary computation* 6.2 (2002), pp. 182–197.

[74] Dario Amodei et al. *Concrete Problems in AI Safety*. 2016. arXiv: 1606.06565 [cs.AI]. URL: https://arxiv.org/abs/1606.06565.

[75] Andrew Y Ng, Daishi Harada and Stuart Russell. "Policy invariance under reward transformations: Theory and application to reward shaping". In: *Icml*. Vol. 99. Citeseer. 1999, pp. 278–287.

[76] Yoshua Bengio et al. "Curriculum learning". In: *Proceedings of the 26th annual international conference on machine learning*. 2009, pp. 41–48.

[77] Joshua Achiam et al. "Constrained policy optimization". In: *International conference on machine learning*. PMLR. 2017, pp. 22–31.

[78] Christopher KI Williams and Carl Edward Rasmussen. *Gaussian processes for machine learning*. Vol. 2. 3. MIT press Cambridge, MA, 2006.

[79] Marc Peter Deisenroth, Dieter Fox and Carl Edward Rasmussen. "Gaussian processes for data-efficient learning in robotics and control". In: *IEEE transactions on pattern analysis and machine intelligence* 37.2 (2013), pp. 408–423.

[80] Miao Liu et al. "Gaussian processes for learning and control: A tutorial with examples". In: *IEEE Control Systems Magazine* 38.5 (2018), pp. 53–86.

[81] 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)*. IEEE. 2018, pp. 3803–3810.

[82] Adil Zouitine et al. "Rrls: Robust reinforcement learning suite". In: *arXiv preprint arXiv:2406.08406* (2024).

[83] Claudio Urrea and Rayko Agramonte. "Kalman filter: historical overview and review of its use in robotics 60 years after its creation". In: *Journal of Sensors* 2021.1 (2021), p. 9674015.

[84] Christopher Edwards, Sarah K Spurgeon and Ron J Patton. "Sliding mode observers for fault detection and isolation". In: *Automatica* 36.4 (2000), pp. 541–553.

[85] Steven X Ding. *Model-based fault diagnosis techniques: design schemes, algorithms, and tools*. Springer Science & Business Media, 2008.

[86] Chen Xu et al. *Can We Detect Failures Without Failure Data? Uncertainty-Aware Runtime Failure Detection for Imitation Learning Policies*. 2025. arXiv: 2503.08558 [cs.RO]. URL: https://arxiv.org/abs/2503.08558.

[87] Chang Nho Cho, Ji Tae Hong and Hong Ju Kim. "Neural network based adaptive actuator fault detection algorithm for robot manipulators". In: *Journal of Intelligent & Robotic Systems* 95 (2019), pp. 137–147.

[88] Cristina Cornelio and Mohammed Diab. "Recover: A neuro-symbolic framework for failure detection and recovery". In: *2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2024, pp. 12435–12442.

[89] Seyed Mohammad Khansari-Zadeh and Aude Billard. "A dynamical system approach to realtime obstacle avoidance". In: *Autonomous Robots* 32 (2012), pp. 433–454.

[90] Arda Inceoglu et al. "Fino-net: A deep multimodal sensor fusion framework for manipulation failure detection". In: *2021 IEEE/RSJ international conference on intelligent robots and systems (IROS)*. IEEE. 2021, pp. 6841–6847.

[91] Zeyi Liu, Arpit Bahety and Shuran Song. "Reflect: Summarizing robot experiences for failure explanation and correction". In: *arXiv preprint arXiv:2306.15724* (2023).

[92] Yanjiang Guo et al. “Doremi: Grounding language model by detecting and recovering from plan-execution misalignment”. In: *2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2024, pp. 12124–12131.

[93] Jiafei Duan et al. “AHA: A vision-language-model for detecting and reasoning over failures in robotic manipulation”. In: *arXiv preprint arXiv:2410.00371* (2024).

[94] Suzanne Tolmeijer et al. “Taxonomy of Trust-Relevant Failures and Mitigation Strategies”. In: *2020 15th ACM/IEEE International Conference on Human-Robot Interaction (HRI)*. 2020, pp. 3–12.

[95] Enshen Zhou et al. “Code-as-monitor: Constraint-aware visual programming for reactive and proactive robotic failure detection”. In: *Proceedings of the Computer Vision and Pattern Recognition Conference*. 2025, pp. 6919–6929.

[96] Gregory LeMasurier et al. “Reactive or proactive? how robots should explain failures”. In: *Proceedings of the 2024 ACM/IEEE International Conference on Human-Robot Interaction*. 2024, pp. 413–422.

[97] Ruichao Wu, Sitar Kortik and Christoph Hellmann Santos. “Automated behavior tree error recovery framework for robotic systems”. In: *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2021, pp. 6898–6904.

[98] Joonho Lee, Jemin Hwangbo and Marco Hutter. “Robust recovery controller for a quadrupedal robot using deep reinforcement learning”. In: *arXiv preprint arXiv:1901.07517* (2019).

[99] Jonathan Booher et al. “CIMRL: Combining IMitation and Reinforcement Learning for Safe Autonomous Driving”. In: *arXiv preprint arXiv:2406.08878* (2024).

[100] Aoran Mei et al. “ReplanVLM: Replanning robotic tasks with visual language models”. In: *IEEE Robotics and Automation Letters* (2024).

[101] Zhouliang Yu et al. “Multireact: Multimodal tools augmented reasoning-acting traces for embodied agent planning”. In: (2023).

[102] Yinpei Dai et al. “RACER: Rich Language-Guided Failure Recovery Policies for Imitation Learning”. In: *arXiv preprint arXiv:2409.14674* (2024).

[103] Jiaming Liu et al. “Self-corrected multimodal large language model for end-to-end robot manipulation”. In: *arXiv preprint arXiv:2405.17418* (2024).

[104] Aneseh Alvanpour et al. “Robot failure mode prediction with explainable machine learning”. In: *2020 IEEE 16th International Conference on Automation Science and Engineering (CASE)*. IEEE. 2020, pp. 61–66.

[105] Maximilian Diehl and Karinne Ramirez-Amaro. “A causal-based approach to explain, predict and prevent failures in robotic tasks”. In: *Robotics and Autonomous Systems* 162 (2023), p. 104376.

[106] Kanghyun Ryu et al. “Curricullm: Automatic task curricula design for learning complex robot skills using large language models”. In: *arXiv preprint arXiv:2409.18382* (2024).

[107] 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)*. IEEE. 2019, pp. 8839–8845.

[108] Shivam Vats et al. “Recoverychaining: Learning local recovery policies for robust manipulation”. In: *arXiv preprint arXiv:2410.13979* (2024).

[109] Kyriacos Shiarlis, Joao Messias and Shimon Whiteson. “Inverse reinforcement learning from failure”. In: (2016).

[110] Amirreza Razmjoo et al. “CCDP: Composition of Conditional Diffusion Policies with Guided Sampling”. In: *arXiv preprint arXiv:2503.15386* (2025).

[111] Sajad Shahsavari et al. “Remote Run-Time Failure Detection and Recovery Control For Quadcopters”. In: *Journal of Integrated Design and Process Science* 25.2 (2022), pp. 120–140.

[112] Huihan Liu et al. “Model-Based Runtime Monitoring with Interactive Imitation Learning”. In: *IEEE International Conference on Robotics and Automation (ICRA)*. 2024.

[113] Rishi Bommasani et al. “On the opportunities and risks of foundation models”. In: *arXiv preprint arXiv:2108.07258* (2021).

[114] Tom Brown et al. “Language models are few-shot learners”. In: *Advances in neural information processing systems* 33 (2020), pp. 1877–1901.

[115] Josh Achiam et al. “Gpt-4 technical report”. In: *arXiv preprint arXiv:2303.08774* (2023).

[116] Aakanksha Chowdhery et al. “Palm: Scaling language modeling with pathways”. In: *Journal of Machine Learning Research* 24.240 (2023), pp. 1–113.

[117] Jacob Devlin et al. “Bert: Pre-training of deep bidirectional transformers for language understanding”. In: *Proceedings of the 2019 conference of the North American chapter of the association for computational linguistics: human language technologies, volume 1 (long and short papers)*. 2019, pp. 4171–4186.

[118] Colin Raffel et al. “Exploring the limits of transfer learning with a unified text-to-text transformer”. In: *Journal of machine learning research* 21.140 (2020), pp. 1–67.

[119] Kaiming He et al. “Deep residual learning for image recognition”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 770–778.

[120] Alexey Dosovitskiy et al. “An image is worth 16x16 words: Transformers for image recognition at scale”. In: *arXiv preprint arXiv:2010.11929* (2020).

[121] Tadas Baltrušaitis, Chaitanya Ahuja and Louis-Philippe Morency. "Multimodal machine learning: A survey and taxonomy". In: *IEEE transactions on pattern analysis and machine intelligence* 41.2 (2018), pp. 423–443.

[122] Yao-Hung Hubert Tsai et al. "Multimodal transformer for unaligned multimodal language sequences". In: *Proceedings of the conference. Association for computational linguistics. Meeting*. Vol. 2019. 2019, p. 6558.

[123] Hassan Akbari et al. "Vatt: Transformers for multimodal self-supervised learning from raw video, audio and text". In: *Advances in neural information processing systems* 34 (2021), pp. 24206–24221.

[124] Alec Radford et al. "Learning transferable visual models from natural language supervision". In: *International conference on machine learning*. PmLR. 2021, pp. 8748–8763.

[125] Chao Jia et al. "Scaling up visual and vision-language representation learning with noisy text supervision". In: *International conference on machine learning*. PMLR. 2021, pp. 4904–4916.

[126] Junnan Li et al. "Blip-2: Bootstrapping language-image pre-training with frozen image encoders and large language models". In: *International conference on machine learning*. PMLR. 2023, pp. 19730–19742.

[127] Deyao Zhu et al. "Minigpt-4: Enhancing vision-language understanding with advanced large language models". In: *arXiv preprint arXiv:2304.10592* (2023).

[128] Christoph Schuhmann et al. "Laion-400m: Open dataset of clip-filtered 400 million image-text pairs". In: *arXiv preprint arXiv:2111.02114* (2021).

[129] Piyush Sharma et al. "Conceptual captions: A cleaned, hypernymed, image alt-text dataset for automatic image captioning". In: *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. 2018, pp. 2556–2565.

[130] Haotian Liu et al. *Visual Instruction Tuning*. 2023. arXiv: `2304.08485 [cs.CV]`. URL: `https://arxiv.org/abs/2304.08485`.

[131] Mohit Shridhar, Lucas Manuelli and Dieter Fox. "Cliport: What and where pathways for robotic manipulation". In: *Conference on robot learning*. PMLR. 2022, pp. 894–906.

[132] Mohit Shridhar, Lucas Manuelli and Dieter Fox. "Perceiver-actor: A multi-task transformer for robotic manipulation". In: *Conference on Robot Learning*. PMLR. 2023, pp. 785–799.

[133] Michael Ahn et al. "Do as i can, not as i say: Grounding language in robotic affordances". In: *arXiv preprint arXiv:2204.01691* (2022).

[134] Moo Jin Kim et al. “Openvla: An open-source vision-language-action model”. In: *arXiv preprint arXiv:2406.09246* (2024).

[135] Mohit Shridhar, Lucas Manuelli and Dieter Fox. “Perceiver-actor: A multi-task transformer for robotic manipulation”. In: *Conference on Robot Learning*. PMLR. 2023, pp. 785–799.

[136] Yunfan Jiang et al. “Vima: General robot manipulation with multimodal prompts”. In: *arXiv preprint arXiv:2210.03094* 2.3 (2022), p. 6.

[137] Hongyi Chen et al. “Automating Robot Failure Recovery Using Vision-Language Models With Optimized Prompts”. In: *arXiv preprint arXiv:2409.03966* (2024).

[138] Junnan Li et al. “Align before fuse: Vision and language representation learning with momentum distillation”. In: *Advances in neural information processing systems* 34 (2021), pp. 9694–9705.

[139] Zongxia Li et al. “Benchmark evaluations, applications, and challenges of large vision language models: A survey”. In: *arXiv preprint arXiv:2501.02189* 1 (2025).

[140] Marc Peter Deisenroth, Gerhard Neumann, Jan Peters et al. “A survey on policy search for robotics”. In: *Foundations and Trends® in Robotics* 2.1–2 (2013), pp. 1–142.

[141] Auke Jan Ijspeert, Jun Nakanishi and Stefan Schaal. “Movement imitation with nonlinear dynamical systems in humanoid robots”. In: *Proceedings 2002 IEEE International Conference on Robotics and Automation (Cat. No. 02CH37292)*. Vol. 2. IEEE. 2002, pp. 1398–1403.

[142] Aude G. Billard, Sylvain Calinon and Florent Guenter. “Discriminative and adaptive imitation in uni-manual and bi-manual tasks”. In: *Robotics and Autonomous Systems* 54.5 (2006). The Social Mechanisms of Robot Programming from Demonstration, pp. 370–384. ISSN: 0921-8890. DOI: https://doi.org/10.1016/j.robot.2006.01.007. URL: https://www.sciencedirect.com/science/article/pii/S0921889006000170.

[143] Stefan Schaal. “Dynamic movement primitives-a framework for motor control in humans and humanoid robotics”. In: *Adaptive motion of animals and machines*. Springer, 2006, pp. 261–280.

[144] 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.

[145] Konstantinos Chatzilygeroudis et al. “A survey on policy search algorithms for learning robot controllers in a handful of trials”. In: *IEEE Transactions on Robotics* 36.2 (2019), pp. 328–347.

[146] Michael Beetz et al. “Towards performing everyday manipulation activities”. In: *Robotics and Autonomous Systems* 58.9 (2010), pp. 1085–1095.

[147] Matthew Grounds and Daniel Kudenko. "Combining reinforcement learning with symbolic planning". In: *European Symposium on Adaptive Agents and Multi-Agent Systems*. Springer. 2005, pp. 75–86.

[148] Fangkai Yang et al. "Peorl: Integrating symbolic planning and hierarchical reinforcement learning for robust decision-making". In: *arXiv preprint arXiv:1804.07779* (2018).

[149] Vasanth Sarathy et al. "Spotter: Extending symbolic planning operators through targeted reinforcement learning". In: *arXiv preprint arXiv:2012.13037* (2020).

[150] Daniel Gordon, Dieter Fox and Ali Farhadi. "What should i do now? marrying reinforcement learning and symbolic planning". In: *arXiv preprint arXiv:1901.01492* (2019).

[151] 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*. IEEE. 2011, pp. 590–595.

[152] Denis Forte et al. "On-line motion synthesis and adaptation using a trajectory database". In: *Robotics and Autonomous Systems* 60.10 (2012), pp. 1327–1339.

[153] You Zhou and Tamim Asfour. "Task-oriented generalization of dynamic movement primitive". In: *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2017, pp. 3202–3209.

[154] Michael D McKay, Richard J Beckman and William J Conover. "A comparison of three methods for selecting values of input variables in the analysis of output from a computer code". In: *Technometrics* 42.1 (2000), pp. 55–61.

[155] Aleš Ude et al. "Task-specific generalization of discrete and periodic dynamic movement primitives". In: *IEEE Transactions on Robotics* 26.5 (2010), pp. 800–815.

[156] 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*. IEEE. 2011, pp. 590–595.

[157] S Mohammad Khansari-Zadeh and Aude Billard. "Learning stable nonlinear dynamical systems with gaussian mixture models". In: *IEEE Transactions on Robotics* 27.5 (2011), pp. 943–957.

[158] Alexandros Paraschos et al. "Probabilistic movement primitives". In: *Advances in neural information processing systems* 26 (2013).

[159] 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*. IEEE. 2010, pp. 411–416.

[160] Katharina Mülling et al. “Learning to select and generalize striking movements in robot table tennis”. In: *The International Journal of Robotics Research* 32.3 (2013), pp. 263–279.

[161] 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)*. IEEE. 2019, pp. 347–358.

[162] Chih-Chung Chang and Chih-Jen Lin. “LIBSVM: A library for support vector machines”. In: *ACM transactions on intelligent systems and technology (TIST)* 2.3 (2011), pp. 1–27.

[163] Dong C Liu and Jorge Nocedal. “On the limited memory BFGS method for large scale optimization”. In: *Mathematical programming* 45.1 (1989), pp. 503–528.

[164] Alessio De Luca, Luca Muratore and Nikos G Tsagarakis. “Autonomous navigation with online replanning and recovery behaviors for wheeled-legged robots using behavior trees”. In: *IEEE Robotics and Automation Letters* 8.10 (2023), pp. 6803–6810.

[165] Corrado Pezzato et al. “Active inference and behavior trees for reactive action planning and execution in robotics”. In: *IEEE Transactions on Robotics* 39.2 (2023), pp. 1050–1069.

[166] Hongmin Wu et al. “Recovering from external disturbances in online manipulation through state-dependent revertive recovery policies”. In: *2018 27th IEEE International Symposium on Robot and Human Interactive Communication (RO-MAN)*. IEEE. 2018, pp. 166–173.

[167] Zhaoyuan Gu, Nathan Boyd and Ye Zhao. “Reactive locomotion decision-making and robust motion planning for real-time perturbation recovery”. In: *2022 International Conference on Robotics and Automation (ICRA)*. IEEE. 2022, pp. 1896–1902.

[168] Xiaohan Zhang et al. “Grounding classical task planners via vision-language models”. In: *arXiv preprint arXiv:2304.08587* (2023).

[169] Caelan Reed Garrett et al. “Integrated task and motion planning”. In: *Annual review of control, robotics, and autonomous systems* 4.1 (2021), pp. 265–293.

[170] Danil S Grigorev, Alexey K Kovalev and Aleksandr I Panov. “VerifyLLM: LLM-Based Pre-Execution Task Plan Verification for Robots”. In: *arXiv preprint arXiv:2507.05118* (2025).

[171] Jason Wei et al. “Chain-of-thought prompting elicits reasoning in large language models”. In: *Advances in neural information processing systems* 35 (2022), pp. 24824–24837.

[172] Shilong Liu et al. “Grounding dino: Marrying dino with grounded pre-training for open-set object detection”. In: *European conference on computer vision*. Springer. 2024, pp. 38–55.

[173] Nikhila Ravi et al. “Sam 2: Segment anything in images and videos”. In: *arXiv preprint arXiv:2408.00714* (2024).

Scientific publications

Contribution Statements

Table 2: Overview of contributions in each paper included in the thesis.

Paper	*Writing*	*Concepts*	*Implementation*	*Evaluation*
I	●	◕	●	●
II	◐	◔	◐	◐
III	◕	◕	◕	◕
IV	●	◕	◕	●
V	●	◕	◕	●
VI	●	◕	◐	◔

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

- ◔ Faseeh Ahmad was a minor contributor to the work
- ◐ Faseeh Ahmad was a contributor to the work
- ◕ Faseeh Ahmad led and did a majority of the work
- ● Faseeh Ahmad led and did almost all of the work

Co-authors are abbreviated as follows:

Table 3: Author Acronyms

Acronym	Name
MM	Matthias Mayr
KC	Konstantinos Chatzilygeroudis
LN	Luigi Nardi
EAT	Elin Anna Topp
JM	Jacek Malec
VK	Volker Krueger
HI	Hashim Ismail
JS	Jonathan Styrud
MS	Maj Stenmark
SSF	Sulthan Suresh-Fazeela

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

MM led the framework design and implementation of the reinforcement learning and SkiROS2 integration. I developed the task-specific skills and planning integration. MM and I jointly conducted experiments in simulation and on the KUKA iiwa. LN contributed to the Bayesian optimization component. MM wrote most of the paper, with feedback from KC and others. I contributed to discussions, experiments, and writing support.

Paper II: Generalizing Behavior Trees and Motion-Generator (BTMG) Policy Representation for Robotic Tasks Over Scenario Parameters

The initial idea was developed in collaboration with MM. I implemented the full framework, designed and ran the experiments, and collected the results. MM supported the formulation and provided feedback throughout. I wrote the paper, with contributions and reviews from all co-authors. The notion of intrinsic and extrinsic parameters was introduced by me, with input from VK. EAT provided valuable feedback on the paper.

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

This idea was co-developed by me and MM. I led the model design, including architecture and input-output mapping. MM supported the BT and reinforcement learning setup. I implemented the core model, extended Skireil for task variations, and performed the

majority of experiments. MM supported baseline implementation. I wrote most of the paper, with valuable feedback from MM.

Paper IV: Adaptable Recovery Behaviors in Robotics: A Behavior Trees and Motion Generators (BTMG) Approach for Failure Management

MM initially proposed failure recovery with BTMGs. I designed the framework, implemented the dual-arm skills in SkiROS2, and extended Skireil. SSF, our master's student, contributed code for dual-arm tasks. I set up and conducted all simulation experiments, including peg-in-hole failure scenarios. MM supported real-robot setup and debugging. I wrote the full paper, with feedback from MM and SSF.

Paper V: Addressing Failures in Robotics using Vision-Based Language Models (VLMs) and Behavior Trees (BT)

I designed the architecture combining VLMs with Behavior Trees, implemented the system, and conducted most experiments in MuJoCo. JS provided the initial codebase, helped with early setup, and contributed initial results. I wrote the full manuscript, with support and reviews from JS and VK.

Paper VI: A Unified Framework for Real-Time Failure Handling in Robotics Using Vision-Language Models, Reactive Planner and Behavior Trees

I developed the initial concept and worked closely with HI to refine the overall framework. JS provided the initial repository that served as a baseline. HI implemented most of the skill logic and the vision pipeline. I wrote the prompts and condition checks, which HI integrated and refined. HI and I jointly ran simulations and performed real-robot experiments, with HI finalizing setup. I wrote the manuscript, with valuable feedback from HI, JS, VK, and MS.

Paper I

M. Mayr, F. Ahmad, K. Chatzilygeroudis, L. Nardi, V. Krueger
Skill-based Multi-objective Reinforcement Learning of Industrial Robot Tasks with Planning and Knowledge Integration
IEEE International Conference on Robotics and Biomimetics (ROBIO), Jinghong, China, 2022, pp. 1995-2002, doi: 10.1109/ROBIO55434.2022.10011996

Chapter 1

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

1 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.

2 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* [1, 2, 3]. 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 [4, 5]. The skill-based approach is powerful when knowledge can be modeled and formalized *explicitly* [1, 2]. 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 above-mentioned 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 [6]. 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 [7] Moreover, the simulation-to-reality gap [8, 9] is bigger in lower-level control states (i.e. torques), and policies working directly on raw control states struggle to transfer learned behaviors to the real systems [6]. Our policy formulation consisting of behavior trees (BT) with a motion generator [10] has shown to be able to learn interpretable and robust behaviors [11].

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

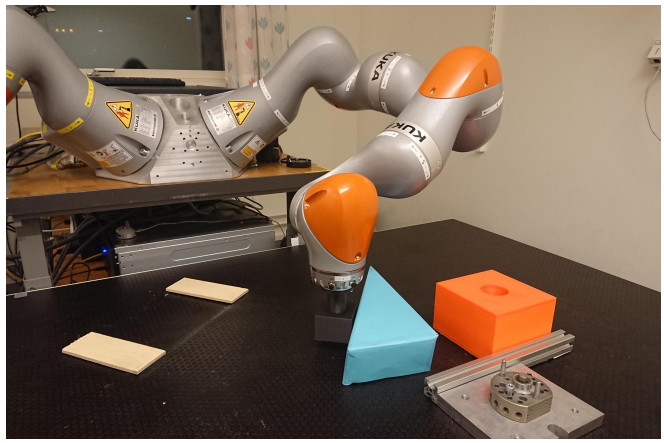


Figure 1.1: The robot setup used for the experiments. Wooden boards indicate the start location for the push task. The goal is the corner between the fixture and the box with the hole for the peg task.

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.

3 Related Work

3.1 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* [12] or "hybrid motions or tool operations" [13]. Other work has a broader skill definition [1, 2, 3, 4, 5, 14, 15]. 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 [16] skills are "high-level reusable robot capabilities, with the goal to reduce the complexity and time consumption of robot programming". However, compared to [3] and [14] they do not use pre- and postconditions. In [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 [1, 2, 4, 5, 14, 18, 16, 13] while [17] lacks such a capability.

In [16] 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.

3.2 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 [19, 20]. These choices also influence the form of priors that can be defined and how they are defined [6]. 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 [21], dynamic movement primitives [22, 23] and feed-forward neural networks [21, 24]. 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, [11] suggests to learn interpretable policies based on behavior trees [10]. They work explicitly in end-effector space and allow for an easy formulation of parameter priors to accelerate learning [25].

3.3 Planning and Learning

Symbolic planning is combined with learning in [26, 27, 28, 29]. In [26], 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. [27] uses the combined symbolic planner with reinforcement learning (RL) in a hierarchical framework to solve complex visual interactive question answering tasks. PEORL [28] 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 [29] 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 [30], the authors combine symbolic planning with behavior trees (BT) to solve blocks world tasks with a robot manipulator. They use modified Genetic Programming (GP) [31] 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.

4 Approach

Our approach consists of two main components that interact in different stages of the learning pipeline: First, *SkiROS* [14], 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 1.2: (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.

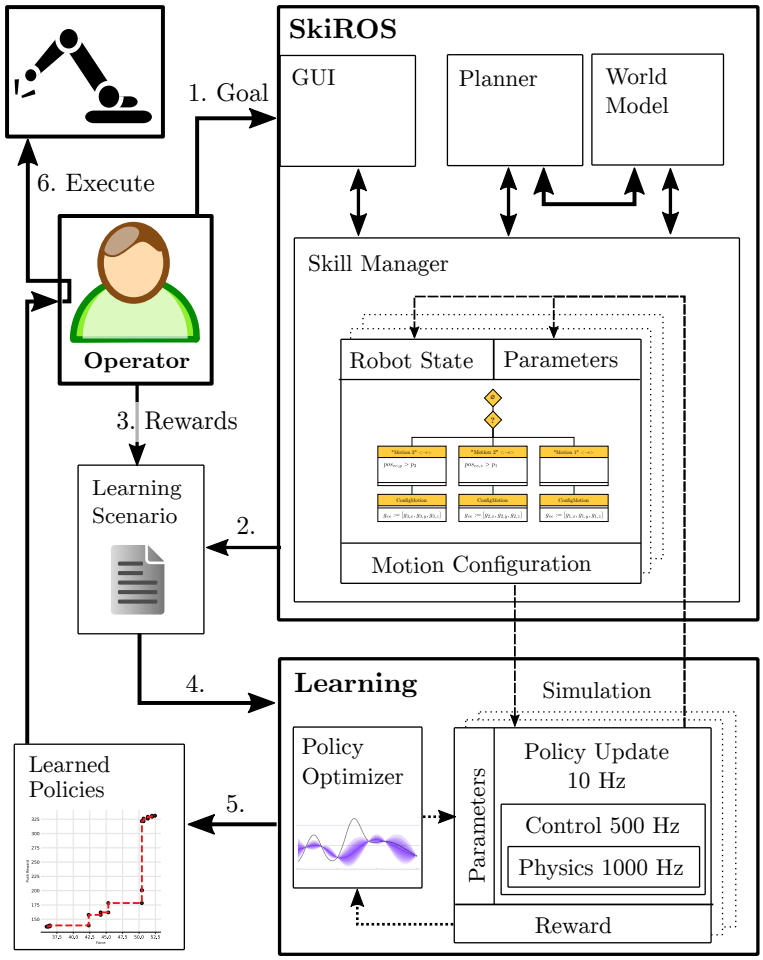


Figure 1.2: The architecture of the system that depicts the pipeline: (1) The operator enters the goal state; (2) a learning scenario for the plan is created; (3) rewards and hyperparameters are specified; (4) learning is conducted using the skills and the information in the world model; (5) after policy learning, the operator can choose which policies to execute on the real system (6).

4.1 Behavior Trees

A Behavior Tree (BT) [32] is a formalism for plan representation and execution. Like [18, 33], 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* [33]. 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 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* [18] 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. 1.3.

4.2 Planning and Knowledge Integration

The Planning Domain Definition Language (PDDL) [34, 4] is used to formulate the planning problem. We use the *SkiROS* [14] 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* [14] 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.

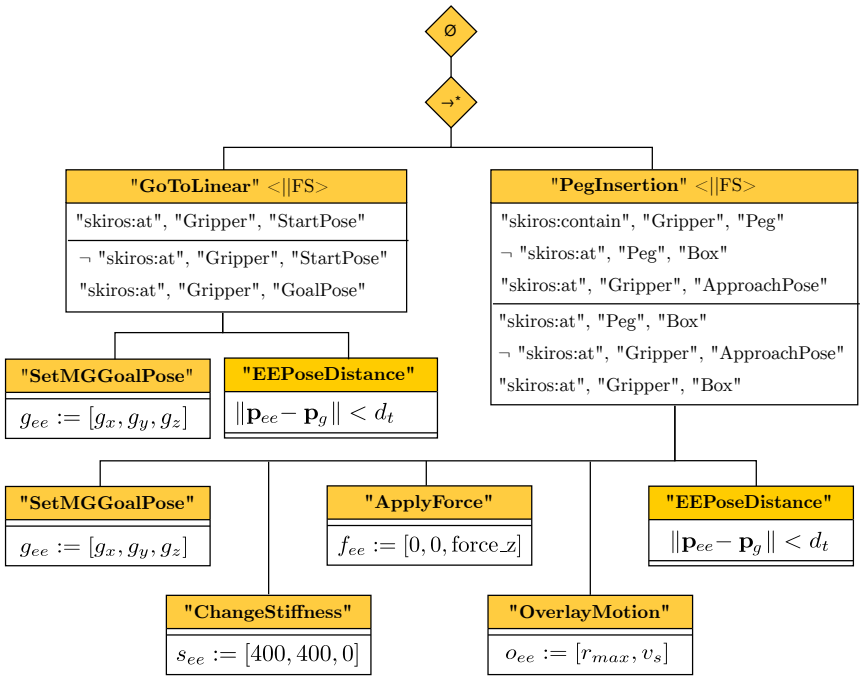


Figure 1.3: The BT of the generated plan for the peg insertion task in *eBT format* [18]. Each node has conditions or pre-conditions shown in the upper half and effects or post-conditions shown in the lower half. The *serial start control flow node* ($\rightarrow^*$) executes in a sequence and remembers the successes. The skills have a *parallel-first-success* processor ($< ||FS >$).

4.3 Policy Optimization

In order to optimize for policy parameters, we adopt the policy search formulation [21, 6, 24]. We formulate a dynamical system in the form:

$$\mathbf{x}_{t+1} = \mathbf{x}_t + M(\mathbf{x}_t, \mathbf{u}_t, \boldsymbol{\phi}_R), \tag{1.1}$$

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:

$$J(\boldsymbol{\theta}) = \mathbb{E}\left[\sum_{t=1}^{T} r(\mathbf{x}_t, \mathbf{u}_t)|\boldsymbol{\theta}\right], \tag{1.2}$$

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. (1.2) 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 [35]), but optimize for all objectives concurrently (Sec. 4.5) using Bayesian Optimization.

4.4 Bayesian Optimization

We consider the problem of finding a global minimizer (or maximizer) of an unknown (black-box) objective function f: $\mathbf{s}^* \in_{\mathbf{s}\in} f(\mathbf{s})$, where is some input design space of interest in D dimensions. The problem addressed in this paper is the optimization of a (possibly noisy) function $f :\rightarrow$ with lower and upper bounds on the problem variables. The variables defining can be real (continuous), integer, ordinal, and categorical as in [36]. 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) [37]. BO approximates $\mathbf{s}^*$ with a sequence of evaluations, $y_1, y_2, \ldots, y_t$ at $\mathbf{s}_1, \mathbf{s}_2, \ldots, \mathbf{s}_t \in$, 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 $\mathbf{s} \in$ 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 [38]. 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* [36, 39, 40, 41]. Our implementation selects the Expected Improvement (EI) acquisition function [42] and we use uniform random samples as a warm-up strategy before starting the optimization.

4.5 Multi-objective Optimization

Let us consider a multiple objectives minimization (or maximization) over in D dimensions. We define $f :\rightarrow \mathbb{R}^p$ 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 $\Gamma \subseteq$ of points which are not dominated by any other point, 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 : $\mathbf{s} \prec \mathbf{s}'$ iff $f(\mathbf{s}) \prec f(\mathbf{s}')$. The set of minimal points in this order is the Pareto-optimal set $\Gamma = \{\mathbf{s} \in : \nexists \mathbf{s}'$ 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 [43].

4.6 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 [10] and used in [11], 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 [10, 11]. E.g. an Archimedes spiral for search.

To make it compliant with the dynamical system in Eq. (1.1), 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. (1.1), 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.

5 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. 1.1: A) pushing an object with uneven weight distribution to a goal pose and B) inserting a peg in a hole with a 1.5mm larger radius. Pure planning-based solutions for both these tasks have a poor performance in reality (Fig. 1.5).

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. 4.6).

5.1 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:

1. *Task completion:* A fixed reward is assigned when the BT returns success upon task completion.
2. *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.
3. *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 ([44, 24]) $r(d_m) = \exp\left(-\frac{1}{2\sigma_w^2}(d_m + d_o)\right)$, where σ_w is a configurable width, d_o is a distance offset and d_m is the input metric.

4. *End-effector distance to a goal:* This reward uses distance between the end effectors current pose and goal pose to calculate the input metric $d_{ee,g} = \|\mathbf{p}_{ee,\mathbf{x}} - \mathbf{p}_g\|$.

5. *End-effector-reference-position distance:* This reward uses the distance between the end effectors reference pose (Sec. 4.6) and its current pose to calculate the input metric. $d_{ee,d} = \|\mathbf{p}_{ee,\mathbf{x}} - \mathbf{x_d}\|$

6. *Object-pose divergence:* This reward uses the translational and angular distance between the object's goal pose and its current pose.

5.2 Push Task

The push task starts by specifying the goal in the *SkiROS* Graphical User Interface (GUI) as: `(skiros:at skiros:ObjectToBePushed-1 skiros:ObjectGoalPose-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 0.07m and is an orthogonal triangle in the horizontal dimensions (x) and (y). It has a length of 0.15m and 0.3m and it weights 2.5kg. For this task we use a square-shaped peg for pushing with a side length of 0.07m and a height of 0.05m. Start and goal locations are $\approx$ 0.43m 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 0.01m and 5 deg, respectively.

We learn for 400 iterations and repeat the experiment ten times. In order to obtain solutions

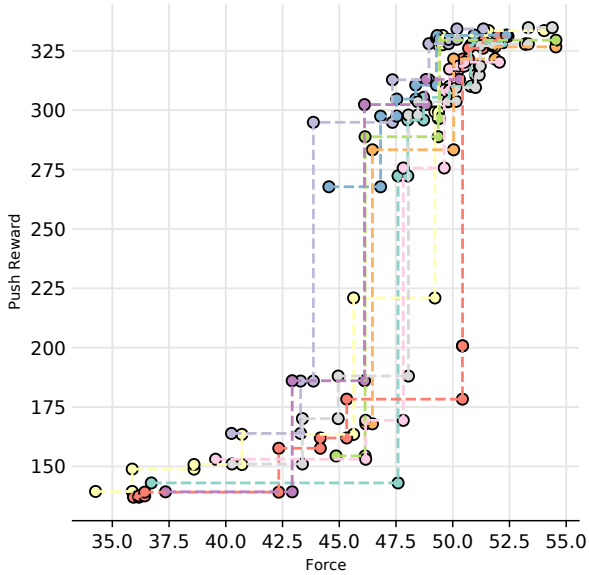


Figure 1.4: Pareto front of the push task. Each experiment has a different color and each point represents a Pareto-optimal solution. It shows that higher rewards for pushing require higher interaction forces with the environment.

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 7mm.

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. 1.5a.

The results of a multi-objective optimization are parameters found along a Pareto front (Sec. 4.5, see Fig. 1.4). 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. 1.5.

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 $\pm$ 64)min and executed (11.1 $\pm$ 30) 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%.

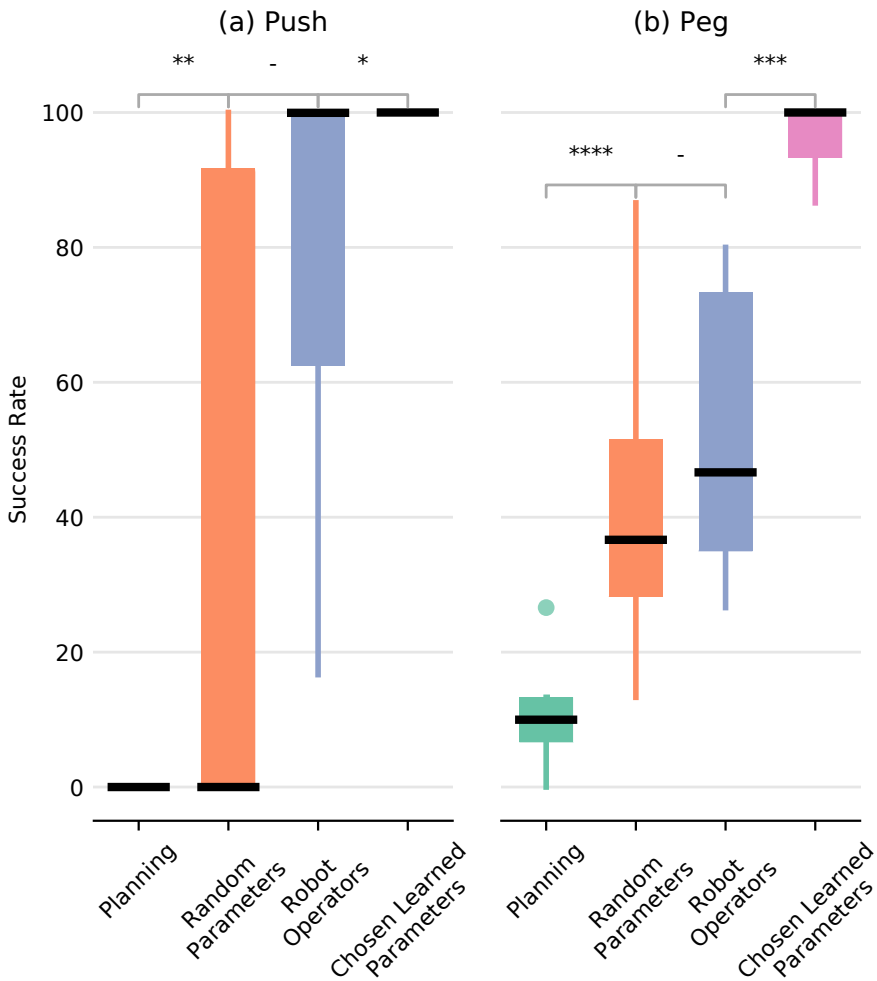


Figure 1.5: The success rates of both experiments. The box plots show the median (black line) and interquartile range (25^{th} and 75^{th} percentile); the lines extend to the most extreme data points not considered outliers, and outliers are plotted individually. The number of stars indicates that the p-value of the Mann-Whitney U test is less than 0.1, 0.05, 0.01 and 0.001 respectively.

5.3 Peg-in-Hole Task

The PDDL goal of the peg insertion task is `(skiros:atskiros:Peg-1skiros:BoxWithHole-1)`. The BT that is generated by the planner is shown in Fig. 1.3 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 [11]. The BT returns success only if the peg is inserted into the box hole by more than 0.01m.

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 7mm 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. 1.5b).

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. 1.5b).

To find policies for this task, the human operators took (31.8 $\pm$ 109)min and executed (39 $\pm$ 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. 1.5b. 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

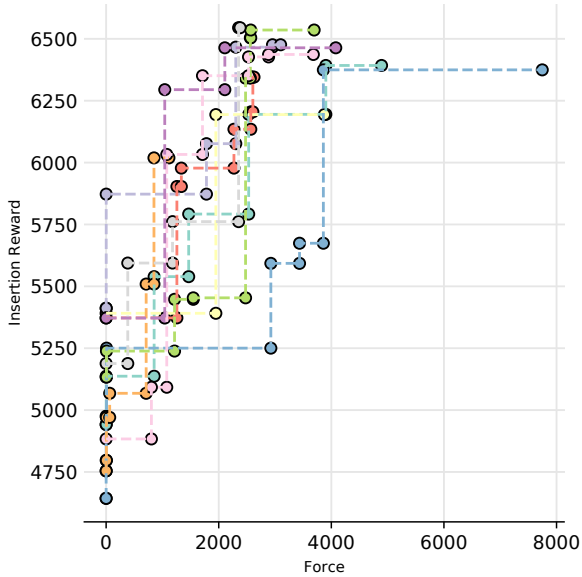


Figure 1.6: Pareto front of the peg task. Each experiment has a different color. The goal is to maximize insertion reward while minimizing the interaction forces.

18.1% faster. Therefore, the learned policies outperformed the human operators in both objectives while also producing more reliable results.

6 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 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`

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

[1] Volker Krueger et al. "Testing the vertical and cyber-physical integration of cognitive robots in manufacturing". In: *Robotics and Computer-Integrated Manufacturing* 57 (2019), pp. 213–229.

[2] Volker Krueger et al. "A Vertical and Cyber–Physical Integration of Cognitive Robots in Manufacturing". In: *Proceedings of the IEEE* 104.5 (2016), pp. 1114–1127.

[3] Simon Bøgh et al. "Does your robot have skills?" In: *Proceedings of the 43rd international symposium on robotics*. VDE Verlag GMBH. 2012.

[4] Matthew Crosby et al. "Integrating Mission and Task Planning in an Industrial Robotics Framework". In: *Proceedings of the Twenty-Seventh International Conference on Automated Planning and Scheduling (ICAPS 2017)*. AAAI. 2017.

[5] 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.

[6] Konstantinos Chatzilygeroudis et al. "A survey on policy search algorithms for learning robot controllers in a handful of trials". In: *IEEE Transactions on Robotics* 36.2 (2019), pp. 328–347.

[7] Mark Edmonds et al. “A tale of two explanations: Enhancing human trust by explaining robot behavior”. In: *Science Robotics* 4.37 (2019), eaay4663.

[8] Sylvain Koos, Jean-Baptiste Mouret and Stéphane Doncieux. “The transferability approach: Crossing the reality gap in evolutionary robotics”. In: *IEEE Transactions on Evolutionary Computation* 17.1 (2012), pp. 122–145.

[9] 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*. 2017, pp. 1121–1124.

[10] 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)*. 2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). 2018, pp. 5964–5971. DOI: 10.1109/IROS.2018.8594319.

[11] 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. 2021.

[12] 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.

[13] Ulrike 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. IEEE. 2003, pp. 3069–3075.

[14] Francesco Rovida et al. “SkiROS-A skill-based robot control platform on top of ROS”. In: *Studies in Computational Intelligence*. Vol. 707. 2017, pp. 121–160.

[15] Ulrike Thomas et al. “A new skill based robot programming language using uml/p statecharts”. In: *2013 IEEE International Conference on Robotics and Automation*. IEEE. 2013, pp. 461–466.

[16] Maj Stenmark et al. “On distributed knowledge bases for robotized small-batch assembly”. In: *IEEE Transactions on Automation Science and Engineering* 12.2 (2015), pp. 519–528.

[17] C. Paxton et al. “CoSTAR: Instructing collaborative robots with behavior trees and vision”. In: *2017 IEEE International Conference on Robotics and Automation (ICRA)*. 2017, pp. 564–571. DOI: 10.1109/ICRA.2017.7989070.

[18] Francesco Rovida, Bjarne Grossmann and Volker Krüger. “Extended behavior trees for quick definition of flexible robotic tasks”. In: *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2017, pp. 6793–6800.

[19] 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. DOI: 10.1109/IROS40897.2019.8967946.

[20] 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)*. 2019, pp. 1010–1017. DOI: 10.1109/IROS40897.2019.8968201.

[21] Marc Peter Deisenroth, Gerhard Neumann and Jan Peters. "A Survey on Policy Search for Robotics". In: *Foundations and Trends® in Robotics* 2.1–2 (2013), pp. 1–142. ISSN: 1935-8253, 1935-8261. DOI: 10.1561/2300000021. URL: https://www.nowpublishers.com/article/Details/ROB-021 (visited on 26/03/2019).

[22] Auke Jan Ijspeert et al. "Dynamical movement primitives: learning attractor models for motor behaviors". In: *Neural computation* 25.2 (2013), pp. 328–373.

[23] Aleš Ude et al. "Task-specific generalization of discrete and periodic dynamic movement primitives". In: *IEEE Transactions on Robotics* 26.5 (2010), pp. 800–815.

[24] Konstantinos Chatzilygeroudis et al. "Black-Box Data-Efficient Policy Search for Robotics". In: *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). 2017, pp. 51–58. DOI: 10.1109/IROS.2017.8202137.

[25] Matthias Mayr et al. "Learning Skill-based Industrial Robot Tasks with User Priors". In: *IEEE International Conference on Automation Science and Engineering (CASE)*. IEEE. 2022.

[26] Matthew Grounds and Daniel Kudenko. "Combining reinforcement learning with symbolic planning". In: *Adaptive Agents and Multi-Agent Systems III. Adaptation and Multi-Agent Learning*. Springer, 2005, pp. 75–86.

[27] Daniel Gordon, Dieter Fox and Ali Farhadi. "What should i do now? marrying reinforcement learning and symbolic planning". In: *arXiv preprint arXiv:1901.01492* (2019).

[28] Fangkai Yang et al. "Peorl: Integrating symbolic planning and hierarchical reinforcement learning for robust decision-making". In: *arXiv preprint arXiv:1804.07779* (2018).

[29] 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)*. 2021.

[30] Jonathan Styrud et al. "Combining Planning and Learning of Behavior Trees for Robotic Assembly". In: *arXiv preprint arXiv:2103.09036* (2021).

[31] John R Koza and John R Koza. *Genetic programming: on the programming of computers by means of natural selection*. Vol. 1. MIT press, 1992.

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

[33] A. Marzinotto et al. "Towards a Unified Behavior Trees Framework for Robot Control". In: *2014 IEEE International Conference on Robotics and Automation (ICRA)*. 2014 IEEE International Conference on Robotics and Automation (ICRA). 2014, pp. 5420–5427. DOI: 10.1109/ICRA.2014.6907656.

[34] Maria Fox and Derek Long. "PDDL2. 1: An extension to PDDL for expressing temporal planning domains". In: *Journal of artificial intelligence research* 20 (2003), pp. 61–124.

[35] Rituraj Kaushik, Konstantinos Chatzilygeroudis and Jean-Baptiste Mouret. "Multi-objective model-based policy search for data-efficient learning with sparse rewards". In: *Conference on Robot Learning*. PMLR. 2018, pp. 839–855.

[36] Luigi Nardi, David Koeplinger and Kunle Olukotun. "Practical Design Space Exploration". In: *International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems*. 2019.

[37] Peter I. Frazier. *A Tutorial on Bayesian Optimization*. 2018. arXiv: 1807.02811 [stat.ML].

[38] 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". In: *arXiv preprint arXiv:1012.2599* (2010).

[39] Luigi Nardi et al. "Algorithmic performance-accuracy trade-off in 3d vision applications using hypermapper". In: *International Parallel and Distributed Processing Symposium Workshops*. 2017.

[40] 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.

[41] 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. ISBN: 978-3-030-86523-8.

[42] 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.

[43] Biswajit Paria, Kirthevasan Kandasamy and Barnabás Póczos. "A Flexible Framework for Multi-Objective Bayesian Optimization using Random Scalarizations". In: *Proceedings of the Thirty-Fifth Conference on Uncertainty in Artificial Intelligence, UAI*. Ed. by Amir Globerson and Ricardo Silva. 2019, p. 267.

[44] 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. ISBN: 978-1-4503-0619-5.

Paper II

F. Ahmad, M. Mayr, E.A. Topp, J. Malec, V. Krueger
Generalizing Behavior Trees and Motion-Generator (BTMG) Policy Representation for Robotic Tasks Over Scenario Parameters
IEEE International Joint Conferences on Artificial Intelligence (IJCAI), Vienna, Austria, 2022, Workshop on Bridging the Gap Between AI Planning and Reinforcement Learning

Chapter 2

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

1 Abstract

We propose a generalization of a behaviour tree and motion-generator based robot arm policy representation for learning and solving tasks such as contact-rich tasks like peg insertion or pushing an object. We use planning to generate skill sequences needed to execute these tasks and rely on reinforcement learning to obtain parameters of the policy. We assume gaussian processes as a suitable method for this generalization and present preliminary, promising results from initial experiments.

2 Introduction

In previous papers [1, 2, 3] we have developed a representation based on behavior trees (BT) [4] and motion-generator (MG), (BTMG). [1]. They are easy to interpret, can be **robust** to faults and errors that can occur during execution and they can be **reactive**, allowing the robot to act and deal with uncertain conditions and recover from failures.

BTMG is a parameteric policy representation that allow us to solve contact-rich tasks like

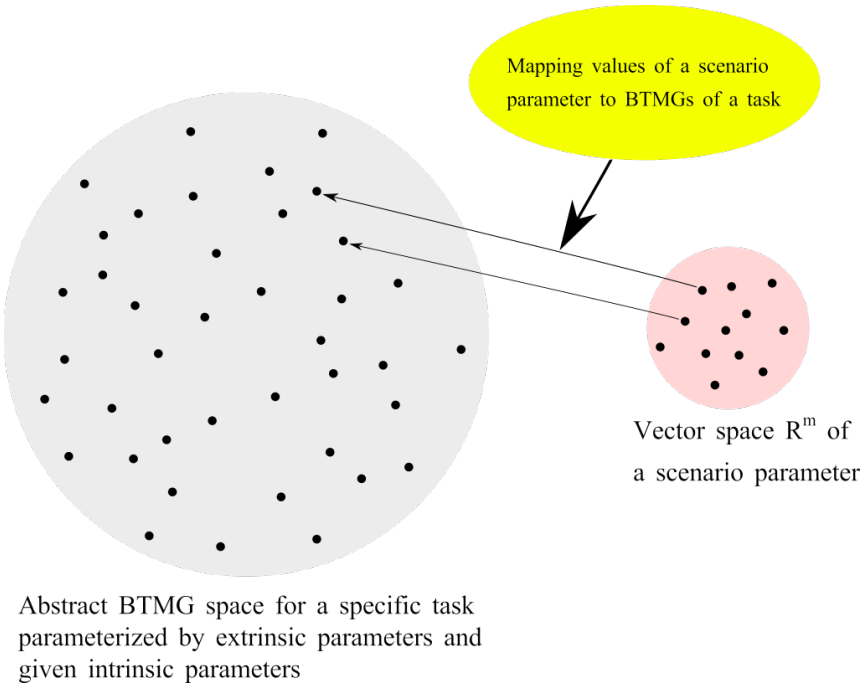


Figure 2.1: *The mapping (yellow) of the vector space of a scenario parameter (pink) to the abstract BTMG space of a task (gray).*

"peg-in-hole" or pushing an object. Parameters of a BTMG can vary from deciding the structure of behavior trees to specifying the actual controller stiffness values of the MG. We can either specify these parameters manually [1] or learn them using reinforcement learning (RL) [2, 3]. We generate skill sequences for these tasks in a skill-based system, SkiROS [5] that uses Planning Domain Definition Langauage (PDDL) for task planning.

Although BTMGs are shown to be quite promising, one key shortcoming of this representation is that they can be scenario specific. For instance, pushing an object to different goal locations using a BTMG requires learning the parameters. This is problematic and is also common for the original formalization of dynamic motion primitives (DMPs) [6]. This problem was later resolved by generalizing DMPs [7]. In this work, we aim to generalize the BTMG policy representation in a similar way. Our proposed solution is to generate a novel BTMG for a new task instance as a weighted sum of the "basis"-BTMG (parametric BTMG for different instances of a task). This poses some interesting questions: 1) What kind of Basis-BTMGs should we use? 2) How many Basis-BTMGs do we need? 3) How can we use Basis-BTMGs to interpolate?

3 Formalization

A BTMG is parameterized by two types of parameters; **intrinsic parameters** and **extrinsic parameters**. Intrinsic parameters decide the structure of the BT, number of control flow and execution nodes, etc. These parameters also decide how much velocity can be allowed, how fast the arm should move, etc. Intrinsic parameters could be implied by the specific task at hand, e.g., push task and peg-in-a-hole task. In this work, we do not want to change intrinsic parameters. Extrinsic parameters on the other hand represent, e.g., how much force can be applied, offsets, path velocity of the end-effector, etc. In a nutshell, extrinsic parameters are optimized while intrinsic ones are assumed to be known apriori. Note that object goal pose is not necessarily a parameter here. Consider a pushing task: while the object goal pose represents the centre of mass of the object at the goal location, the point on the object where the peg touches the object is expected to be different. The centre of mass of the object should be at the goal location. We specify pushing the object through a push vector(see Figure 2.2 defined by start and goal offsets from object start and goal locations.

In previous work [3], we have used RL to learn extrinsic parameters of the BTMG policy representation for a specific instance of a push task and a peg insertion task.

Apart from BTMGs, we also consider scenario parameters like object goal pose, object start pose, object weight, etc. These parameters represent variations or dimensions over which we want to generalize the BTMG representation of a task. Figure 2.1 shows the vector space $\mathbb{R}^m$ of a particular scenario parameter (shown in pink). Every point in this space shows a set of unique values of scenario parameters. For instance, any point or object goal pose would be a 6D-vector $x, y, z, \alpha, \beta, \gamma$ representing the goal pose of the object.

4 Mapping

In order to generate various policy representations of a task that generalizes over a scenario parameters, we are interested in a mapping that maps scenario parameters to the corresponding extrinsic BTMG parameters for a task. Figure 2.1 shows the mapping (yellow) that maps the vector space of a scenario parameter to BTMGs of a specific task.

We propose to use gaussian processes (GP) [8, 9, 10] as a mapping function. We start by collecting data samples using RL by learning extrinsic parameters for BTMGs of a task over particular scenario parameters. We use these samples to train the GP by using scenario parameters as input and extrinsic parameters as output. The idea is to then use this trained GP to interpolate and return the values of extrinsic parameters for different values of the scenario parameters.

We also want to clarify that using GP to interpolate is not a new idea as it has already been used in literature [11, 8]. The novelty of our approach lies in using GPs as a mapping function in the context of BTMG policy representation that allows it to generalize over scenario parameters.

Using GP has two major benefits: 1) It provides mean and variance bounds over the extrinsic parameters of BTMG of a skill. 2) They are known for generalizing over domains and have been used in this context in Dynamic Motion Primitives (DMPs) as discussed before.

5 Experiments

We tested our approach on a push task where the robot had to push an object from a start location to specified object goal poses, see Figure 2.2. In our setting, the BTMG of the push task has four learnable extrinsic parameters: 1) Offsets in start locations s_x, s_y, 2) Offsets in goal locations g_x, g_y. Together, these parameters decide the push vector. We start by collecting training and testing samples of goal locations of the object. Instead of randomly choosing samples over the space we use Latin hypercube space[12] to achieve evenly distributed samples across the entire region. We choose defined number of samples within the bounds. Choosing the number of samples is not a trivial task and dependent on work space and the type of the scenario parameter.

We use RL to get the best s_x, s_y, g_x, g_y for every training sample. These are used to train our GP. The GP takes object goal pose as input and produces offsets s_x, s_y, g_x, g_y as output. For simplicity, we only change the x and y coordinates of the object goal pose. The trained GP is then used to generate offsets for the test points.

We trained the GP on samples distributed across a restrictive space and tested it on unseen samples. The initial results look promising as the GP was able to find offsets that managed to solve the task for all the test points. The offsets managed to push the object throughout without slipping off. We analyzed the performance of offsets by calculating the error between actual and specified goal pose of the object. Initial results suggest that the error for the offsets obtained thorough GP is in the same range for the offsets learned through RL.

6 Future Work

For future work, we are planning to generalize over a larger space to obtain BTMG parameters for multiple scenario parameters together. We would also like to extend this generalization to other tasks.

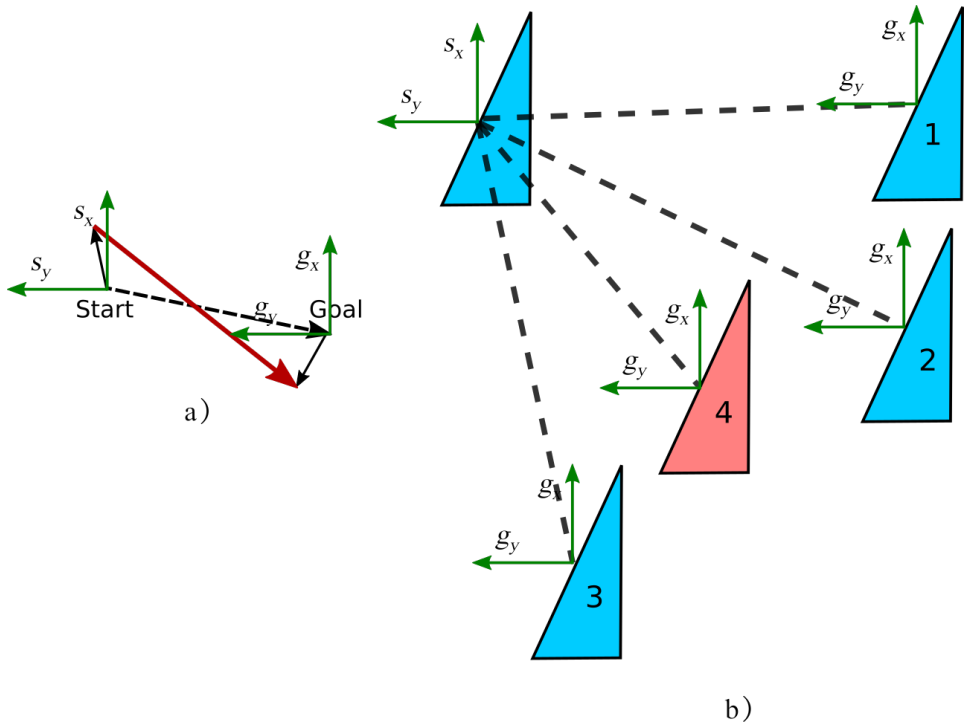


Figure 2.2: *a) Shows the push vector (red) defined by offsets in start and goal location. The image b) shows the push task setup with different goal locations where (1-3) are training examples and (4) is a test example.*

We would also like to evaluate the performance of trained GP models for a skill by comparing it with baseline regression models. Since, we aim to use a generic model, we expect GP to perform well to generalize scenario parameters of BTMG of difference skills.

We would also like to investigate sensitivity of the model to different scenario parameters. Basically how well GP performs for different types of scenario parameters? Furthermore, we would also like to see if GP can be used to interpolate intrinsic parameters as well i.e. generating new BTs for a skill.

References

[1] 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)*. 2018 IEEE/RSJ International Conference on Intelligent

Robots and Systems (IROS). 2018, pp. 5964–5971. DOI: 10.1109/IROS.2018.8594319.

[2] 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. 2021.

[3] Matthias Mayr et al. “Skill-based Multi-objective Reinforcement Learning of Industrial Robot Tasks with Planning and Knowledge Integration”. In: *arXiv preprint arXiv:2203.10033* (2022).

[4] 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*. 2014 IEEE/RSJ International Conference on Intelligent Robots and Systems. 2014, pp. 1482–1488. DOI: 10.1109/IROS.2014.6942752.

[5] Francesco Rovida et al. “SkiROS-A skill-based robot control platform on top of ROS”. In: *Studies in Computational Intelligence*. Vol. 707. 2017, pp. 121–160.

[6] Auke Jan Ijspeert, Jun Nakanishi and Stefan Schaal. “Learning rhythmic movements by demonstration using nonlinear oscillators”. In: *Proceedings of the ieee/rsj int. conference on intelligent robots and systems (iros2002)*. CONF. 2002, pp. 958–963.

[7] Auke Jan Ijspeert et al. “Dynamical movement primitives: learning attractor models for motor behaviors”. In: *Neural computation* 25.2 (2013), pp. 328–373.

[8] Carl Edward Rasmussen. *Gaussian processes in machine learning*. Springer Science & Business Media, 2003.

[9] 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* (2011).

[10] You Zhou and Tamim Asfour. “Task-oriented generalization of dynamic movement primitive”. In: *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (2017).

[11] Denis Forte et al. “On-line motion synthesis and adaptation using a trajectory database”. In: *Robotics and Autonomous Systems* 60.10 (2012), pp. 1327–1339.

[12] K.Q. Ye. “Orthogonal column Latin hypercubes and their application in computer experiments”. In: *Journal of the American Statistical Association* (1998).

Paper III

F. Ahmad, M. Mayr, V. Krueger
Learning to Adapt the Parameters of Behavior Trees and Motion Generators (BTMGs) to Task Variations
IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Detroit, MI, USA, 2023, pp. 10133-10140, doi: 10.1109/IROS55552.2023.10341636

Chapter 3

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

1 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.

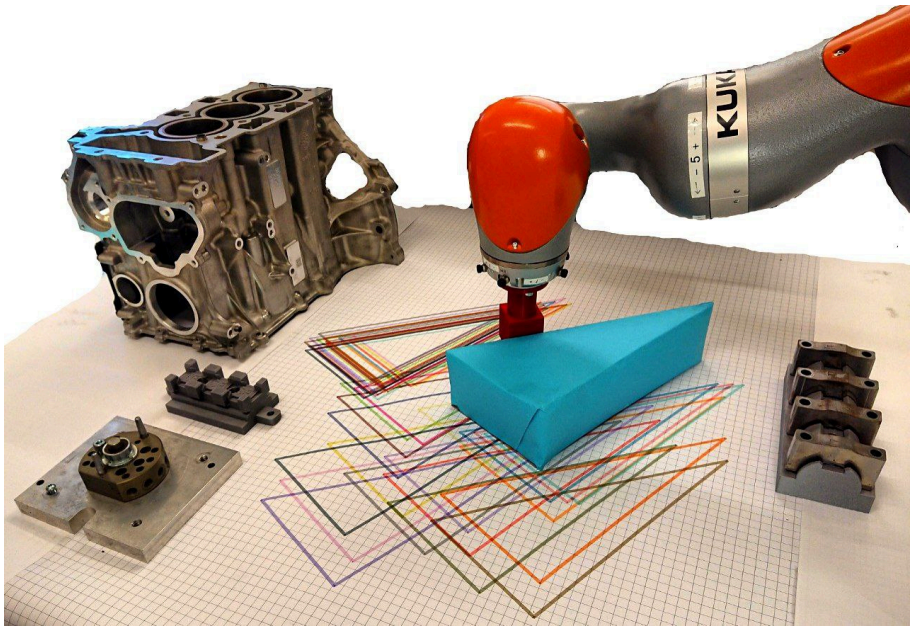


Figure 3.1: The experimental setup. It shows the object with the skewed weight distribution that is pushed with a 45mm wide peg. On the table the different start and goal positions for the object can be seen in different colours. On the sides, some example sizes for obstacles are shown.

2 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. [1] have suggested a representation that combines behavior trees (BT) [2, 3] 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 [1, 4] and pushing an object to a target location [5, 6].

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 [1]. 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 [1]. Furthermore, they have the ability to be **reactive** [2], 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 [1, 4, 7].

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 [1]. 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) [5, 6, 8]. 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) [9, 10] and was later addressed in [11].

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 vari-

ations.

- 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.

3 Related Work

Movement primitives, based on motor primitives theory [12, 13], are mathematical formulations of dynamic systems that generate motions. Two well-known movement primitives used in robotics are Dynamic Movement Primitives (DMPs) [9, 10] and Probabilistic Movement Primitives (ProMPs)[14]. Movement primitives can be generalized and have proven successful in various robotics applications, such as dynamic motion primitives [9, 10]. 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 [11].

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 [15, 16, 17, 18], and can generalize to new goals by interpolating weights of neighboring DMPs [19] or by using Gaussian Process Regression (GPR) to generate new parameters [20]. Furthermore, GPs [21] have been used to generalize DMPs to external task variations, arbitrary movements, and adapting trajectories to new situations online in [22, 23, 20], respectively. In [24], 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 [25, 26], 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 [27, 28] and solving Human-Robot collaborative tasks [29] 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 [20, 22, 23]. 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.

4 BTMG and Task Variations

We define BTMG as a parametric policy representation, BTMG(θ) where $\theta \in \mathbb{R}^N$. The parameters θ 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 θ_i and **extrinsic** parameters θ_e [30].

Intrinsic parameters θ_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 θ_i. These parameters are fixed and independent of the task instance, meaning that T_p uses the same θ_i values regardless of the starting position, or the target position of the object. In other words, θ_i is situation-invariant. Within the scope of this paper, these parameters are assumed to be known a priori.

Extrinsic parameters θ_e are situation dependent e.g. to determine the applied force, offsets, and the velocity of the end effector. Again, θ_e can be specified manually [1, 31], inferred through a reasoning framework, or learned using RL. We have already demonstrated how RL can be used to obtain BTMG parameters [4] and used it in simulation and on a real robot to solve multi-objective tasks [6, 8].

In addition to θ, we also consider task variations $v \in \mathbb{R}^M$. 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 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 3.2). We take two task variations $v_1 = (v_{s_x}, v_{s_y}, v_{g1_x}, v_{g1_y})$ and $v_2 = (v_{s_x}, v_{s_y}, v_{g2_x}, v_{g2_y})$ that define the start and goal positions of the object. For variations v_1 and v_2, we have corresponding $\theta_{e1} = (\theta_{e1_{s_x}}, \theta_{e1_{s_y}}, \theta_{e1_{g_x}}, \theta_{e1_{g_y}})$ and $\theta_{e2} = (\theta_{e2_{s_x}}, \theta_{e2_{s_y}}, \theta_{e2_{g_x}}, \theta_{e2_{g_y}})$ that collectively define the start and the goal locations for the pushing action.

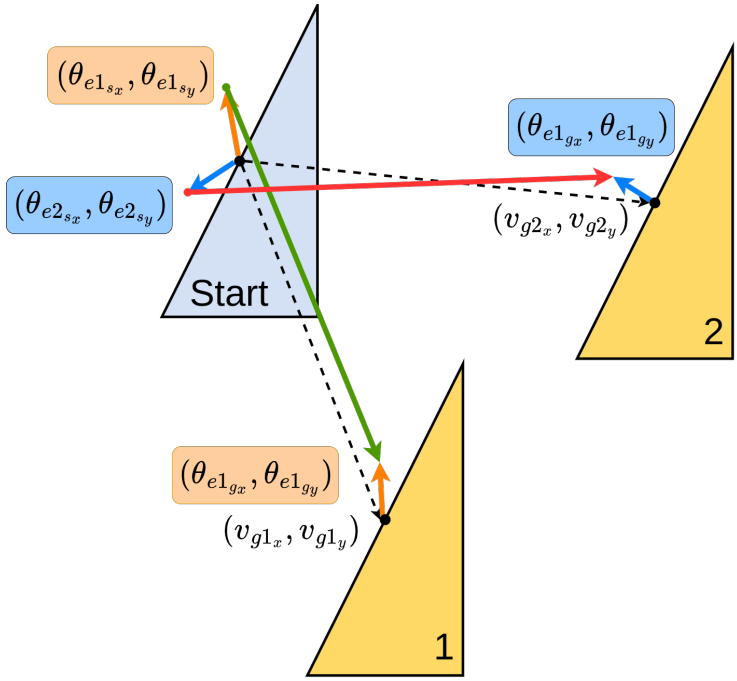


Figure 3.2: An illustration of two simplified task variations v_1 and v_2 in the pushing task that only vary the goal location. The orange and blue vectors are set by the respective learned extrinsic parameters θ_{e1} and θ_{e2}, so that they define the resulting green and red push vectors that should successfully push the object.

As θ_i has no impact on adapting BTMGs to different variations, our objective in this paper is to establish a relationship between θ_e and v that would enable the adaptation of BTMGs to new variations.

5 Approach

In this section, we explain how we adapt BTMG parameters for a new task variation by using the *PerF* model. Figure 3.3 shows how the *PerF* model works in comparison with a *direct* model. The overall approach is divided into the training (Sec. 5.1) and query phase (Sec. 5.2). In the training phase, we pass each task variation $v_k \in \mathbb{V}_{train}$, into an extended RL pipeline similar to [6]. 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:

1. *Best policies*: For every task variation we get the best performing policy:

$$\mathbb{T} = \{(v_k, \theta^*_{e,v_k}) | k = 1, \ldots, n\}$$

2. *All evaluated configurations and their rewards:*
$\mathbb{K} = \{(v_k, \theta_{ei,v_k}, r_{\theta_{ei,v_k}}) | k = 1, \ldots, n \text{ and } i = 1, \ldots, t \leq t_{\max}\}$

3. *All evaluated configurations and their feasibility:*
$\mathbb{E} = \{(v_k, \theta_{ei,v_k}, f_{\theta_{ei,v_k}}) | k = 1, \ldots, n \text{ and } i = 1, \ldots, t \leq t_{\max}\}$

The *direct* model M is trained with the set $\mathbb{T}$ and, as a result, learns to predict $\hat{\theta}_e$ given 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 θ_e. The model further uses $\hat{r}$ and $\hat{f}$ to generate a surrogate reward function that obtains $\hat{\theta}_e$ given v. For more details on how we obtain set $\mathbb{T}$, we direct the reader to [6]. To obtain sets $\mathbb{K}$ and $\mathbb{E}$, we follow the same procedure as in [6], 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 θ_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.

5.1 Training Phase

We frame the mapping of the task variations v to the extrinsic BTMG parameters θ_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 3.3. We propose to use GP and weighted SVM to learn $\hat{J} : (\theta_e, v) \mapsto \hat{r} \in \mathbb{R}$ and $\hat{F} : (\theta_e, v) \mapsto \hat{f} \in \{0, 1\}$. $\hat{J}$ and $\hat{F}$ are trained by data points in sets $\mathbb{K}$ and $\mathbb{E}$, provided by the RL pipeline introduced in [4].

For each task variation, $v_k \in \mathbb{V}_{train}$, similar to [4, 6], we define $J_{v_k}(\theta_e)$ as the expected sum of individual rewards over time, given a sequence of extrinsic parameters $\theta_{e1}, \theta_{e2}, \ldots, \theta_{et} \in \theta_e$.

In [4, 6], we use Bayesian optimization (BO) as a black-box optimization method to obtain the optimal policy parameters θ^*_e and the best reward $J_{v_k}(\theta^*_e)$. In this paper, however, we use BO to obtain $J_{v_k}(\theta_e)$ by computing $J_{v_k}(\theta_{e1}), J_{v_k}(\theta_{e2}), \ldots, J_{v_k}(\theta_{et})$ over the sequence $\theta_{e1}, \theta_{e2}, \ldots, \theta_{et}$. This allows us to not only have the optimal policy parameters θ^*_e and the corresponding best reward $J_{v_k}(\theta^*_e)$ but it also provides us with intermediate θ_{et} and $J_{v_k}(\theta_{et})$. 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 v_k, we define the feasibility function $F_{v_k}(\theta_e)$ 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 $J_{v_k}(\theta_e)$, we obtain $F_{v_k}(\theta_e)$ by computing $F_{v_k}(\theta_{e1}), F_{v_k}(\theta_{e2}), \ldots,$ $F_{v_k}(\theta_{et})$ for the sequence of evaluations $\theta_{e1}, \theta_{e2}, \ldots, \theta_{et}$. For more details about the pipeline, we refer the reader to the policy optimization section in [4, 6].

To model $\hat{J}$ and $\hat{F}$, we obtain a sequence of BTMG parameter vectors, $\theta_{e1}, \theta_{e2}, \ldots, \theta_{et}$, along with their corresponding reward values $J_{v_k}(\theta_{e1}), J_{v_k}(\theta_{e2}), \ldots, J_{v_k}(\theta_{et})$ and feasibility values $F_{v_k}(\theta_{e1}), F_{v_k}(\theta_{e2}), \ldots, F_{v_k}(\theta_{et})$ 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.

5.2 Query Phase

The goal of this phase is to query the trained model with a new task variation $v_p \in \mathbb{V}_{test}$ and obtain a $\hat{\theta}_e$ by optimizing $\hat{J}(\theta_{et}|v_p)$ under the feasibility constraint $\hat{F}(\theta_{et}|v_p)$ (Figure 3.3). 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 θ_e for a new v_p.

We begin the optimization process by specifying the optimizer type, the bounds for θ_e, and the maximum number of iterations $t_{\max}$. In our experiments, we used the Limited-memory Broyden–Fletcher–Goldfarb–Shanno (L-BFGS) [32, 33] algorithm, which refines an initial estimate of θ_{e1} to iteratively obtain improved evaluation points θ_{et}, where $t \leq t_{\max}$, using the derivative as the driving function. For each new task variation v_k, we run the optimizer to obtain a sequence of evaluation points θ_{et}.

Using $\hat{J}$ and $\hat{F}$, we define a surrogate reward $r_{v_p} = \hat{r}_{\theta_{et},v_p} - (1 - \hat{f}_{\theta_{et},v_p})) * \mu$. Here, the first term corresponds to the output reward value computed by $\hat{J}$, while the second term penalizes the reward if $\hat{f}_{\theta_{et},v_p}$ maps to 0. We penalize the reward $\hat{r}_{\theta_{ei},v_p}$ by a small factor μ. We query the surrogate reward r_{v_p} for defined number of iterations or until the optimizer converges.

After the optimization phase, we select the θ_{et} that maximizes both $\hat{J}(\theta_{et}|v_p)$ and is feasible

$\hat{F}(\theta_{et}|v_p)$.

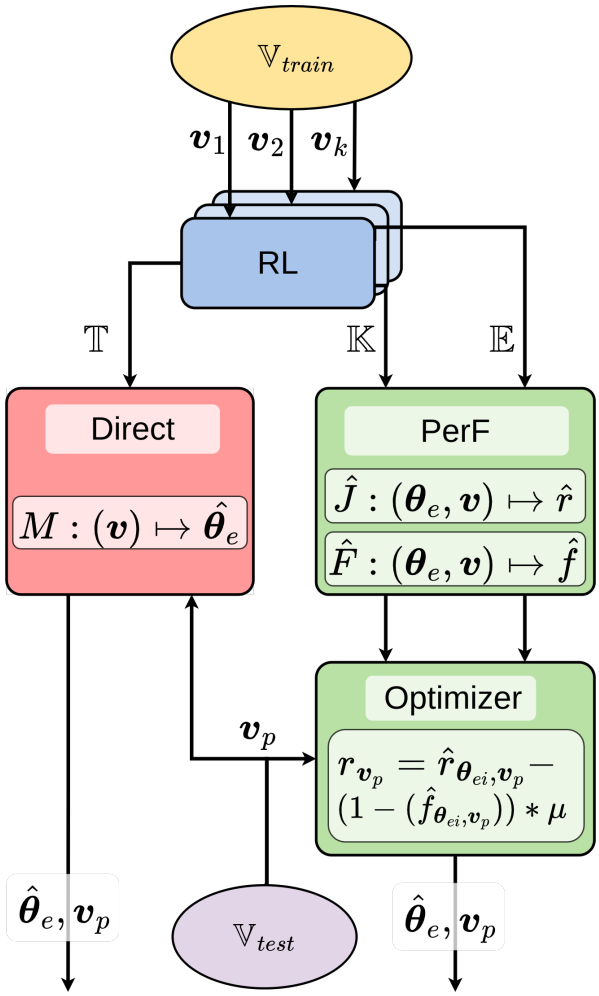


Figure 3.3: The pipeline of our approach and the *direct* model baseline. For every task variation $\mathbf{v}$, an RL problem is solved and the respective results are provided to the GP models. When querying for a new task variation $\mathbf{v}_p$ both models are queried for a set of extrinsic parameters $\hat{\theta}_e$.

6 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 obstacle avoidance task

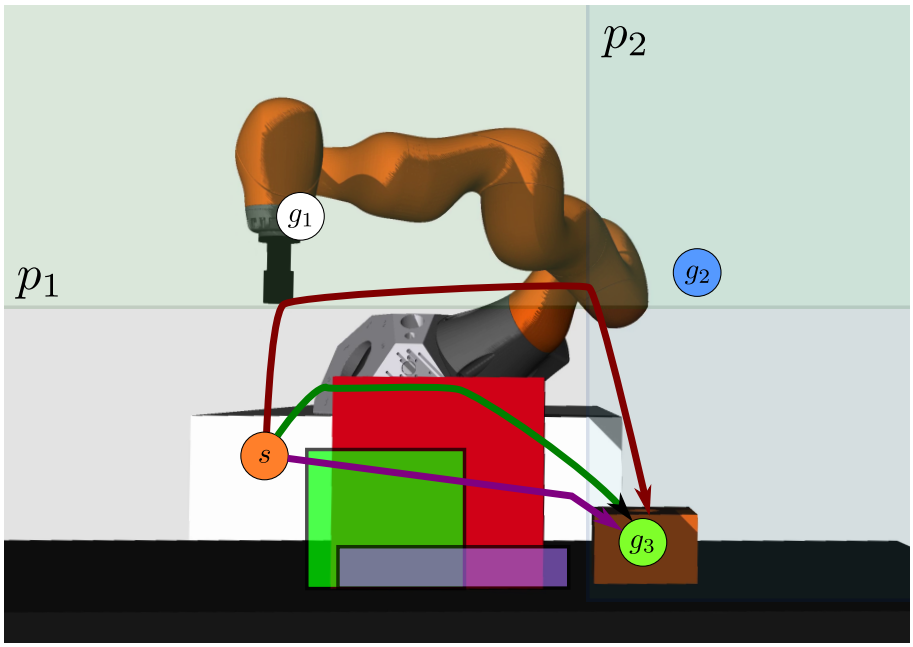


Figure 3.4: The obstacle task with some of the variations of the object location, width, and height. For each object configuration, valid example trajectories are shown in the same color. For the red trajectory, the intermediate goal points ($\mathbf{g}_1$ and $\mathbf{g}_2$) and two motion switching thresholds (p_1 and p_2) are shown.

and a pushing task, each having its own challenges. For simulation, we utilized the *DART* simulation toolkit [24] and in both simulation and reality, the robot arm was controlled using a Cartesian impedance controller [34], 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 [4].

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. 5.1, 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 [6] 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.

We consider task-specific reward functions for both tasks. The rewards and feasibility measures for the tasks are defined separately in their respective sections.

6.1 Obstacle Avoidance Task

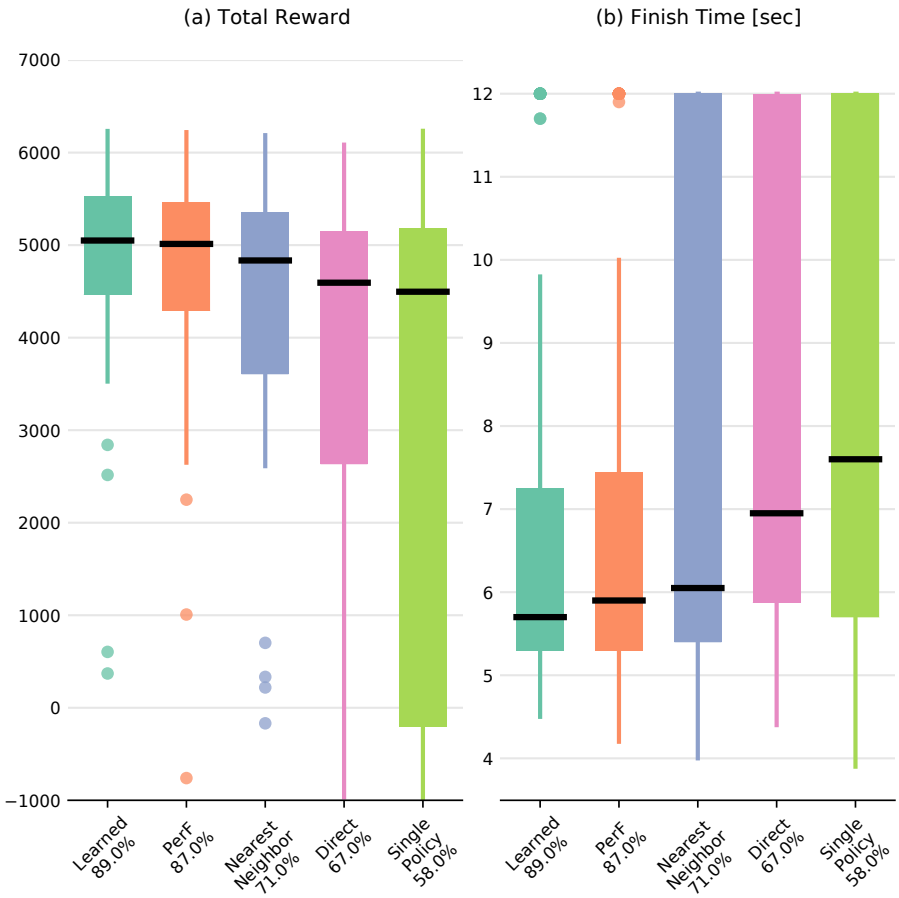
The objective of the obstacle avoidance task is to move the robot's end effector from the start to the goal location while avoiding an obstacle in the workspace. As shown in Fig. 3.4, the obstacle can vary in size and position. The goal is to find policies that navigate the robot around the obstacle 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. 3.4). The obstacle varies in height from 0.049m to 0.331m and in width from 0.09m to 0.331m. The horizontal position ranges from 0.274m to 0.311m 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 40mm. 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 [4, 6]. 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. 3.4.



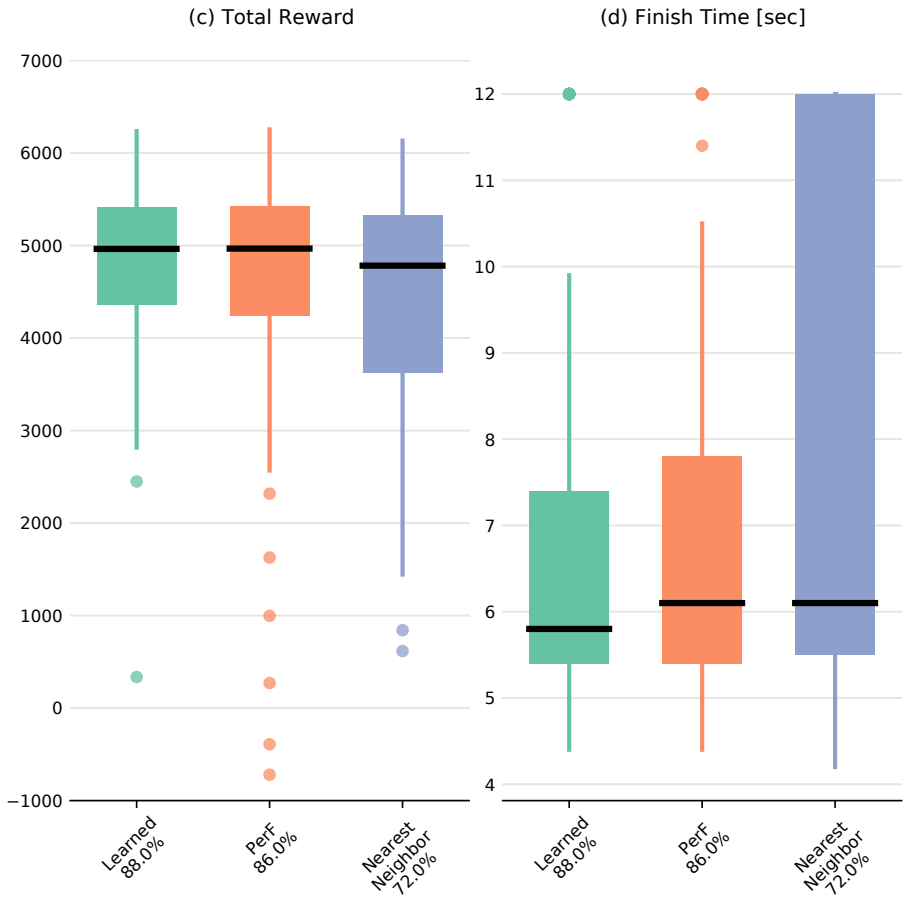


Figure 3.5: The total reward (a, c) and the execution time (b, d) of the obstacle task in simulation (a, b) and on the real system (c, d). The box plots show the median (black line) and interquartile range (25^{th} and 75^{th} percentile); the lines extend to the most extreme data points not considered outliers, and outliers are plotted individually. The success percentages are shown below the method names.

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. 3.5a) and b) and Table 3.1. 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 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. 3.5c) and d) have only minor variations from the simulation results. This also demonstrates the robustness of this policy formulation as a whole.

6.2 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. 3.1 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 0.16m around a center point. For the goal position, a triangular-shaped region is used. Fig. 3.1 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.

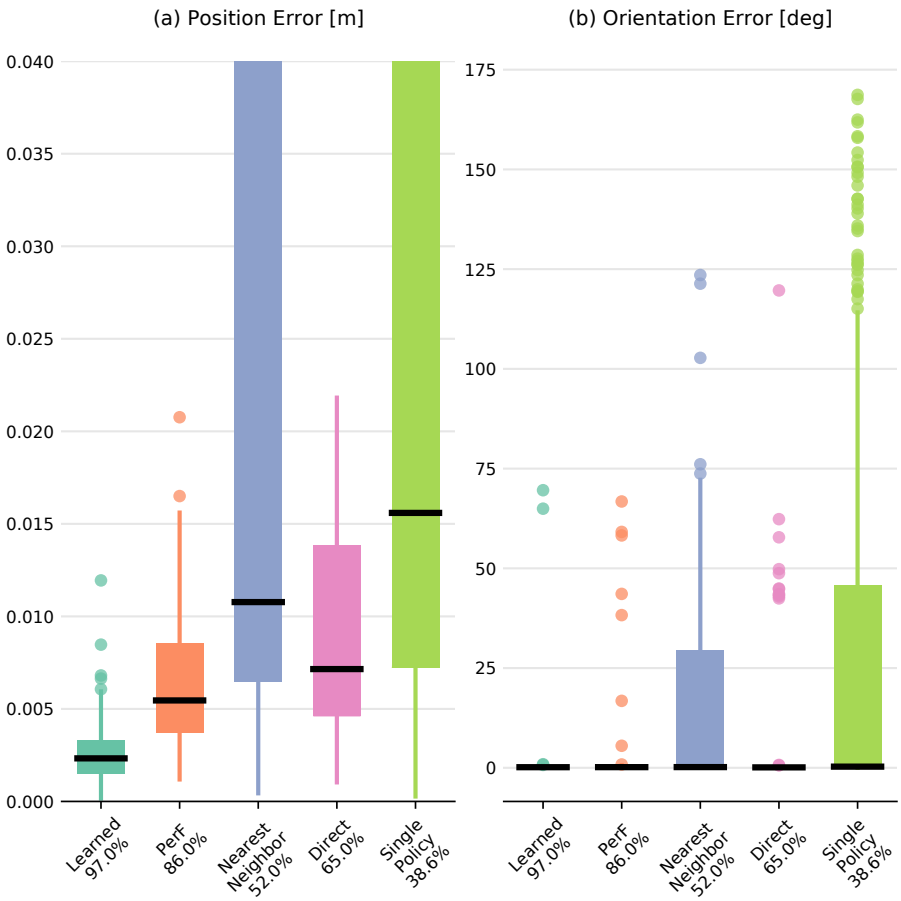
Similarly to previous work [5, 6], the push task has four BTMG parameters that are learned. They are depicted in Fig. 3.2. These parameters control additional start and goal offsets in the horizontal directions (x, y), determining the shape of the push vector that is indicated in Fig. 3.2. The start and goal orientation of the object for this task are fixed.

The object being pushed is an right-angled triangular object with dimensions 0.3m x 0.15m x 0.07m, and a weight of 2.5kg. The tool on the end effector is a cubic peg with side lengths of 45mm 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 measures for the policy.

Results and Discussion

The results for the simulation are shown in Figure 3.6a) 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 11mm and the orientation error is less than 30degrees. 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 neighbor 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.



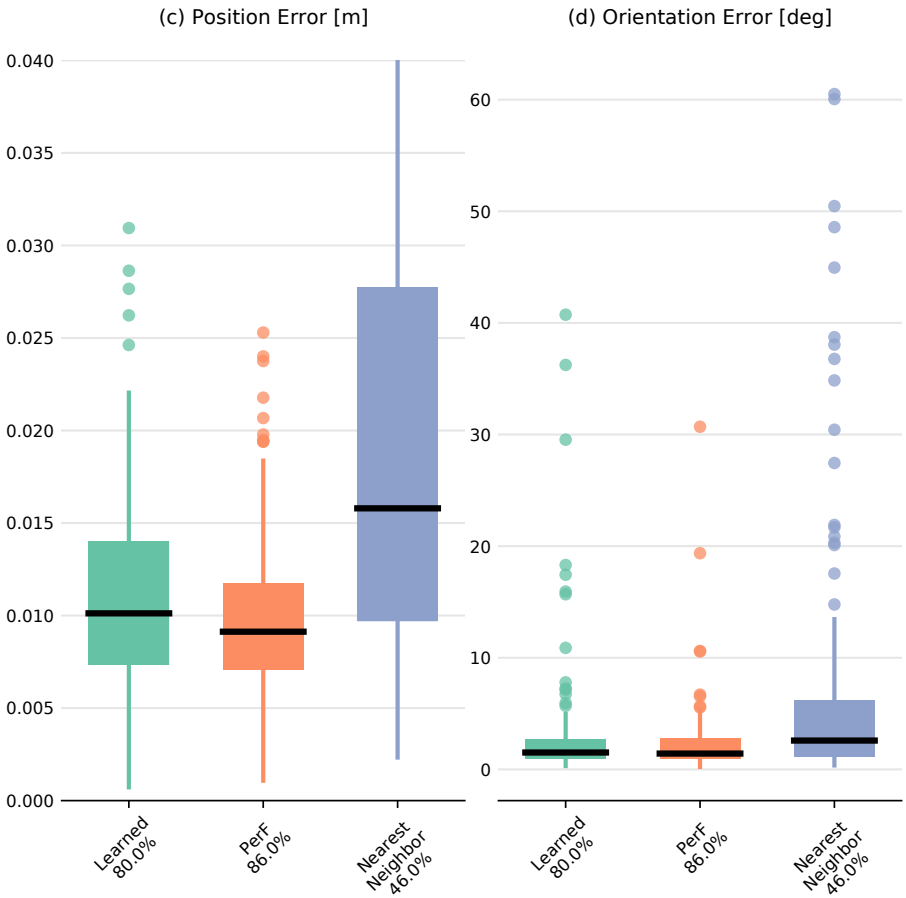


Figure 3.6: The final position error (a, c) and orientation error (b, d) of the push task in simulation (a, b) and on the real system (c, d). The box plots show the median (black line) and interquartile range (25^{th} and 75^{th} percentile); the lines extend to the most extreme data points not considered outliers, and outliers are plotted individually. The success percentages are shown below the method names.

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 4mm but keep the same angular maximum.

Table 3.1: The median performance values and the 25^{th} and 75^{th} percentiles for both tasks. A "-" indicates that configuration was not evaluated.

Task	Performance Measure	Environment	*Learned*		*PerF*		*Nearest Neighbor*		*Direct*		*Single Policy*	
			Median	Percentiles	Median	Percentiles	Median	Percentiles	Median	Percentiles	Median	Percentiles
Obstacle	Total Reward	Simulation	5050	(4467, 5531)	**5013**	(4290, 5462)	4834	(3607, 5357)	4594	(2635, 5143)	4496	(-200, 5184)
		Reality	4963	(4357, 5414)	**4966**	(4238, 5426)	4782	(3625, 5327)	–	–	–	–
	Finish Time [sec]	Simulation	5.7	(5.3, 7.3)	**5.9**	(5.3, 7.5)	6.1	(5.4, 12)	7	(5.9, 12)	7.6	(5.7, 12)
		Reality	5.8	(5.4, 7.4)	**6.1**	(5.4, 7.8)	6.1	(5.5, 12)	–	–	–	–
Push	Position Error [m]	Simulation	0.002	(0.002, 0.003)	**0.006**	(0.004, 0.009)	0.011	(0.007, 0.083)	0.007	(0.005, 0.014)	0.016	(0.007, 0.123)
		Reality	0.01	(0.007, 0.014)	**0.009**	(0.007, 0.012)	0.016	(0.01, 0.028)	–	–	–	–
	Orientation Error [deg]	Simulation	0.15	(0.07, 0.33)	**0.16**	(0.07, 0.34)	0.18	(0.08, 29.48)	0.11	(0.06, 0.26)	0.29	(0.1, 45.76)
		Reality	1.51	(0.98, 2.76)	**1.42**	(0.94, 2.74)	2.58	(1.18, 6.23)	–	–	–	–

The results for the evaluation on the real system are in Fig. 3.6c) and d) as well as in Table 3.1. 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.

7 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

[1] 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)*. 2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). 2018, pp. 5964–5971. DOI: 10.1109/IROS.2018.8594319.

[2] 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*. 2014 IEEE/RSJ International Conference on Intelligent Robots and Systems. 2014, pp. 1482–1488. DOI: 10.1109/IROS.2014.6942752.

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

[4] 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. 2021.

[5] Matthias Mayr et al. "Combining Planning, Reasoning and Reinforcement Learning to solve Industrial Robot Tasks". In: *IROS 2022 Workshop on Trends and Advances in Machine Learning and Automated Reasoning for Intelligent Robots and Systems* (2022).

[6] 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.

[7] 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)*. 2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). 2017, pp. 6793–6800. DOI: 10.1109/IROS.2017.8206598.

[8] 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.

[9] 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. 2001, 752–757 vol.2. DOI: 10.1109/IROS.2001.976259.

[10] Auke Jan Ijspeert, Jun Nakanishi and Stefan Schaal. “Learning rhythmic movements by demonstration using nonlinear oscillators”. In: *Proceedings of the ieee/rsj int. conference on intelligent robots and systems (iros2002)*. CONF. 2002, pp. 958–963.

[11] Auke Jan Ijspeert et al. “Dynamical movement primitives: learning attractor models for motor behaviors”. In: *Neural computation* 25.2 (2013), pp. 328–373.

[12] Ferdinando A Mussa-Ivaldi. “Modular features of motor control and learning”. In: *Current opinion in neurobiology* 9.6 (1999), pp. 713–717.

[13] Tamar Flash and Binyamin Hochner. “Motor primitives in vertebrates and invertebrates”. In: *Current opinion in neurobiology* 15.6 (2005), pp. 660–666.

[14] Alexandros Paraschos et al. “Probabilistic movement primitives”. In: *Advances in neural information processing systems* 26 (2013).

[15] 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*. IEEE. 2011, pp. 5006–5011.

[16] KeJun Ning et al. “A novel trajectory generation method for robot control”. In: *Journal of Intelligent & Robotic Systems* 68 (2012), pp. 165–184.

[17] 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 (2018), pp. 3781–3788.

[18] 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)*. IEEE. 2019, pp. 4301–4308.

[19] 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*. IEEE. 2013, pp. 5636–5643.

[20] 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* (2011).

[21] Christopher KI Williams and Carl Edward Rasmussen. *Gaussian processes for machine learning*. Vol. 2. MIT press Cambridge, MA, 2006.

[22] 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)*. IEEE. 2016, pp. 889–894.

[23] 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)*. IEEE. 2016, pp. 3913–3919.

[24] Jeongseok Lee et al. "DART: Dynamic Animation and Robotics Toolkit". In: *Journal of Open Source Software* 3.22 (2018), p. 500. ISSN: 2475-9066. DOI: 10.21105/joss.00500. URL: https://joss.theoj.org/papers/10.21105/joss.00500 (visited on 18/02/2020).

[25] 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*. IEEE. 2010, pp. 411–416.

[26] Katharina Mülling et al. "Learning to select and generalize striking movements in robot table tennis". In: *The International Journal of Robotics Research* 32.3 (2013), pp. 263–279.

[27] K. Muelling, J. Kober and J. Peters. "Learning Table Tennis with a Mixture of Motor Primitives". In: *10th IEEE-RAS International Conference on Humanoid Robots (Humanoids 2010)*. 2010. URL: http://www.ias.informatik.tu-darmstadt.de/uploads/Publications/Muelling_ICHR_2012.pdf.

[28] S. Gomez-Gonzalez et al. "Using Probabilistic Movement Primitives for Striking Movements". In: *Proceedings of the International Conference on Humanoid Robots (HUMANOIDS)*. 2016.

[29] G. Maeda et al. "Probabilistic Movement Primitives for Coordination of Multiple Human-Robot Collaborative Tasks". In: 3 (2017), pp. 593–612. URL: http://www.ias.tu-darmstadt.de/uploads/Team/PubGJMaeda/gjm_2016_AURO_c.pdf.

[30] 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.

[31] 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.

[32] 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.

[33] 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.

[34] 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).

Paper IV

F. Ahmad, M. Mayr, S. Suresh-Fazeela, V. Krueger
Adaptable Recovery Behaviors in Robotics: A Behavior Trees and Motion Generators (BTMG) Approach for Failure Management
IEEE 20th International Conference on Automation Science and Engineering (CASE), Bari, Italy, 2024, pp. 1815-1822, doi: 10.1109/CASE59546.2024.10711715

Chapter 4

Adaptable Recovery Behaviors in Robotics: A Behavior Trees and Motion Generators (BTMG) Approach for Failure Management

1 Abstract

In dynamic operational environments, particularly in collaborative robotics, the inevitability of failures necessitates robust and adaptable recovery strategies. Traditional automated recovery strategies, while effective for predefined scenarios, often lack the flexibility required for on-the-fly task management and adaptation to expected failures. Addressing this gap, we propose a novel approach that models recovery behaviors as adaptable robotic skills, leveraging the Behavior Trees and Motion Generators (BTMG) framework for policy representation. This approach distinguishes itself by employing reinforcement learning (RL) to dynamically refine recovery behavior parameters, enabling a tailored response to a wide array of failure scenarios with minimal human intervention. We assess our methodology through a series of progressively challenging scenarios within a peg-in-a-hole task, demonstrating the approach's effectiveness in enhancing operational efficiency and task success rates in collaborative robotics settings. We validate our approach using a dual-arm KUKA robot.

2 Introduction

In dynamic operational environments, ensuring the efficiency and adaptability of collaborative robots is crucial. Unlike traditional manufacturing line robots, collaborative robots are designed for on-the-fly task deployment, facing a unique set of challenges and failures. A Failure is defined as an inability to perform a task as intended due to unforeseen errors or disturbances, which may include hardware malfunctions, software bugs, or unexpected environmental conditions [1]. For instance, in a piston engine assembly process [2], common issues such as misalignment of the engine block, obstruction by misplaced tools, and incorrect piston orientation due to handling errors can lead to substantial production delays. Addressing these failures promptly and effectively is crucial for maintaining seamless operations and efficiency.

Current strategies for managing these failures include human intervention, systematic failure analysis [3], and automated recovery strategies [4, 5]. Human intervention relies on operator expertise for problem-solving. Failure analysis systematically identifies root causes to prevent future issues, but it requires time and expertise. Automated recovery strategies, on the other hand, leverage intelligent systems for quick detection and correction of failures, significantly reducing downtime and enhancing consistency. However, these strategies come with high initial costs and complexity in integration. Automated recovery strategies often rely on predefined scenarios and responses, lacking the flexibility to adapt to different variations of the expected failures [4]. While they can efficiently address a range of anticipated problems, their effectiveness diminishes where adaptability and rapid response to novel challenges are to be addressed.

The necessity for adaptive recovery behaviors is evident in collaborative tasks like piston engine assembly, where the nature of an obstruction dictates the required strategy. Whether it involves removing a small obstacle or applying force to displace a larger one, the recovery behavior must flexibly adjust its approach based on the specific challenge. We propose a hybrid approach that combines human expertise with the dynamic adaptability of recovery behaviors. Unlike traditional automated strategies that rely on predefined responses, our method leverages RL to dynamically refine and adjust recovery parameters. For instance, in the piston engine assembly problem, RL allows the system to learn the precise force needed to push an obstacle based on its weight without compromising safety. Although a reasoner can determine the correct parameters, it requires a detailed task model. Instead, we use RL to optimize behavior parameters best suited to the task. This adaptability ensures our robots can effectively handle a wider range of failure scenarios with minimal human intervention, making the recovery process more efficient and responsive.

Inspired by the literature, one effective way to enhance robot capabilities involves the adoption of skills [6]. These robot skills, defined by specific parameters, preconditions, and

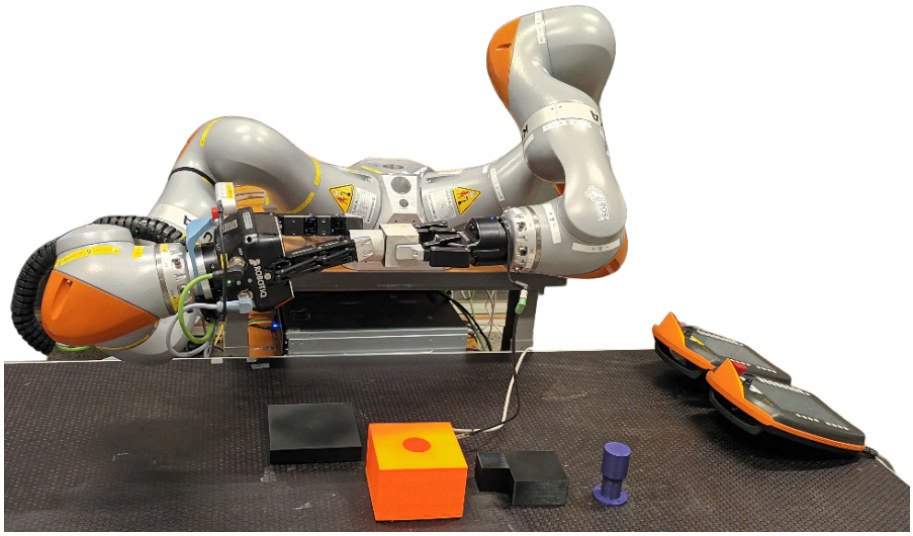


Figure 4.1: The real dual-arm *KUKA iiwa* setup executing a handover task before inserting the peg in the orange block. On the table, the purple peg for Scenario 1, alongside various obstacles that block the opening an be seen.

postconditions [7, 8], enable interaction with the environment in a robust, adaptable, and flexible manner, mirroring the qualities we seek for effective failure recovery. Building on this foundation, we suggest to model recovery behaviors as dedicated robotic skills, distinct from standard production skills and specifically designed to address failures, see Figure 4.2. This strategy equips robots with a specialized toolkit for managing known failure scenarios, streamlining the skill set for simplicity and reducing maintenance requirements. For example, in piston engine assembly, alongside the primary assembly skill, we could have a recovery skill like 'pick-place' to remove obstacles such as misplaced tools blocking the engine hole.

Various execution strategies for skills have been explored in literature [9, 10]. We have chosen the Behavior Trees and Motion Generators (BTMG) framework for its strengths in robustness, modularity, interpretability, and reactivity, which are good for effectively handling complex robotic tasks. This framework serves as the foundation for representing both standard operational skills and recovery behaviors.

In this paper, we extend the BTMG representation with the introduction of adaptable recovery behaviors inside the representation, drawing on insights from Styrud et al. [11]. The parameters of these recovery behaviors can be set manually, through reasoning, or using RL, offering flexibility based on the complexity of the task. We evaluate this approach with the peg-in-a-hole task, by gradually introducing failures to increase task complexity and demonstrate the necessity for sophisticated recovery behaviors. The failure instances are

specified by experts, and we assume the existence of a module that handles failure detection and identification. Following are the main contributions of the paper:

- We extend the BTMG policy representation with the introduction of adaptable recovery behaviors, incorporating RL to dynamically adjust behavior parameters in response to task requirements.
- We present challenging scenarios as failure cases and adapt the parameters of recovery behaviors to suit the specific requirements of the task.
- We evaluate the performance of the proposed method on a peg-in-a-hole task using dual arm KUKA iiwa robot, demonstrating its effectiveness in adapting to and recovering from failures. .

3 Related Work

The development of recovery behaviors to mitigate failures and disturbances has emerged as a critical area of research, aiming to enhance the resilience and autonomy of robotic systems. In [12], an approach was introduced for the recovery of high-degree-of-freedom motor behaviors in robots, utilizing rank-ordered differential constraints to quickly adapt to damage, malfunction, or environmental changes. [13] uses Deep Reinforcement Learning to control recovery maneuvers for quadrupedal robots, showcasing dynamic and reactive behaviors that enable a robot to recover from falls with a high success rate. [14] developed a state-dependent recovery policy that allows robots to recover from external disturbances across various tasks and conditions. In [15], the authors explored push recovery for bipedal robot locomotion, integrating decision-making and motion planning to address perturbations. Lastly, [16] proposes the T-resilience algorithm, a novel method that enables robots to autonomously discover compensatory behaviors in unanticipated situations, thereby facilitating fast damage recovery.

Furthermore, the utilization of recovery behaviors in behavior trees (BT) has also emerged as a recent interesting area of research. [17, 18] provides a detailed survey on behavior trees in the domain of robotics and AI, highlighting the usage of BT across a wide landscape. [1] introduces a framework that integrates an execution generation tool, a learning module, and a recovery pipeline to facilitate error detection, diagnosis, and recovery, showcasing the potential of BT in managing complex error recovery processes with minimal human intervention. In the context of wheeled robots, [19] demonstrates a hierarchical online hybrid planner for autonomous navigation. They utilize BT in enabling wheeled-legged robots to autonomously navigate and recover from collisions or planner failures, emphasizing the framework's capability to handle dynamic challenges without human intervention.

On a side note [20] introduced a direct extension to BT by adding a new leaf node type within the structure, aimed at specifying desired states rather than actions. This innovation demonstrates improved runtime adaptability and suggests a method for integrating recovery states directly into the BT structure, enhancing their flexibility in error recovery scenarios. [21] introduces a framework called Robust Logical-Dynamical Systems (RLDS), which combines the advantages of BT with theoretical performance guarantees. RLDS exemplifies how BT can be extended to achieve robust, reactive behavior in dynamic environments, particularly in manipulation tasks.

Our work distinguishes itself from aforementioned studies by leveraging RL to dynamically define the parameters of recovery behaviors, a novel approach that significantly enhances adaptability and effectiveness in failure management. This emphasis on RL-driven parameterization underscores its potential utility, a point we elaborate on through various failure scenarios in our experimental section, demonstrating the benefits of integrating RL into recovery strategies.

4 Background

In this section, we discuss the relevant concepts that serve as background knowledge for this paper.

4.1 Behavior Trees

Behavior Trees (BT) [22] are a hierarchical model for task planning and decision-making, widely known in robotics [17] for their modularity, flexibility, and clarity. Initially conceived for video game AI to simulate complex behaviors, BT have been effectively adapted for robotic tasks, enabling structured execution of actions from simple to complex decision-making processes. A BT structures as a directed acyclic graph, initiating execution from the root node and ticking at regular intervals to adapt dynamically to environmental changes. This execution involves traversing the tree based on control logic, evaluating conditions, executing actions, and applying decorators to modify outcomes [23]. The nodes are executed only when they are ticked and return *Success*, *Failure* or *Running*. *Control flow nodes* are the non-leaf nodes that control the execution flow, with sequence (logical AND) and selector (logical OR) being the most common. These nodes are responsible for determining the order and conditions under which child nodes are executed. The leaf nodes are called *execution* nodes and are further divided into *action* and *condition* nodes. *Condition* nodes only return *Success* or *Failure* and are used to evaluate the robot's state or environment. *Action* nodes execute the tasks, such as movement or manipulation, and return statuses indicating *Success*, *Failure* or *Running*. Lastly, we have *decorator* nodes that modify the

behavior or outcome of their child nodes to meet specific criteria or constraints.

BT offer several advantages in robotics. The readable and hierarchical structure enhances modularity [24, 25], allowing for easy modification and expansion of robotic behaviors. This modularity, combined with the clarity of the BT framework, simplifies understanding and debugging, making BT an attractive choice for complex robotic applications. Furthermore, BT also support reactivity [26], enabling robotic systems to dynamically respond to changes in the environment.

4.2 Behavior Trees and Motion Generators (BTMG)

Leveraging the foundational structure of Behavior Trees, Rovida et al. [2] integrate it with an arm motion generation strategy known as Motion Generators (MG) to develop the Behavior Trees and Motion Generators (BTMG) policy representation. MG, as detailed in [2], employs an impedance controller to control the robot's end-effectors in Cartesian space. This approach not only enables the execution of the primary motion but also permits the superimposition of additional motions through a generic varying Cartesian wrench. Furthermore, MG incorporates mechanisms for constraining velocities, accelerations, and torques, thereby addressing safety requirements comprehensively. For an in-depth exploration of MG refer to [2]. The addition of MG to the BT structure is done via having *action* and *condition* nodes specific to the MG. An example would be a node that allows us to change the stiffness of the end-effector or specify the force applied by the end-effector. This means not only we can control the flow of execution but also specify controller specific values. This gives an additional control over the actual execution of a task.

A BTMG is a parameterized policy representation with parameters broadly categorized into two types: *intrinsic* and *extrinsic*, as mentioned in [27, 28]. *Intrinsic* parameters encompass elements like the structure of BT, the quantity of BT nodes, and the type of motion generator employed. Conversely, *extrinsic* parameters include variables such as the applied force, position offsets, and the end-effector's velocity. The specification of *extrinsic* parameters can be done manually [2], through reasoning, or by employing RL [29, 30, 28], offering flexibility in adapting the BTMG framework to diverse tasks and environments.

In our prior work, we have successfully used BTMG policy representation [2] for skill execution strategies in complex robotic tasks, including peg-in-a-hole, object pushing, and obstacle avoidance [31, 29]. We have further enhanced this approach by employing RL to dynamically learn and adjust the BTMG parameters [31], allowing for adaptability in response to task variations [27, 32]. Additionally, we have utilized planning techniques within the BTMG framework to sequence skills effectively [30] and demonstrated how incorporating priors can expedite the learning process [33].

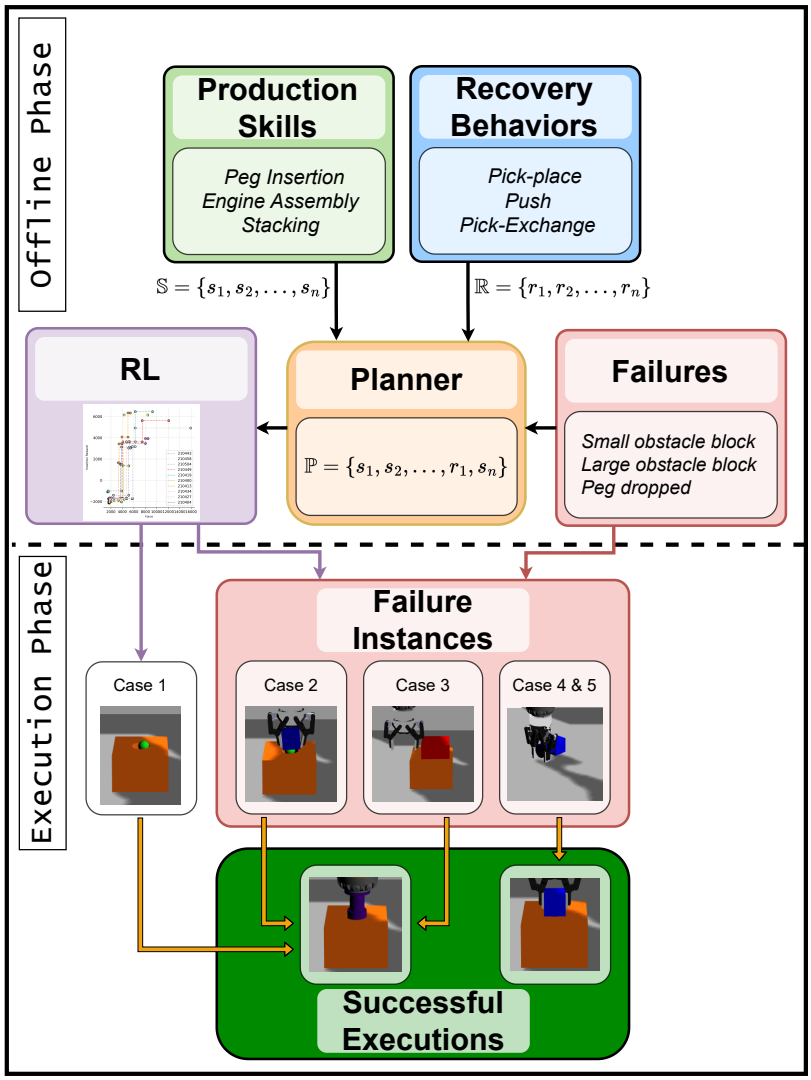


Figure 4.2: The figure shows the peg-in-a-hole task execution using our approach. We have separate sets of production and recovery behaviors. We can use a planner to come up with a sequence for a given failure specification. Subsequently, we tune the learnable parameters via reinforcement learning. Ultimately, appropriate recovery behaviors and skills are applied based on the identified failure, ensuring successful peg insertion. In the image we see successful peg insertions for all the scenarios.

4.3 Learning parameters of BTMG

In addition to predefined configurations, the extrinsic parameters of the BTMG representation can also be learned. This capability is crucial for adjusting the robotic skills dynamically, ensuring efficient task execution across diverse scenarios. To achieve this, we employ a policy optimization search method, as outlined in [34, 35], which is instrumental in learning the extrinsic parameters present in the *action* nodes of the tree. The objective is to derive a policy π, where the action $\mathbf{u} = \pi(\mathbf{x}|\boldsymbol{\theta})$ is determined based on the state $\mathbf{x}$ and policy parameters $\boldsymbol{\theta}$, aimed at maximizing the expected long-term reward over T time steps of policy execution. The optimization of these parameters is facilitated through *Bayesian Optimization* (BO) [36], enabling the identification of extrinsic parameters that are adaptable to diverse situations. For an in-depth exploration of this optimization process and its applications, refer to [30, 29].

5 Approach

In this section, we outline our approach, beginning with the assumptions guiding our work. We then detail the recovery behaviors implemented, the role of the planner in our framework, and conclude with an overview of the experimental scenarios designed to test our methodology. The overall approach is shown in Figure 4.2.

5.1 Assumptions

Following are the assumptions for this work:

1. We operate under the premise that we are addressing expected and known failures within operational processes, leveraging human experience and historical data to anticipate these failures.
2. Given a comprehensive set of skill primitives, we assume our system has the capability to dynamically generate suitable recovery behaviors for any known failure, assuming a solution exists within the parameter space defined by these skills.

The first assumption acknowledges the predictability of common failures in operational processes, with an understanding that basic parameters like object type, size, and weight are known and can be stored as knowledge about the system. For instance, in SkiROS2, this information can be managed using the world model [8, 7, 37].

Addressing the second assumption more deeply, our approach leverages the natural capabilities of skills, planning, and parameter estimation to generate recovery behaviors tailored to

specific failure scenarios. This method allows for the dynamic creation of recovery strategies by learning the necessary parameters (categorized as *extrinsic* parameters within the BTMG framework) and determining the optimal sequence of skills for complex error situations. This strategy, inspired by the methodologies demonstrated in [11], ensures that we are not constrained to having a predefined skill for each recovery situation but can adapt and respond effectively to a wide range of failures.

5.2 Recovery Behaviors

In our framework, recovery behaviors are defined as specialized skills aimed at restoring a robotic system to its desired state after encountering a failure. These behaviors, defined by specific parameters, preconditions, and postconditions, are crafted from a predefined set of skill primitives, ensuring a seamless integration into the robot's comprehensive skill set. Each recovery behavior has deterministic effects. For example, the pick-place behavior always involves picking the object and placing it at a designated location. While the actual position may vary, the effect of the action remains the same. Within the BTMG policy representation, these recovery behaviors are integrated into the broader execution strategy, leveraging the capabilities of the SkiROS2 platform [7] for effective implementation.

The generation of recovery behaviors from skill primitives is inspired by the approach outlined in [11, 38], emphasizing the versatility and power of a well-defined set of primitives. Our set includes:

- **GripperOpen** and **GripperClose**, controlling the state of the gripper.
- **GoToLinear**, moving the end-effector linearly while maintaining orientation. Also allows positional offsets in specified directions.
- **ChangeStiffness**, adjusting the stiffness of the end-effector.
- **ApplyForce**, applying force in a specified direction.

These primitives serve as the building blocks for constructing the recovery behaviors necessary for addressing specific failure scenarios encountered during task execution. The parameters for these behaviors can be finely tuned manually, through reasoning, or using RL, with RL playing a pivotal role in enhancing their robustness and adaptability. We identify which parameters require optimization through RL [30], and refine them based on the task's needs using Bayesian Optimization (BO), as detailed in Section 4.3.

Following are the recovery behaviors used in this paper:

- **Pick-Place:** Generated from *GripperOpen*, *GripperClose*, *GoToLinear*, and *ChangeStiffness*, this behavior enables obstacle removal, specifying parameters like *arm* and *obstacle*.
- **Push:** Incorporating *ApplyForce* alongside the other primitives, this behavior is tailored for displacing heavier obstacles, with parameters such as *force* being optimized through RL.
- **Pick-Exchange:** A complex behavior utilizing all five primitives to facilitate object transfer between arms. Additionally allows *offsets* in x and y directions that can be set manually or through RL.

This methodology underscores the power of our set of skill primitives to generate the necessary recovery behaviors effectively. It also illustrates the ease with which this set can be extended should the need arise, ensuring that recovery behaviors are both situationally dependent and swiftly generated based on the available primitives. The adaptability and quick generation of these behaviors, as demonstrated in [11], are critical for our approach, allowing for rapid response to a wide range of failure scenarios.

5.3 Planner

In the context of collaborative tasks, our approach leverages the knowledge of expected failures to inform the design of recovery behaviors, encoding these failures as preconditions and postconditions. For instance, during a peg-in-a-hole task, a typical failure such as an obstruction in the hole by a small block can be explicitly defined as a precondition for triggering a recovery behavior. The successful clearance of the obstruction, resulting in an unblocked hole, is set as the postcondition, marking the completion of the recovery behavior. This structured encoding allows for the potential use of planning algorithms to sequence the necessary recovery behaviors for the task at hand, although it's important to note the practical challenges involved.

While we have demonstrated the use of the Problem Domain Description Language (PDDL) planner within the SkiROS2 framework to orchestrate sequences of skills for robotic tasks [30], applying such planning in real-time collaborative scenarios poses unique challenges. In these settings, cycle times are critical, and it may not be feasible to invoke planning for every action, especially for tasks requiring rapid execution. Instead, a strategy [26, 11] can be employed where preconditions are continuously monitored, and the planner is triggered only when specific conditions are not met, necessitating recovery actions. This ensures planning is used efficiently, only when necessary to address deviations from expected task execution, thus maintaining operational efficiency while benefiting from adaptable recovery behaviors. Through this tailored application of planning, we can navigate the complexities of integrating automated recovery strategies effectively.

5.4 Scenarios

To evaluate our approach, we introduce a series of progressively more challenging scenarios within the peg-in-a-hole task, each necessitating distinct recovery behaviors, see Figure 4.3. In every scenario, we focus on learning the parameters of the peg insertion skill. The differentiation among these scenarios hinges on the utilization of recovery behaviors, the method of parameterization for these behaviors (learned via RL or manually specified), and whether the execution of a recovery behavior necessitates a relearning of the *PegInsertion* skill parameters due to changes in the task environment. In all the scenarios, we learn the parameters *PegInsertion* skill via RL.

1. **Baseline:** Utilizes only the *PegInsertion* skill, serving as control (Figure 4.3a).
2. **Static Recovery:** Uses a manually specified *pick-place* recovery behavior to address a simple obstruction (Figure 4.3b).
3. **Dynamic Recovery:** Employs a *push* recovery behavior with RL-determined parameters for handling a more complex obstruction (Figure 4.3c).
4. **Static Recovery with Behavior Changes:** Features a manually specified *pick-exchange* recovery behavior that alters the task environment, necessitating the relearning of peg insertion skill parameters (Figure 4.3d).
5. **Dynamic Recovery with Behavior Changes:** Similar to the previous scenario but uses RL to learn the *offsets* for grasping the peg during the *pick-exchange* recovery behavior (Figure 4.3d).

For all scenarios, we utilize the reward function in [29]. While it is true that different behaviors might benefit from distinct reward functions to reflect their unique optimization criteria, our study focuses on the overall adaptability and robustness of the recovery behaviors.

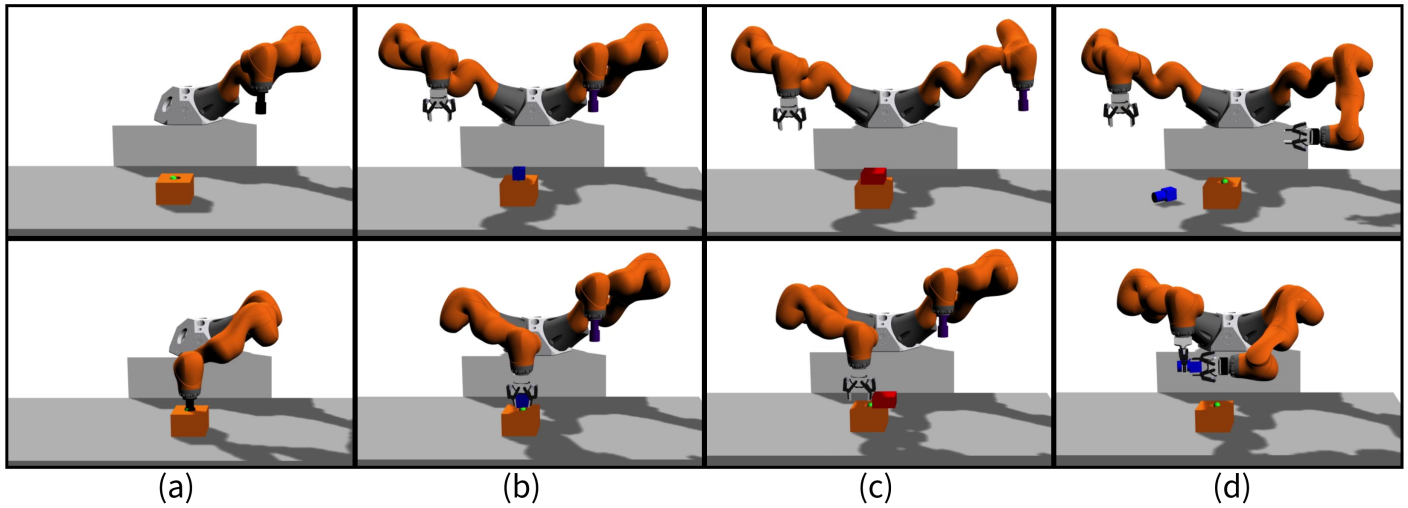


Figure 4.3: Illustration of the *PegInsertion* skill alongside its associated failure scenarios and corresponding recovery behaviors. Panel (a) shows the initial and final states of the *PegInsertion* skill without failure. Panel (b) displays the failure state due to the small blue obstacle blocking the hole with the recovery behavior of *pick-place*. Panel (c) showcases the failure state due to the large red obstacle blocking the hole with the recovery behavior of *push*. Lastly, panel (d) presents the failure states due to the peg being dropped with the recovery behavior of *pick-exchange*.

6 Experimental Setup

Our set of experiments evaluate the effectiveness of recovery behaviors in a peg-in-a-hole task, progressively introducing more challenging failures to require different recovery strategies. The task primarily utilizes a single production skill, the *PegInsertion* skill, complemented by various implementations of *pick-place*, *push* and *pick-exchange* recovery skills, differing based on the learning status of their parameters. The experiments are conducted in the *DART* simulator [39], employing a Cartesian impedance controller for arm manipulation [40].

6.1 Peg-in-a-hole Task

The objective of our task is to insert a peg into a hole within a box, as depicted in Figure 4.1 and 4.3. We utilize the *GoToLinear* skill for precise end-effector positioning and the *PegInsertion* skill for insertion. The *PegInsertion* skill activates upon the end effector's arrival at the box's approach pose, where it dynamically adjusts the end effector's stiffness to zero in the z-direction (downwards) and applies a targeted force in the same direction. Additionally, it incorporates an overlaying circular motion, akin to an Archimedean spiral, aimed at the box's center. The learnable parameters of this skill, including the end-effector's applied force, path velocity, path distance, and radius, are crucial for successful insertion. For an in-depth exploration of the BTMG representation and further skill specifics, we refer the interested reader to [30].

The task is framed as a multi-objective challenge focusing on successful insertion and minimizing applied force, with reward metrics aligned with those in [30]. To evaluate the successful insertion, we employ a trio of reward metrics: the success of the BT execution, the proximity of the peg to the hole, and its distance from the box. For gauging the applied force, a singular reward metric quantifies the cumulative force exerted by the peg. To enhance system robustness, we use domain randomization, varying the location of the block with a hole by a standard deviation of 8 via Gaussian distribution and changing the arm's starting position across five positions. For each scenario, we conduct 40 iterations, with each iteration being evaluated five times to account for domain randomization. Each scenario is repeated 10 times. This approach ensures robustness in our assessments by introducing variability in the task environment. A policy is considered successful if it manages to achieve peg insertion in at least three out of the five evaluations. Across all scenarios, the clearance between the peg and the hole is maintained at 3 to standardize the task difficulty.

6.2 Results and Discussion

Across all scenarios and for each repetition, we identified at least one policy capable of successfully inserting the peg into the hole, demonstrating the effectiveness of our recovery behaviors and the adaptability of the *PegInsertion* skill under varied failures. Notably, the introduction of recovery behaviors, whether static or dynamic, did not impede the task's success, highlighting the robustness of our approach. The Pareto fronts for each scenario illustrate the trade-off between insertion success and applied force, with diverse policies achieving the task across all repetitions, see Figure 4.4, 4.5, 4.6, 4.7 and 4.8. Each distinct color in Figure 4.4, 4.5, 4.6, 4.7 and 4.8 represents a single experiment or repetition of a scenario and bold points represent the pareto-optimal execution policy. This diversity underscores our approach's flexibility, enabling effective completion of a task subjected to multiple objectives.

By focusing on collaborative robots and leveraging the adaptability of recovery behaviors, our approach provides an alternative to automated recovery strategies. Even though, the static and dynamic scenarios we presented could be addressed through automated recovery strategies, our method uses RL to dynamically adjust recovery behavior parameters, ensuring effective response to environmental changes. This adaptability, crucial for on-the-fly task management, sets our approach apart, offering a flexible solution to the changing demands of dynamic environments.

Additionally, it is pertinent to reference findings from our previous work [30], where we evaluated the efficacy of using RL to specify parameters for the *PegInsertion* skill against three distinct baselines: planning with predefined parameter values, random policy selection, and policies chosen by robot operators. The learned policies from our RL-based approach outperformed the alternative strategies in terms of success rates. Therefore, we opted not to directly compare our current approach with these baselines in the present study. Our focus in this study was the effectiveness of adaptable recovery behaviors for failure handling. Furthermore, it is also worth mentioning that the adaptability of our approach can be further enhanced by accommodating different task variations, as demonstrated in our previous work [28]. In that study, we trained a model to predict the long-term reward of different policies, showing that the policies suggested by this model perform comparably to those optimized directly through RL. In principle, this predictive model could be used in place of direct RL optimization, potentially accelerating the adaptability process for recovery behaviors in response to varying task conditions.

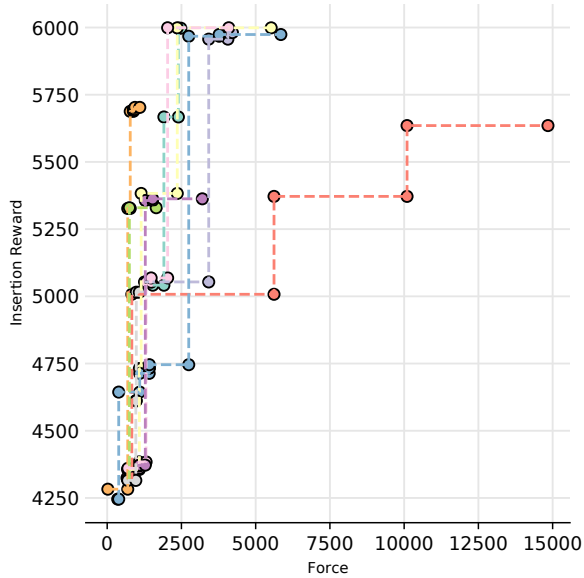


Figure 4.4: Pareto front for Scenario 1: *Baseline*. Each experiment is denoted by a distinct color, with each bold point representing a pareto-optimal policy ready for execution. The optimizer tries to strike a balance between the reward for successful insertion and the force applied by the end-effector.

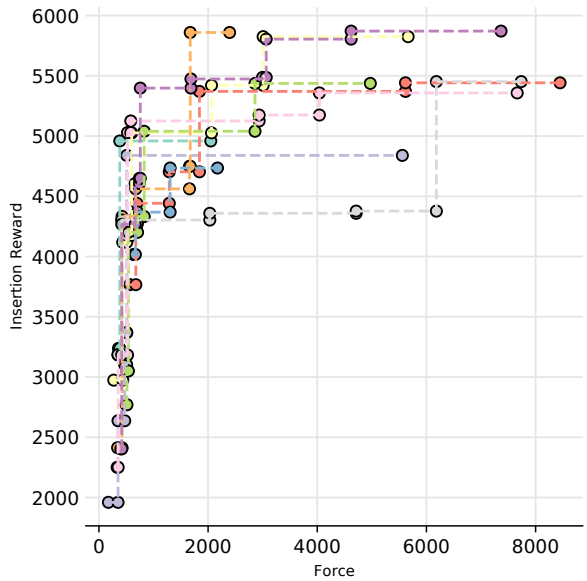


Figure 4.5: Pareto front for Scenario 2: *Static Recovery*. This demonstrates that achieving a higher insertion reward necessitates greater force application, as observed from the force exerted by the end-effector during the search for the hole.

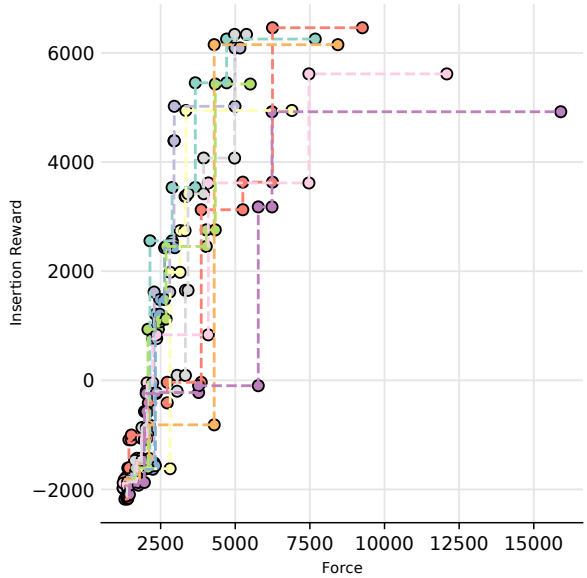


Figure 4.6: Pareto front for Scenario 3: *Dynamic Recovery. In this scenario*, the application of force is notably higher because pushing the obstacle away requires additional force before the peg can be inserted into the hole.

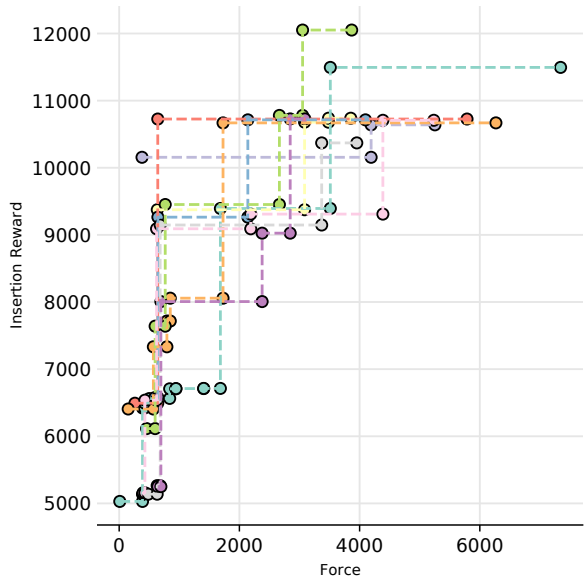


Figure 4.7: Pareto front for Scenario 4: *Static Recovery with behavior changes*. In this scenario, we observe a similar trend to Scenario 2, where a higher force is necessary as the peg searches for the hole, reflecting the similar nature of the tasks.

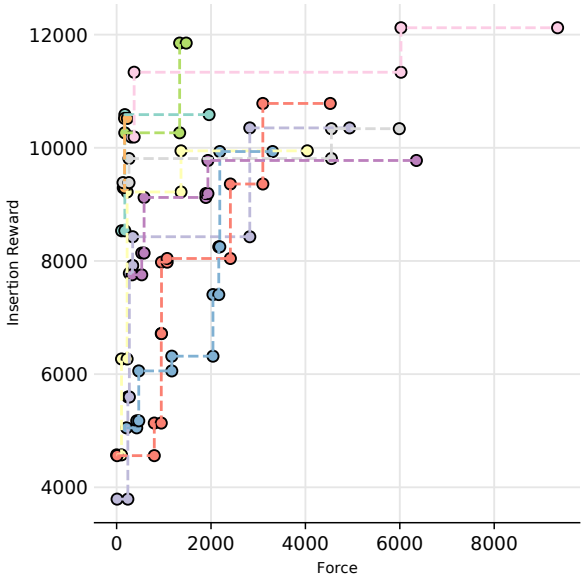


Figure 4.8: Pareto front for Scenario 5: *Dynamic Recovery with behavior changes*. In this scenario, the wide distribution of policies suggests the task's complexity, as picking up and exchanging the dropped peg significantly impacts the success of the peg insertion.

7 Conclusion and Future Work

In this paper, we presented a novel approach that models recovery behaviors as robotic skills to effectively manage and recover from failures in robotic tasks. By defining these recovery behaviors with specific parameters, preconditions, and postconditions, and utilizing the BTMG policy representation as the execution strategy, we have demonstrated a structured method to represent and implement these behaviors. The adaptability of these behaviors is enhanced through RL to dynamically adjust parameters. Our approach enables robots to autonomously recover from disruptions and resume normal operations seamlessly. We evaluated our methodology through the peg-in-a-hole task by gradually introducing challenging failures and recovering from them, thereby testing the resilience and adaptability of our recovery strategies. By framing this task as a multi-objective challenge, focusing on successful insertion while minimizing applied force, we showcased the effectiveness of our approach. Our results confirm that the integration of recovery behaviors, modeled as adaptable robotic skills within the BTMG framework, significantly enhances the robot's ability to recover from failures, thereby improving operational efficiency and task success rates.

In our future work, we aim to develop a comprehensive recovery pipeline that not only identifies failures but also selects the appropriate recovery skills automatically to address

them effectively. This pipeline will enhance our current set of recovery skills, making them capable of handling not just anticipated failures but also unexpected ones within certain limits. Our goal is to leverage the structure of a BT and its tick signals, which return different states, to pinpoint the exact location of a failure by analyzing which node returns a failure state. This diagnostic capability will enable us to match the specific pre- and post-conditions of a failure, facilitating the selection of suitable recovery behaviors from a more generalized skill set. To achieve this, we plan to explore the use of a recursive tree structure [26], which will play a crucial role in dynamically choosing the most effective recovery behavior based on the situation at hand. Additionally, we will also like to explore the creation of a dataset of various failures to demonstrate learning recovery behaviors from skill primitives, akin to a reformulation in [11] focused on error recovery. This effort will include verifying the sufficiency of our skill primitives for comprehensive recovery scenarios, enhancing the system's adaptability and resilience in complex environments.

ACKNOWLEDGMENTS

We thank Momina Rizwan and Simon Kristoffersson Lind for their discussions and feedback. This work was partially supported by the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by Knut and Alice Wallenberg Foundation. We used Generative AI tools for editing, including sentence structuring, spelling, and grammar corrections.

References

[1] Ruichao Wu, Sitar Kortik and Christoph Hellmann Santos. "Automated Behavior Tree Error Recovery Framework for Robotic Systems". In: *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2021, pp. 6898–6904.

[2] 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)*. 2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). 2018, pp. 5964–5971. DOI: 10.1109/IROS.2018.8594319.

[3] FN Jusuf et al. "Review on defenses against common cause failures on digital safety system". In: *AIP Conference Proceedings*. Vol. 2374. 1. AIP Publishing. 2021.

[4] Yumeng Lei et al. "Artificial Intelligence Planning of Failure Recovery Strategies in Discrete Manufacturing Automation". In: *2023 IEEE 19th International Conference on Automation Science and Engineering (CASE)*. IEEE. 2023, pp. 1–8.

[5] Lucas VR Alves and Patrícia N Pena. "Secure recovery procedure for manufacturing systems using synchronizing automata and supervisory control theory". In: *IEEE Transactions on Automation Science and Engineering* 19.1 (2020), pp. 486–496.

[6] Mikkel Rath Pedersen et al. "Robot skills for manufacturing: From concept to industrial deployment". In: *Robotics and Computer-Integrated Manufacturing* 37 (2016), pp. 282–291.

[7] Francesco Rovida et al. "SkiROS-A skill-based robot control platform on top of ROS". In: *Studies in Computational Intelligence*. Vol. 707. 2017, pp. 121–160.

[8] 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 (IROS)*. IEEE. 2023, pp. 6273–6280.

[9] Andreas Herzig, Laurent Perrussel and Zhanhao Xiao. "On hierarchical task networks". In: *Logics in Artificial Intelligence: 15th European Conference, JELIA 2016, Larnaca, Cyprus, November 9-11, 2016, Proceedings 15*. Springer. 2016, pp. 551–557.

[10] 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)*. IEEE. 2023, pp. 5807–5813.

[11] Jonathan Styrud et al. "BeBOP–Combining Reactive Planning and Bayesian Optimization to Solve Robotic Manipulation Tasks". In: *2024 International Conference on Robotics and Automation (ICRA)*. IEEE, 2024.

[12] George Council and Shai Revzen. "Recovery of Behaviors Encoded via Bilateral Constraints". In: *arXiv preprint arXiv:2005.00506* ().

[13] Joonho Lee, Jemin Hwangbo and Marco Hutter. "Robust recovery controller for a quadrupedal robot using deep reinforcement learning". In: *arXiv preprint arXiv:1901.07517* (2019).

[14] Hongmin Wu et al. "Recovering from external disturbances in online manipulation through state-dependent revertive recovery policies". In: *2018 27th IEEE International Symposium on Robot and Human Interactive Communication (RO-MAN)*. IEEE. 2018, pp. 166–173.

[15] Zhaoyuan Gu, Nathan Boyd and Ye Zhao. "Reactive locomotion decision-making and robust motion planning for real-time perturbation recovery". In: *2022 International Conference on Robotics and Automation (ICRA)*. IEEE. 2022, pp. 1896–1902.

[16] Sylvain Koos, Antoine Cully and Jean-Baptiste Mouret. "Fast damage recovery in robotics with the t-resilience algorithm". In: *The International Journal of Robotics Research* 32.14 (2013), pp. 1700–1723.

[17] Matteo Iovino et al. *A Survey of Behavior Trees in Robotics and AI*. 2020. arXiv: 2005.05842 [cs.RO].

[18] Razan Ghzouli et al. “Behavior Trees and State Machines in Robotics Applications”. In: *IEEE Transactions on Software Engineering* (2023).

[19] Alessio De Luca, Luca Muratore and Nikos G Tsagarakis. “Autonomous navigation with online replanning and recovery behaviors for wheeled-legged robots using behavior trees”. In: *IEEE Robotics and Automation Letters* (2023).

[20] C Pezzato, C Hernandez and M Wisse. “Active inference and behavior trees for reactive action planning and execution in robotics. arXiv”. In: *arXiv preprint arXiv:2011.09756* (2020).

[21] Chris Paxton et al. “Representing robot task plans as robust logical-dynamical systems”. In: *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2019, pp. 5588–5595.

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

[23] Magnus Olsson. *Behavior trees for decision-making in autonomous driving*. 2016.

[24] 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*. 2014 IEEE/RSJ International Conference on Intelligent Robots and Systems. 2014, pp. 1482–1488. DOI: 10.1109/IROS.2014.6942752.

[25] 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 (TCPS)* 6.2 (2022), pp. 1–36.

[26] Michele Colledanchise and Petter Ögren. *Behavior trees in robotics and AI: An introduction*. CRC Press, 2018.

[27] 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.

[28] 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)* (2023), pp. 10133–10140. URL: https://api.semanticscholar.org/CorpusID:257532298.

[29] 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. 2021.

[30] 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.

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

[32] 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, pp. 10133–10140.

[33] 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.

[34] Konstantinos Chatzilygeroudis et al. “Black-Box Data-Efficient Policy Search for Robotics”. In: *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). 2017, pp. 51–58. DOI: 10.1109/IROS.2017.8202137.

[35] Konstantinos Chatzilygeroudis et al. “A survey on policy search algorithms for learning robot controllers in a handful of trials”. In: *IEEE Transactions on Robotics* 36.2 (2019), pp. 328–347.

[36] Luigi Nardi, David Koeplinger and Kunle Olukotun. “Practical Design Space Exploration”. In: *International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems*. 2019.

[37] 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.

[38] 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.

[39] Jeongseok Lee et al. “DART: Dynamic Animation and Robotics Toolkit”. In: *Journal of Open Source Software* 3.22 (2018), p. 500. ISSN: 2475-9066. DOI: 10.21105/joss.00500. URL: https://joss.theoj.org/papers/10.21105/joss.00500 (visited on 18/02/2020).

[40] Matthias Mayr and Julian M. Salt-Ducaju. “A C++ Implementation of a Cartesian Impedance Controller for Robotic Manipulators”. In: *Journal of Open Source Software* 9.93 (Jan. 2024), p. 5194. DOI: 10.21105/joss.05194. URL: https://joss.theoj.org/papers/10.21105/joss.05194.

Paper V

F. Ahmad, J. Styrud, V. Krueger
Addressing Failures in Robotics using Vision-Based Language Models (VLMs) and Behavior Trees (BT)
To appear in European Robotics Forum 2025, vol. 36, Springer Proceedings in Advanced Robotics, Springer, 2025, ch. 43, ISBN 978-3-031-89470-1. arXiv:2411.01568, 2024.

Chapter 5

Addressing Failures in Robotics using Vision-Based Language Models (VLMs) and Behavior Trees (BT)

1 Abstract

In this paper, we propose an approach that combines Vision Language Models (VLMs) and Behavior Trees (BTs) to address failures in robotics. Current robotic systems can handle known failures with pre-existing recovery strategies, but they are often ill-equipped to manage unknown failures or anomalies. We introduce VLMs as a monitoring tool to detect and identify failures during task execution. Additionally, VLMs generate missing conditions or skill templates that are then incorporated into the BT, ensuring the system can autonomously address similar failures in future tasks. We validate our approach through simulations in several failure scenarios.

Keywords: Robotics, Failure Detection, Behavior Trees, Vision Language Models, Recovery Behaviors

2 Introduction

Modern robotic systems can handle complex tasks in controlled environments, but transitioning into dynamic, small-batch manufacturing introduces new challenges, especially around failure management. Failures; unforeseen disturbances that prevent task completion; can lead to costly delays and risks, particularly in shared workspaces [1]. The ability

to detect, identify, and recover from failures autonomously is crucial for ensuring the robustness of robotic systems.

Traditional failure management strategies in robotics include human intervention, failure analysis [2], and automated recovery strategies [1]. These approaches have limitations: human intervention is time-consuming, failure analysis requires expertise, and automated strategies often lack flexibility in handling unforeseen scenarios. Our recent work [3] introduced a novel method using automated recovery behaviors modeled as robotic skills with parameters, preconditions, and postconditions, executed through Behavior Trees and Motion Generators (BTMG) policy representation [4]. This approach optimizes recovery policies using Reinforcement Learning (RL) [5] and also adapts the parameters to different task variations [6].

However, two key limitations remain: (1) the system assumes failure detection and identification are already solved, requiring prior knowledge of the failure, and (2) it only handles known failures with predefined solutions. These limitations make it difficult to address unforeseen failures. We propose addressing these gaps by utilizing Vision Language Models (VLMs) to detect, identify, and generate solutions for unknown failures. By integrating VLMs with Behavior Trees (BTs), our approach autonomously monitors task execution, identifies failure states, and generates missing conditions or skill templates to recover from failures. The BT is then updated using a reactive planner [7] to handle similar future occurrences.

Main Contributions

- We propose a novel integration of VLMs with BTs for monitoring, failure detection, identification and recovery in robotic systems.
- We use VLMs to generate missing preconditions or skill templates to address failures and update the BT policy.
- We conduct experiments to demonstrate the effectiveness of the approach.

3 Background

This section provides essential background concepts to our proposed approach, focusing on behavior trees, reactive planner and vision-based language models.

3.1 Behavior Trees (BT)

Behavior Trees (BTs) are hierarchical models for task execution, known for their modularity and flexibility [8]. A BT organizes task execution through nodes that receive tick signals, indicating readiness for execution. BTs consist of two types of nodes: control-flow nodes and execution nodes. *Control-flow nodes* manage execution flow and return statuses of success, failure, or running; examples include Sequence (AND) and Fallback/Selector (OR). *Execution nodes*, which are leaf nodes, represent either skills ("!") or conditions ("?"). Skills perform specific tasks, while conditions evaluate the environment, returning only success or failure. BTs offer modularity and clarity, making them ideal for robotics applications, particularly in dynamic environments where flexibility is required [9].

3.2 Reactive Planner

Reactive planners generate BTs dynamically, using a backchaining approach to select skills that satisfy goal conditions [10]. The process iteratively selects skills based on their preconditions and postconditions, constructing a BT that satisfies the specified goal. This recursive process continues until a leaf node is reached or a predefined depth is attained. The planner ensures adaptability by dynamically responding to changes in the environment without requiring extensive re-planning. This planner has been extended for various applications in [11, 12].

3.3 Vision Language Models (VLM) in Robotics

Vision Language Models (VLMs) are powerful tools in robotics, enabling a deeper understanding of complex environments by combining visual data with language inputs. VLMs excel at tasks such as scene interpretation, object recognition, and generating control skills based on visual cues and task descriptions [13, 14]. Recent applications of VLMs in robotics include failure recovery, task planning, and multimodal reasoning, with systems like ReplanVLM [15] and AHA [16] demonstrating their ability to reason over failures and generate dynamic solutions.

4 Approach

We extend the existing framework [3] to handle unknown failures by integrating VLMs for failure detection, identification and recovery and generate missing preconditions or skill templates to be incorporated into the BT.

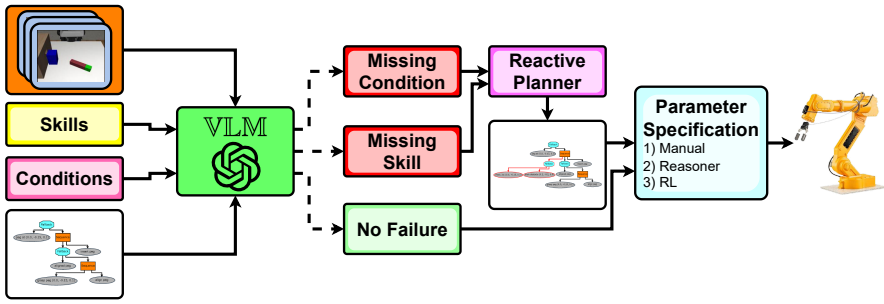
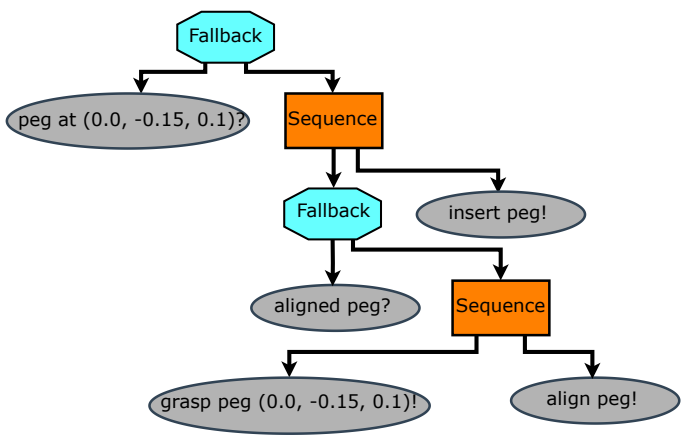


Figure 5.1: Overview of the proposed approach, where the VLM takes a set of images, skills, conditions, and a BT as input. The VLM uses this information to provide missing conditions or skills, which are then used to update the BT through a planner.

4.1 Failure Detection and Idenitification

Failure detection refers to the system's ability to recognize when a task cannot be completed due to unforeseen errors, such as hardware malfunctions or environmental disturbances. This can be achieved by using sensor data, such as from cameras or force-torque sensors, and comparing it against the expected postconditions of skills. For example, in a peg-in-hole task, if an object blocks the hole, the system detects this failure when the postcondition of the "insert" skill (peg inserted) is not met [1] (see Figure 5.3).

Failure identification involves describing the failure using existing system conditions and understanding why the task could not be completed. For instance, in the peg-in-hole task, the missing precondition when an obstacle is blocking the hole could be "Not any obstacle at hole" for the insert skill. This allows the system to formulate strategies for dealing with similar failures in the future.



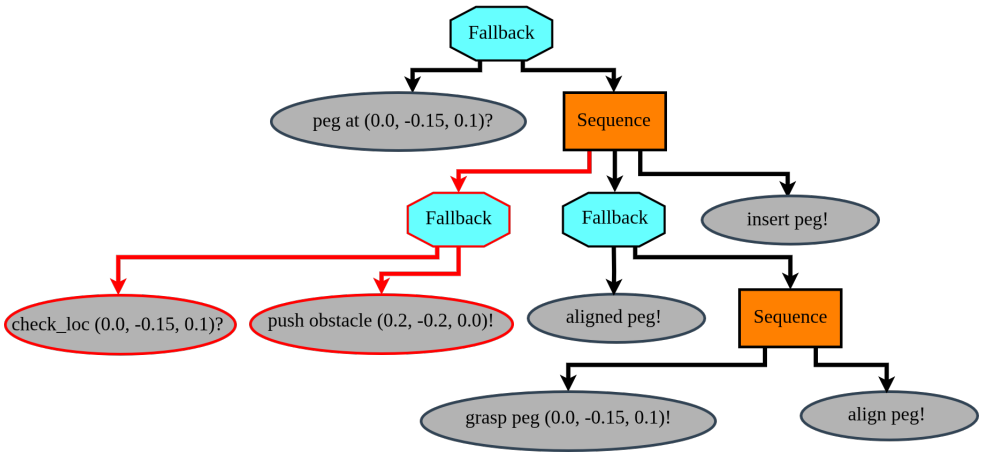


Figure 5.2: Comparison of Behavior Trees (BTs). The left side shows the initial BT, while the right side illustrates the updated BT, with the changes highlighted in red connections.

4.2 Monitoring using VLM

We use VLMs to enable failure detection, identification and recovery (Figure 5.1). Before task execution, the VLM is queried with images of the task environment, the BT structure, and the skills and conditions involved. The VLM then assesses whether the task will succeed and, if not, identifies the cause of failure (detection). It also suggests the missing condition (identification) that could prevent the failure. If the system lacks the required skill to recover from a failure (recovery), the VLM suggests an appropriate recovery skill based on the provided library of existing skills (see Figure 5.2).

4.3 Condition and Skill Template Generation

When the VLM identifies a missing condition or skill, it updates the BT accordingly. For example, if a small obstacle blocks the peg, the condition "hole free" is generated and added as a precondition to the "insert" skill. The reactive planner regenerates the BT by incorporating this condition, ensuring similar failures are handled in the future [7]. If a recovery skill is missing, the VLM generates an skill template that follows a structured format and requires some manual inputs to complete. For instance, if a large object blocks the peg hole and the gripper cannot grasp it, the VLM suggests a "push" skill template to remove the obstacle (see Figure 5.3). This template is added to the BT, and in the future, this process could be fully automated, allowing the system to autonomously recover from failures.

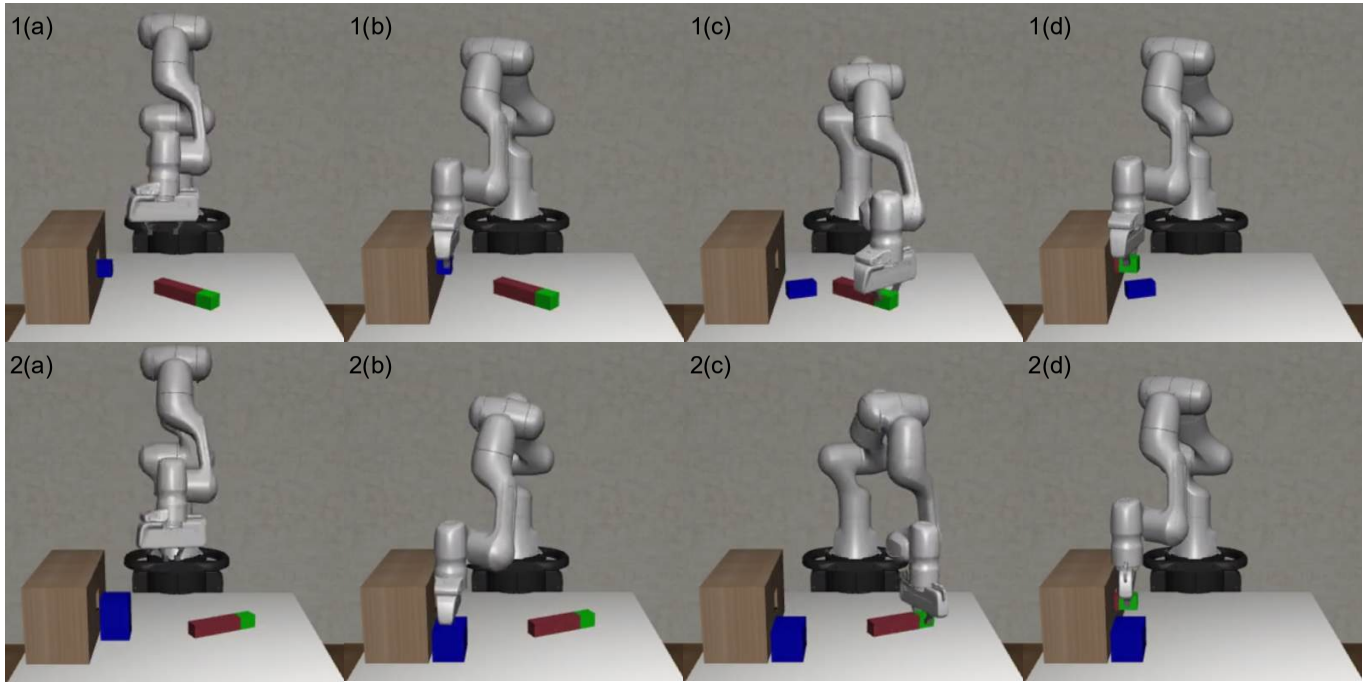


Figure 5.3: Scenes showing peg-in-hole task execution with obstacles. The first row (1(a)–1(d)) illustrates the task with a small obstacle, while the second row (2(a)–2(d)) depicts the task with a large obstacle.

5 Experiments

We validated the proposed approach using simulations in robosuite [17] and OpenAI's GPT-4. The experiments were designed around three tasks, each involving unknown failures:

- *Peg-in-Hole Task:* Two scenarios—(A) a small obstacle inside the hole, and (B) a large obstacle positioned in front of the hole.
- *Lift Task:* An additional cube is placed on top of the target object, creating an unforeseen failure.
- *Door Opening Task:* The robot attempts to open a door but lacks the precondition that the handle must be turned first.

6 Evaluation Metrics and Results

We evaluated the VLM's performance using three key metrics: consistency in failure detection and recovery, the importance of vision input, and skill feasibility considerations (ensuring that suggested skills, such as a "grasp" skill, are feasible based on the gripper's affordance and object location). For all experiments, we used model parameters of temperature and top_p set to 0.1, which resulted in more deterministic and focused outputs, reducing randomness and ensuring that the model consistently chose the most likely responses.

- **Consistency of Failure Detection and Recovery:** The VLM's reliability was tested across 10 trials per scenario, consistently detecting and recovering from failures in all tasks, achieving 100% consistency.
- **Vision Importance Ablation Study:** To assess the impact of visual input, we compared VLM (with visual input) and LLM (without visual input). In the *Peg-in-Hole (small obstacle)*, *Lift*, and *Door Opening* tasks, both models achieved 100% accuracy. However, in the *Peg-in-Hole (large obstacle)* task, the VLM achieved 100% accuracy, while the LLM reached 30% accuracy without skill feasibility considerations and 60% with feasibility checks.
- **Skill Feasibility Considerations:** When skill feasibility is considered, LLM performance improved but still fell short of the VLM. The VLM excelled in complex scenarios like the *Peg-in-Hole (large obstacle)* task, generating feasible recovery skills autonomously.

7 Conclusion and Future Work

In this paper, we introduced a method for integrating Vision Language Models (VLMs) with Behavior Trees (BTs) to autonomously detect, identify, and recover from failures in robotic systems. By generating missing conditions and skill templates, the system can effectively handle unknown failures and adapt its execution policy for future tasks. Future work will focus on several key areas: expanding the range of failure scenarios to include more complex and dynamic environments, improving the skill generation mechanism to move from generating skill templates to directly producing feasible skills, thereby reducing manual input. Additionally, we aim to fine-tune an open-source model to further enhance the system's performance and adaptability across diverse robotic tasks.

References

[1] R. Wu et al. "Automated Behavior Tree Error Recovery Framework for Robotic Systems". In: *IEEE ICRA 2021*.

[2] F. Jusuf et al. "Review on Defenses Against Common Cause Failures on Digital Safety Systems". In: *AIP Conference Proceedings 2021*.

[3] F. Ahmad et al. "Adaptable Recovery Behaviors in Robotics: A Behavior Trees and Motion Generators (BTMG) Approach for Failure Management". In: *IEEE CASE 2024*.

[4] F. Rovida et al. "Extended behavior trees for quick definition of flexible robotic tasks". In: *IEEE/RSJ IROS 2017*.

[5] M. Mayr et al. "Skill-based Multi-objective Reinforcement Learning of Industrial Robot Tasks with Planning and Knowledge Integration". In: *IEEE ROBIO 2022*.

[6] F. Ahmad et al. "Learning to Adapt the Parameters of Behavior Trees and Motion Generators (BTMGs) to Task Variations". In: *IEEE/RSJ IROS 2023*.

[7] J. Styrud et al. "Automatic Behavior Tree Expansion with LLMs for Robotic Manipulation". In: *arXiv preprint arXiv:2409.13356* (2024).

[8] M. Colledanchise and P. Ögren. *Behavior Trees in Robotics and AI: An Introduction*. CRC Press 2018.

[9] M. Iovino et al. "A Survey of Behavior Trees in Robotics and AI". In: *Robotics and Autonomous Systems 2022* ().

[10] M. Colledanchise et al. "Towards Blended Reactive Planning and Acting Using Behavior Trees". In: *IEEE ICRA 2019*.

[11] M. Iovino et al. “A Framework for Learning Behavior Trees in Collaborative Robotic Applications”. In: *IEEE CASE 2023*.

[12] J. Styrud et al. “BeBOP-Combining Reactive Planning and Bayesian Optimization to Solve Robotic Manipulation Tasks”. In: *IEEE ICRA 2024*.

[13] H. Chen et al. “Automating Robot Failure Recovery Using Vision-Language Models With Optimized Prompts”. In: *arXiv preprint arXiv:2409.03966* (2024).

[14] M. J. Kim et al. “OpenVLA: An Open-Source Vision-Language-Action Model”. In: *arXiv preprint arXiv:2406.09246* (2024).

[15] A. Mei et al. *ReplanVLM: Replanning Robotic Tasks with Visual Language Models*. Available at: `https://arxiv.org/abs/2407.21762`. 2024.

[16] J. Duan et al. *AHA: A Vision-Language-Model for Detecting and Reasoning Over Failures in Robotic Manipulation*. Available at: `https://arxiv.org/abs/2410.00371`. 2024.

[17] *Robosuite: A Framework for Robot Learning*. 2023. URL: `https://robosuite.ai`.

Appendix

Peg-in-a-Hole with Small Obstacle

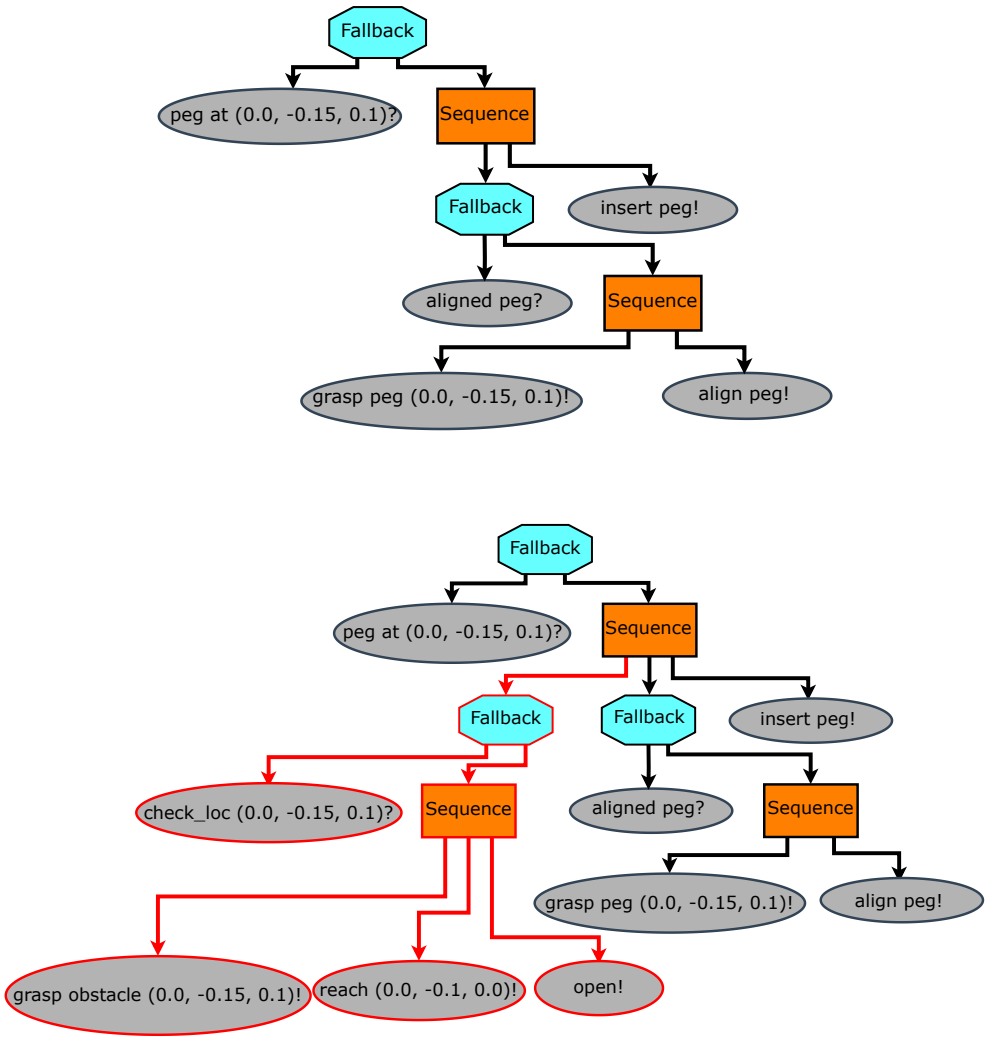


Figure 5.4: Comparison of Behavior Trees (BTs) for the peg-in-a-hole task with a small obstacle. The initial tree (top) lacks a condition that checks whether the hole is free before insertion or not. The Visual Language Model (VLM) suggests this condition, which is then added as precondition of the insert skill, producing the updated tree (bottom) generated by the reactive planner. Changes are highlighted in red.

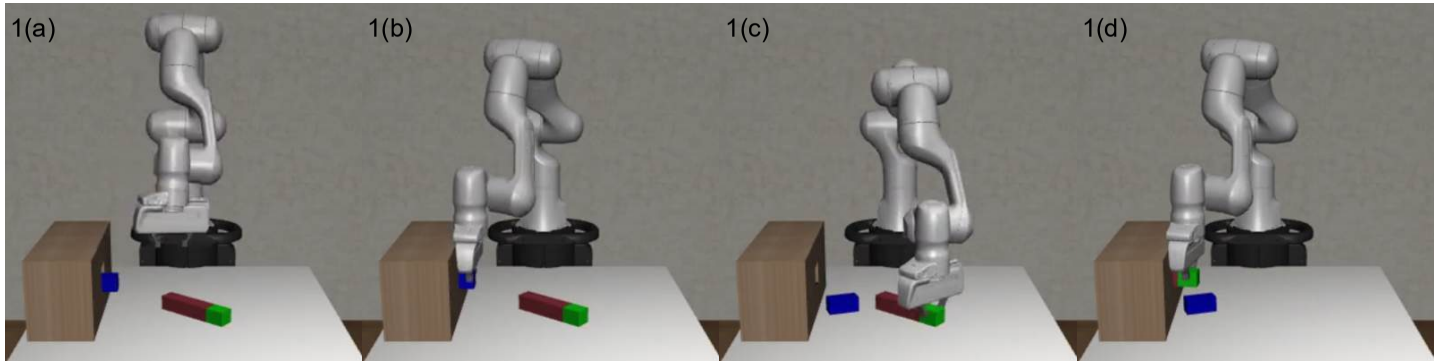


Figure 5.5: Scenes depicting the execution of the peg-in-a-hole task with a small obstacle.

Peg-in-a-Hole with Large Obstacle

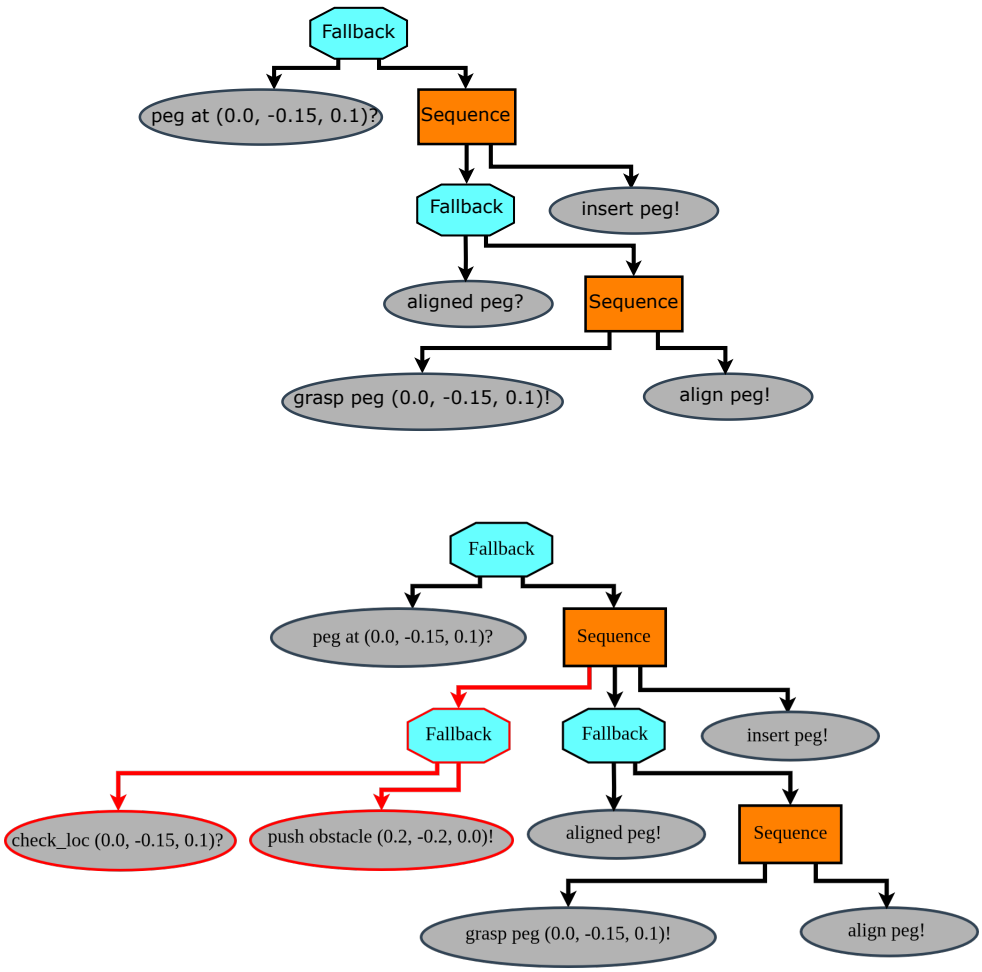


Figure 5.6: Comparison of Behavior Trees (BTs) for the peg-in-a-hole task with a large obstacle. This is similar to task 5 with difference in the size of obstacle larger than the gripper affordance. The VLM identifies that the obstacle is too large for the gripper and suggests a push skill in place of grasp, reach, and open skills as used in the small obstacle scenario (Fig. 5.4). The modified tree (right) with the added skill is shown, with changes marked in red.

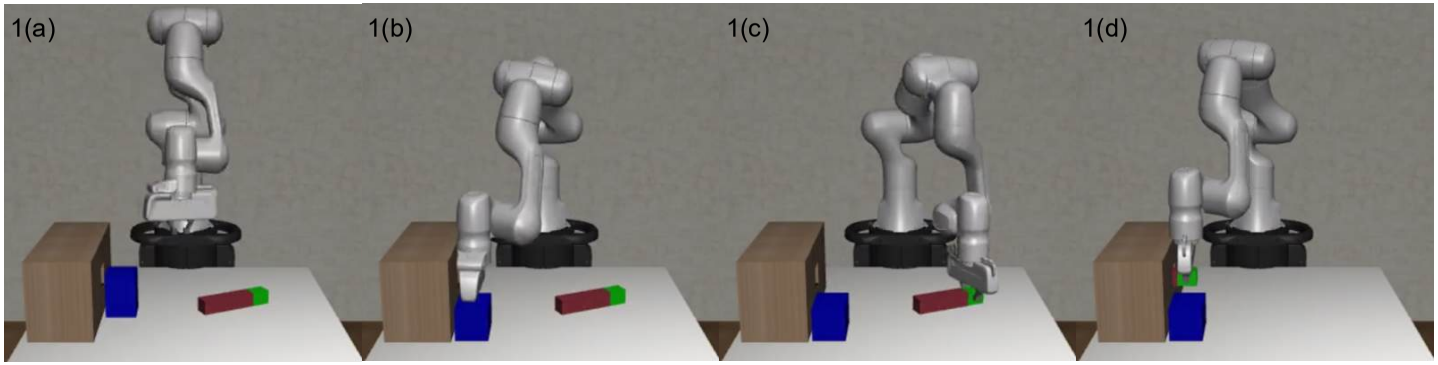
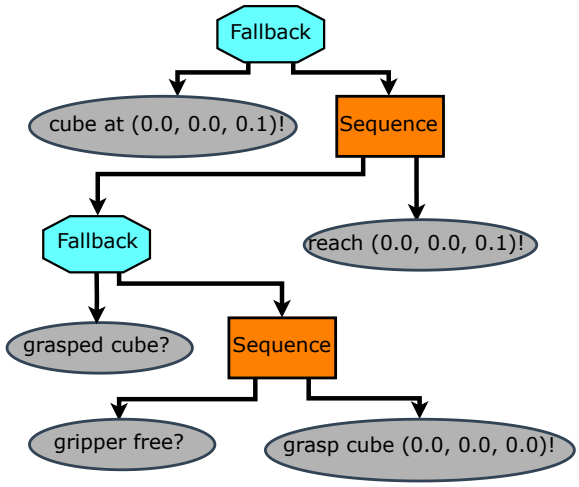


Figure 5.7: Scenes showing the execution of the peg-in-a-hole task with a large obstacle.

Lift Task


Fallback
cube at (0.0, 0.0, 0.1)!
Sequence
Fallback
reach (0.0, 0.0, 0.1)!
grasped cube?
Sequence
gripper free?
grasp cube (0.0, 0.0, 0.0)!

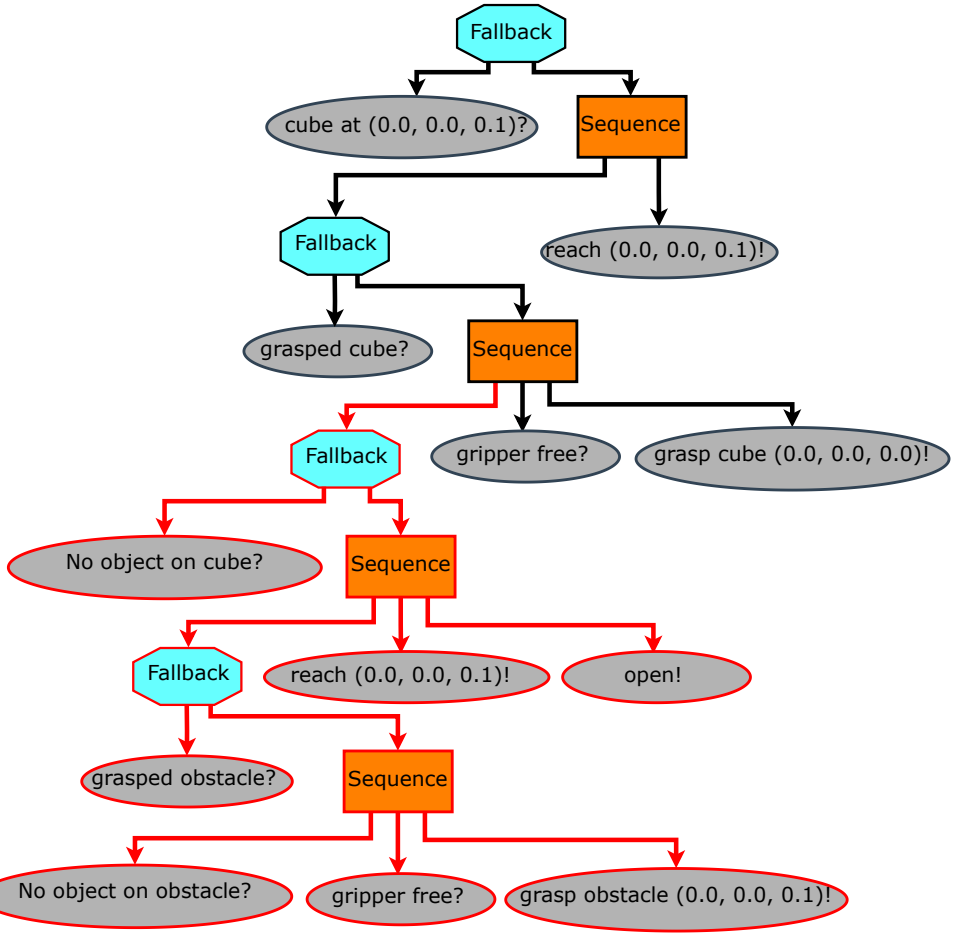


Figure 5.8: Comparison of Behavior Trees (BTs) for the lift task. The initial tree (top) lacks a check for a blue cube (obstacle) on the red cube. The VLM adds this missing condition as precondition of grasp skill, verifying if the target object is clear. Additionally, we also add "gripper free" check to ensure the gripper is empty. The updated BT (bottom) shows these changes in red.

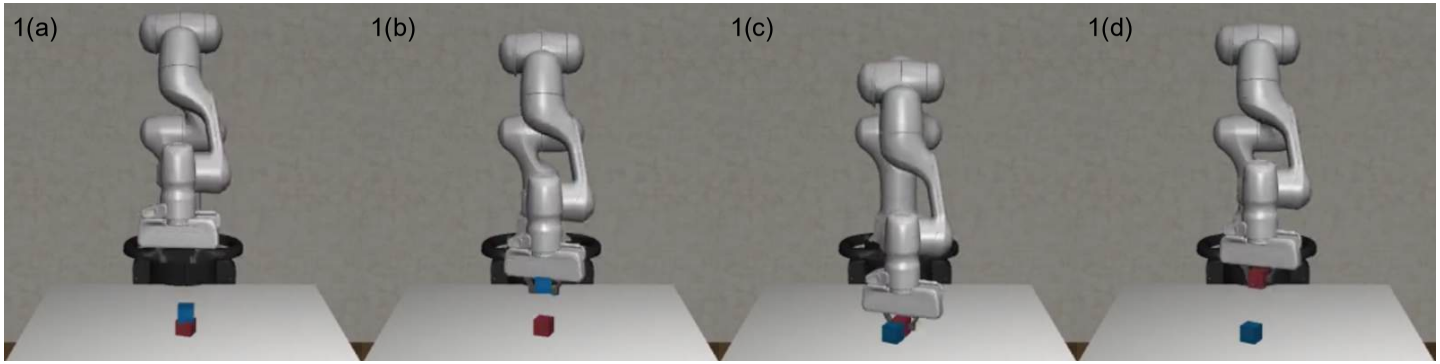


Figure 5.9: Scenes depicting the execution of the lift task, showing various stages from approach to successful lift.

Door Handle Opening Task

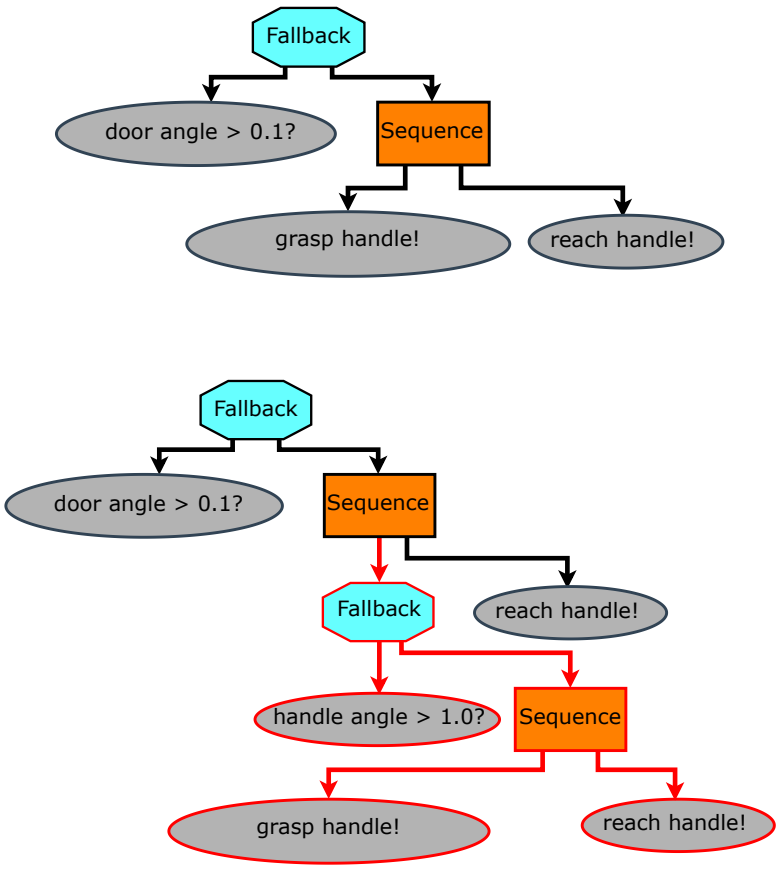


Figure 5.10: Comparison of Behavior Trees (BTs) for the door handle opening task. The initial tree (top) lacks a condition to check the handle angle, ensuring the handle is turned before opening. The VLM suggests adding this condition as a precondition for the reach skill, resulting in the modified tree (bottom) with changes shown in red.

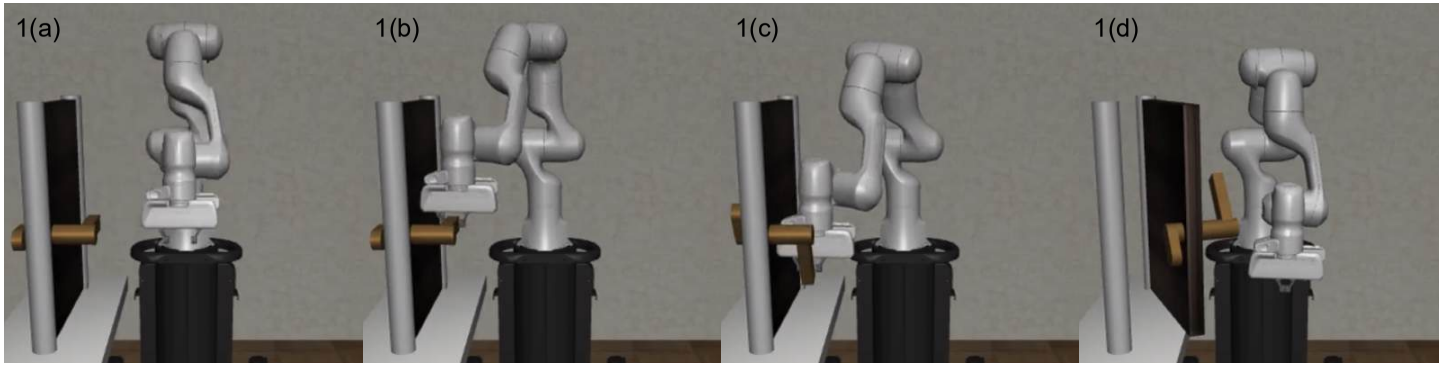


Figure 5.11: Scenes illustrating the stages of the door handle opening task, progressing from initial approach to successful handle operation.

Paper VI

F. Ahmad*, **H. Ismail***, J. Styrud, V. Krueger
A Unified Framework for Real-Time Failure Handling in Robotics Using Vision-Language Models, Reactive Planner and Behavior Trees
arXiv:2503.15202, 2025.

Chapter 6

A Unified Framework for Real-Time Failure Handling in Robotics Using Vision-Language Models, Reactive Planner and Behavior Trees

1 Abstract

Robotic systems often face execution failures due to unexpected obstacles, sensor errors, or environmental changes. Traditional failure recovery methods rely on predefined strategies or human intervention, making them less adaptable. This paper presents a unified failure recovery framework that combines Vision-Language Models (VLMs), a reactive planner, and Behavior Trees (BTs) to enable real-time failure handling. Our approach includes pre-execution verification, which checks for potential failures before execution, and reactive failure handling, which detects and corrects failures during execution by verifying existing BT conditions, adding missing preconditions and, when necessary, generating new skills. The framework uses a scene graph for structured environmental perception and an execution history for continuous monitoring, enabling context-aware and adaptive failure handling. We evaluate our framework through real-world experiments with an ABB YuMi robot on tasks like peg insertion, object sorting, and drawer placement, as well as in AI2-THOR simulator. Compared to using pre-execution and reactive methods separately, our approach achieves higher task success rates and greater adaptability. Ablation studies highlight the importance of VLM-based reasoning, structured scene representation, and execution history tracking for effective failure recovery in robotics.

2 Introduction

Modern robotic systems excel in controlled environments, but struggle with dynamic environments such as small batch manufacturing, particularly in handling execution failures [1]. Failures such as unexpected obstacles, sensor inaccuracies, or misaligned objects disrupt operations, causing costly delays [2]. Unlike repetitive, pre-planned tasks in large-scale production, small batch manufacturing demands adaptability to frequent task variations. Similarly, in collaborative assembly lines, where robots work alongside humans, real time failure handling is crucial for safe and efficient execution [3]. Developing autonomous failure recovery mechanisms that enable robots to detect, identify, and correct failures without human intervention is essential for improving reliability and reducing downtime [4].

To address these challenges, failure recovery methods range from learning based approaches that rely on data driven policies to structured execution frameworks designed for modular and interpretable decision making. Many learning based methods employ end-to-end architectures where robotic control policies are trained directly from data [5, 6]. While effective across diverse tasks, these methods often lack interpretability and verifiability, making them unsuitable for safety critical domains requiring robust, failure resistant execution especially in high stakes environments where errors can damage expensive equipment or disrupt operations.

Structured execution frameworks, such as Behavior Trees (BTs) [7], provide a modular framework for verification, adaptation, and efficient failure recovery. They define execution policies as hierarchical compositions of reusable skills [8], enabling fine-grained monitoring while ensuring compliance with safety standards [9]. Their modularity supports incremental recovery, avoiding the computational cost of full replanning [10]. While BTs can be manually designed, reactive planners automate their generation using a backchaining approach that selects skills based on preconditions and postconditions [11]. This allows robots to construct reactive execution policies that adapt to unexpected conditions in real-time without requiring full replanning.

In our prior work [12], we introduced a failure recovery framework that used a Vision-Language Model (VLM) for pre-execution plan verification. The system analyzed input skills, execution conditions, the planned BT, and pre-execution images to assess whether the plan contained sufficient knowledge for successful execution. If critical preconditions or required skills were missing, it suggested modifications to prevent execution errors, reducing failures caused by incomplete task knowledge. However, this approach did not account for failures arising during execution due to unforeseen disturbances, environmental changes, or hardware errors.

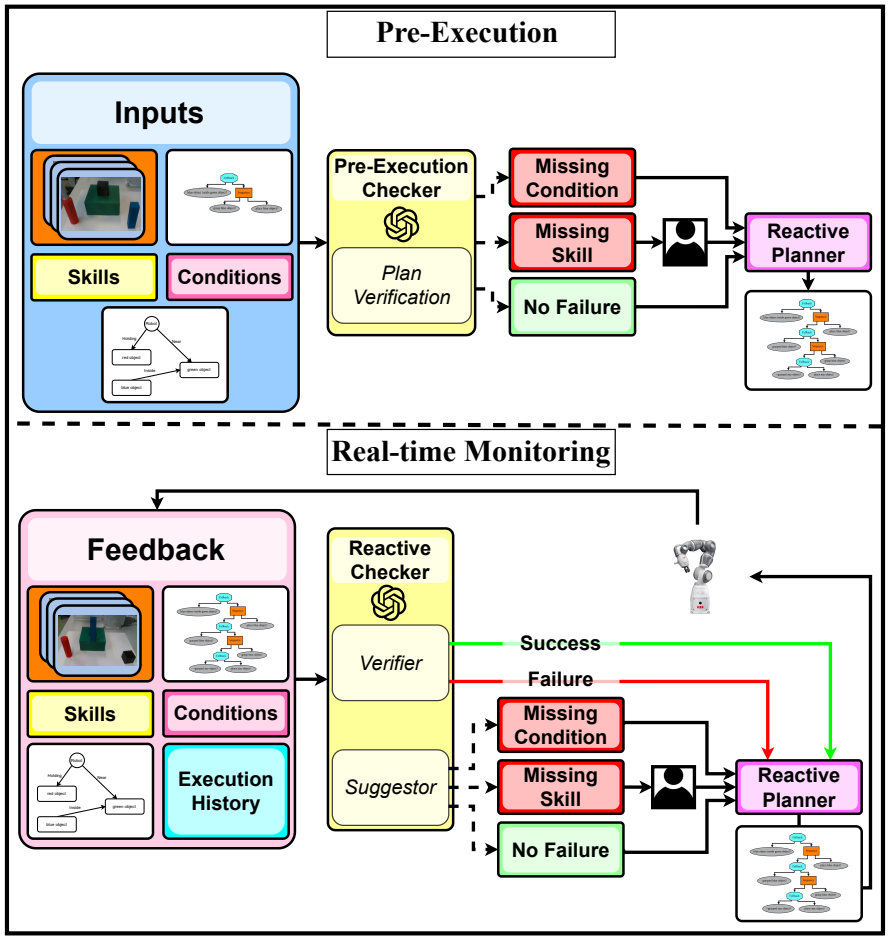


Figure 6.1: Overview of our approach, which consists of two phases: pre-execution verification and real-time monitoring. The pre-execution phase verifies the entire planned BT proactively using a VLM based on inputs (images, scene graphs, skills, and conditions). The real-time phase continuously monitors execution, where the VLM verifies preconditions, postconditions, and infers missing preconditions for individual skills using updated inputs and execution history. A reactive planner dynamically generates and adapts the BT as the robot's execution policy.

While pre-execution verification helps prevent many failures, it cannot predict all possible execution-time issues. A robot may generate a valid pick-and-place plan, yet unexpected events, such as human intervention or object displacement, can still cause grasp failures. Addressing such failures requires real-time monitoring and corrective actions, which is only possible through a reactive mechanism. Without continuous failure monitoring, robots cannot effectively detect and adapt to failures as they occur, making reactive checks essential

for robust autonomous execution.

Building on our prior work [12], this paper presents a unified failure recovery framework that extends pre-execution plan verification with real-time execution monitoring (Figure 6.1) to detect, identify, and correct errors dynamically. Our framework integrates reactive failure handling using a continuously updated execution history, which records skill execution states, timestamps, and scene graph updates for adaptive failure recovery. To improve situational awareness, we incorporate scene graphs that track object-object and robot-object spatial relationships throughout execution. Unlike [13], which generates scene graphs post-execution, our method updates them continuously, enabling immediate detection of environmental changes. Additionally, while [12] suggested missing skills only pre-execution, our approach supports both pre-execution and reactive skill suggestions, ensuring failures are addressed proactively and dynamically. This work makes the following key contributions:

- A unified failure recovery framework integrating Vision-Language Models (VLMs), reactive planners, and Behavior Trees (BTs) for pre-execution failure verification and real-time reactive failure handling.
- Real-time failure detection, identification, and correction using an incrementally updated execution history that tracks skill conditions, execution timestamps, and scene graph updates.
- Experimental validation in AI2-THOR [14] and a real-world ABB YuMi robot, demonstrating improved failure recovery across diverse environments.

3 Related Work

Failure recovery in robotics has been extensively studied, from predefined strategies to modern learning-based techniques and Large Language Models (LLMs) for adaptive failure handling. This section reviews these methodologies and highlights the distinctions between existing works and our approach.

3.1 Traditional Failure Recovery Strategies

Early methods relied on human intervention, predefined recovery strategies, and automated solutions based on failure mode analysis. While human-in-the-loop strategies offer flexibility, they are labor-intensive and limit scalability [15]. Predefined strategies handle known failure cases well but struggle with novel issues [16]. Systematic failure analysis, such as Failure Mode and Effects Analysis (FMEA), requires expert knowledge and does not generalize to dynamic environments [17]. Automated recovery methods attempt autonomy

but remain constrained by predefined failure modes [18, 19]. Unlike these approaches, our framework continuously updates a dynamic execution history for real-time failure detection and adaptation.

3.2 Learning-Based Failure Recovery

Recent approaches explore reinforcement learning (RL) and imitation learning (IL) to develop recovery strategies from experience [5, 6]. RL-based methods require extensive training in simulations, making real-world deployment difficult [20]. IL-based methods like RACER [21] improve recovery using demonstrations but struggle to generalize. Neuro-symbolic methods combine structured reasoning with learning, improving interpretability but facing scalability challenges [22, 23, 24]. Our approach avoids data-heavy training by leveraging Vision-Language Models (VLMs) for reasoning-based failure recovery, enabling flexible and context-aware corrections in real time.

3.3 Failure Recovery with Large Language Models (LLMs) and Vision-Language Models (VLMs)

LLMs and VLMs have become integral to robotic failure recovery due to their reasoning capabilities. Several approaches leverage LLM-based reasoning for failure detection and correction, including REFLECT [25], AHA [20], DoReMi [26], ReplanVLM [27], RECOVER [22], and Code-as-Monitor [28]. REFLECT provides hierarchical post-execution summaries but lacks real-time intervention. AHA fine-tunes a VLM for failure detection at task checkpoints but lacks structured execution policies. DoReMi enforces dynamic execution constraints but relies on LLM-generated constraints, introducing variability. ReplanVLM integrates pre-execution validation with execution monitoring using GPT-4V but depends on LLM-driven re-planning rather than structured failure handling.

Unlike these, our framework integrates a reactive planner and Behavior Trees (BTs) for structured, real-time failure handling at both pre-execution and reactive levels. *RECOVER*[22] uses ontology-driven neuro-symbolic reasoning for real-time failure detection but requires domain-specific engineering, limiting adaptability. *Code-as-Monitor*[28] translates natural language constraints into executable monitors for proactive (handling foreseeable failures) and reactive failure detection but lacks explicit recovery mechanisms. Unlike these, our execution history continuously updates skill execution states, enabling VLMs to analyze failures dynamically rather than post-execution. Compared to *AHA* and *ReplanVLM*, which focus on high-level reasoning or planning corrections, our approach ensures modular and adaptive failure recovery by integrating structured execution policies via BTs and a reactive planner. Additionally, recent work [29] explores intent-based BT planning using LLMs for goal interpretation, whereas our method actively modifies execution policies by suggest-

ing missing preconditions, postconditions, and skills in real time, ensuring robust failure recovery in dynamic environments.

4 Background

In this section, we discuss the relevant concepts that serve as background knowledge for the paper.

4.1 Behavior Trees

Behavior Trees (BTs) are a hierarchical execution model valued for their modularity, flexibility, and reactivity in robotic decision-making [30, 31]. Originally developed for game AI, BTs now provide interpretable and scalable task execution in robotics [9, 7]. Their structure simplifies behavior design, modification, and debugging while enabling real-time adaptation to dynamic environments [32].

A BT is a directed acyclic graph where execution begins at the root node, propagating tick signals to evaluate and execute behaviors dynamically. Nodes return *Success*, *Failure*, or *Running*, with control-flow nodes (e.g., *Sequence*, *Fallback*) managing execution order and execution nodes (e.g., *action*, *condition*) implementing robot skills. This structured execution enables task decomposition and fine-grained monitoring. Once adapted to handle a failure, the BT becomes a reusable execution policy, reducing reliance on model queries and improving efficiency over time.

4.2 Reactive Planner

Reactive planners generate Behavior Trees (BTs) dynamically using backchaining, selecting skills that satisfy goal conditions [33]. Starting from the goal, the planner works backward through skill preconditions and postconditions, iteratively expanding the BT until all conditions are met or a termination criterion is reached. This approach enables robots to adapt to environmental changes without full re-planning, leveraging BT modularity for flexible execution [11]. The PDDL-based reactive planner used in this work follows [11], ensuring efficiency by removing redundant nodes and introducing composite subtrees for complex tasks. This facilitates real-time, autonomous failure recovery while maintaining computational efficiency. As backchaining inherently selects skills that achieve required postconditions, explicit VLM-generated postcondition suggestions are unnecessary.

4.3 Vision-Language Models

Vision-Language Models (VLMs) combine visual perception with language-based reasoning, making them effective for robotic failure recovery [22, 21]. They enable robots to detect, identify, and correct failures by analyzing execution conditions and task states.

In our prior work [12], GPT-4o was used for pre-execution verification, where the VLM assessed if a planned execution contained sufficient knowledge to succeed. It performed three key tasks: failure detection (checking for potential failures based on available conditions), failure identification (diagnosing root causes by analyzing missing or incorrect preconditions), and failure correction (suggesting modifications such as adding missing preconditions or required skills). This approach reduced failures due to incomplete task knowledge but did not address execution-time failures from unforeseen disturbances or environmental changes.

This work extends VLMs to real-time execution monitoring and correction. The VLM continuously analyzes execution states, providing corrective suggestions based on evolving conditions. To improve reasoning, we integrate scene graphs that dynamically track object-object and robot-object relationships, improving failure detection. Additionally, an execution history records skill preconditions, postconditions, and execution timestamps, enabling structured failure analysis. By combining pre-execution checks with reactive real-time monitoring, our framework ensures continuous adaptation to failures, enhancing robustness in autonomous robotic execution.

5 Approach

To enable real-time robotic failure recovery, our framework integrates a reactive planner, Behavior Trees (BT), and Vision Language Models (VLM). The failure monitoring process is divided into pre-execution failure verification and real-time execution monitoring, each addressing failure detection, identification, and correction. Additionally, we extend the system with a scene graph and execution history to improve failure reasoning and adaptation. All failure handling mechanisms rely on the following key inputs:

- **Images** capturing the scene from multiple angles using two cameras (front and side views) to improve spatial understanding.
- **Skills** with predefined pre- and postconditions.
- **Known conditions** for environment reasoning.
- **Scene graph** representing spatial object relations.

- **Behavior Tree (BT)** defining execution policy.
- **Execution history** (real-time only) tracking past actions and scene updates.

Failure handling follows a three-phase process: *detection* identifies potential failures, *identification* determines the root cause by pinpointing the affected skill and unmet condition, and *correction* modifies the BT through precondition adjustments or skill additions to ensure successful execution. inspired by chain-of-thought [34] reasoning, we structure failure recovery prompt into these three phases. This improves the VLM performance by guiding it step-by-step toward the correct solution. If no failure is detected during the detection phase, the system skips the identification and correction steps, optimizing computational efficiency in both pre-execution and real-time monitoring.

To explain concretely our failure handling process, we use a peg-in-hole task, where the goal is to insert the blue object inside the green object, while red and black objects act as obstacles. Figures 6.4 and 6.5 illustrate different failure types with VLM responses. These figures also show various prompts, color-coded to distinguish between failure detection, identification, and correction questions posed to the VLM[1]. From here onward, we will consider a BT for peg-in-hole task execution that does not yet account for failures, as shown in Figure 6.2, unless specified otherwise.

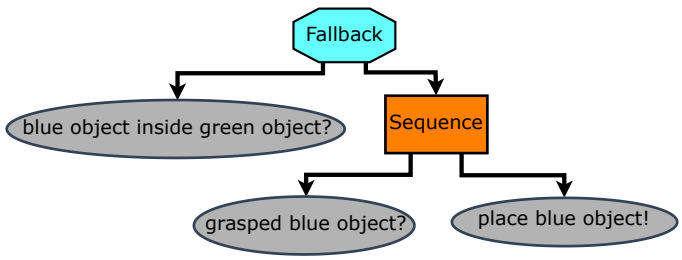


Figure 6.2: BT of the peg-in-hole task without failure handling

[1]Full prompts and code will be released after the submission process.

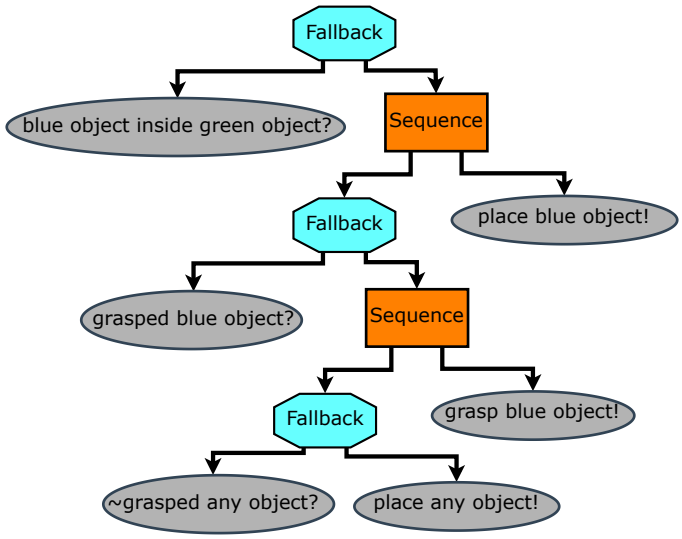


Figure 6.3: Extended BT execution where a missing precondition is added, ensuring the gripper is empty before grasping target object.

5.1 Pre-Execution Failure Verification

Before execution [12], we validate the planned BT by proactively checking for missing preconditions or potential execution failures. This step prevents errors before they occur, reducing failures caused by incomplete task knowledge. A GPT-4o-based VLM performs this verification by analyzing the inputs.

- **Detection**: Flags anomalies where the planned BT may fail based on the current scene. *For example, in the peg-in-hole task, if a black cube blocks the hole, the pre-execution checker detects a potential failure (Figure 6.4(a)).*
- **Identification**: Pinpoints the failing skill and the root cause, whether due to missing knowledge or an incorrect assumption. *In this case, the VLM identifies that the place skill will fail as the BT does not ensure the hole is unoccupied before placement (Figure 6.4(a)).*
- **Correction**: Suggests a missing precondition to update the BT and prevent failure. *Here, the system adds occupied(hole) as a precondition for place, prompting the reactive planner to remove the black cube before placement (Figure 6.4(a)).*

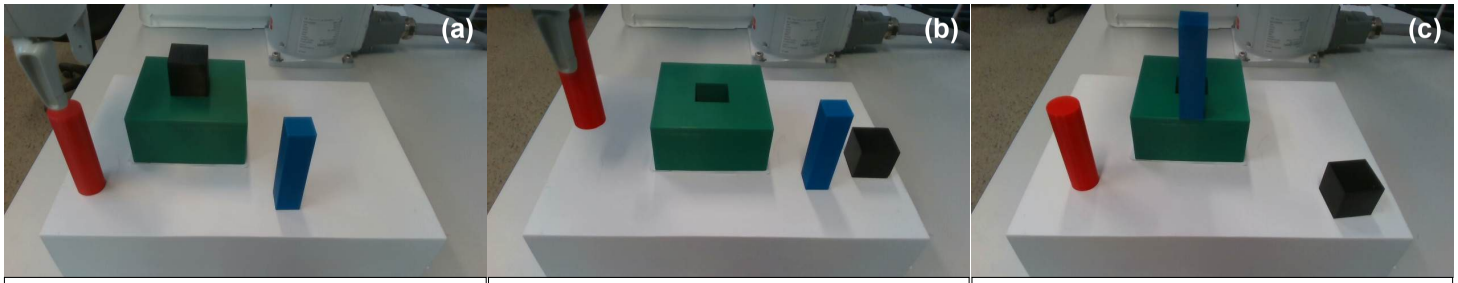


Figure 6.4: Three failure instances with corresponding VLM responses. (a) Pre-execution verification detects that the black object blocks the hole, and the VLM suggests adding the missing precondition for the place skill. (b) Precondition verification identifies that the grasp skill fails due to an unmet condition, as the robot is already holding a red object. (c) Postcondition verification detects a failed placement since the blue object is on top of the green object instead of inside. Failure detection (red), identification (orange), and correction (blue) are indicated with corresponding VLM responses in black.

5.2 Real-Time Failure Monitoring

While pre-execution verification minimizes failures, unexpected execution failures may still occur due to sensor inaccuracies, dynamic obstacles, or external disturbances. To handle these, we introduce a real-time failure monitoring module comprising a *Verifier* and a *Suggestor*. Both modules use the same inputs as pre-execution verification but incorporate continuously updated scene graphs, images, and execution history for improved reasoning.

5.2.1 Verifier

Ensures that execution aligns with expected conditions by performing *precondition verification before execution* and *postcondition verification after execution*.

Precondition Verification Before executing a skill, the *Verifier* checks if the skill preconditions hold. Consider the BT in Figure 6.3 with an existing `grasped any object` precondition in this case.

- **Detection**: Flags an anomaly if the preconditions for the skill in the BT are unmet. *For example, in the peg-in-hole task, if the robot has already grasped a red object but needs to grasp the blue object, the verifier detects an anomaly (Figure 6.4(b)). This failure can occur if a human intervenes after the pre-execution failure check by manually placing the red object inside the gripper.*
- **Identification**: Determines the violated precondition and the cause of failure. *In this case, it finds that the* `grasped any object` *precondition of the grasp skill is not satisfied (Figure 6.4(b)).*
- **Correction**: Prevents execution by marking relevant preconditions as unsatisfied. The reactive planner will then automatically expand the BT to satisfy the marked preconditions. *For instance, the BT adapts by placing the currently held object before attempting the new grasp (Figure 6.4(b)).*

Postcondition Verification After executing a skill, the *Verifier* checks if expected postconditions hold.

- **Detection**: Flags an anomaly if the executed skill fails to meet its postconditions. *For instance, if the robot places the blue object on top of the hole instead of inside, the verifier detects a failure (Figure 6.4(c)).*

- **Identification**: Identifies the violated postcondition and the cause of failure. *Here, it finds that the "inside" condition is violated because the object is on top rather than inside (Figure 6.4(c)).*
- **Correction**: Returns *Failure*, triggering the reactive planner to adjust execution dynamically. *The BT reattempts placement in the next tick (Figure 6.4(c)).*

5.2.2 Suggestor

The *Suggestor* dynamically infers missing preconditions when a skill fails due to unmet conditions.

- **Detection**: Flags an anomaly when a skill is likely to fail due to an unmet precondition. *For example, in the peg-in-hole task, the red object is already occupying the hole (Figure 6.5(a)).*
- **Identification**: Identifies the missing precondition and the cause of failure. *In this case, it determines that the place skill is missing a precondition ensuring the hole is empty before insertion (Figure 6.5(a)).*
- **Correction**: Suggests the missing precondition, prompting the BT to update accordingly. *The model suggests occupied(hole) as a precondition, allowing the reactive planner to expand the BT accordingly (Figure 6.5(a)).*

5.3 Skill Addition

While modifying preconditions can resolve many failures, some cases require introducing new skills. The skill addition can be suggested either pre-execution or reactively depending on when the potential failure case arises. The pre-execution stage implements our prior work in the [12] paper. if no existing skill can address a detected failure, the system suggests a missing skill (Figure 6.5(b)). In the reactive phase, the VLM checks execution feasibility before executing every skill. If the current skill is predicted to fail, a missing skill is suggested to remove the failure (Figure 6.5(c)).

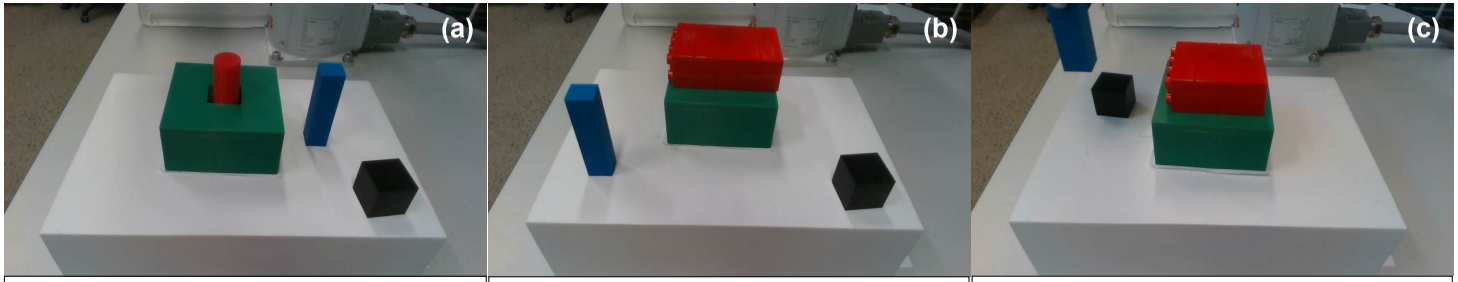


Figure 6.5: The figure illustrates three failure scenarios and corresponding VLM responses. (a) Precondition suggestor: The red object inside the green object leads the VLM to identify a missing precondition for the place skill. (b) Pre-execution missing skill generation: The VLM identifies the need for a push skill to remove the red object. (c) Real-time missing skill generation: The VLM suggests generating the push skill during execution. Failure detection (red), identification (orange), and correction (blue) phases are depicted, with VLM responses in black.

- **Detection**: Identifies when the available skills can not resolve a failure. *For example, in the peg-in-hole task, if a non-pickable object blocks the hole and the system detects an unresolved failure (Figure 6.5(b)).*

- **Identification**: Determines the missing capability and the skill that fails due to this limitation. *In this case, the pick skill fails because the object is non-pickable (Figure 6.5(b)).*

- **Correction**: The VLM suggests a new skill to resolve the failure, ensuring compatibility with the robot's world model. The suggestion includes:

  – The name of the missing skill.

  – A code template defining the skill.

  – Predefined preconditions and postconditions.

  For example, if a robot cannot grasp an object, the VLM may suggest a "Push" skill as an alternative, providing a skill description with predefined conditions (Figure 6.5(b)). Figure 6.5(c) illustrates the reactive version occurring during execution, where the robot first places the blue object on the table before executing the "Push" skill to move the red object. To ensure consistency, the system restricts the VLM to known world model conditions, preventing arbitrary condition generation.

LLMs can reason but are unreliable for planning due to hallucinations and nondeterminism. Vision-language-action models generate short-horizon actions with a few exceptions but often require large datasets and offer limited support for structured failure recovery. Our reactive planner ensures valid plans through backchaining over grounded conditions, enabling modular and traceable recovery without retraining.

5.4 Scene Graph Representation

To enable real-time monitoring, our system maintains an evolving scene graph that tracks spatial relationships between objects and the robot. Unlike REFLECT [13], which regenerates the scene graph from scratch at each timestep, our approach continuously updates it by modifying relationships and adding or removing nodes as needed.

The scene graph is constructed using:

- **RGB-D images and point clouds** for capturing scene depth and object positioning.

- **Grounding DINO** [35] for object detection and **SAM2** [36] for instance segmentation and tracking.

- **RANSAC and PCA-based pose estimation** to estimate 6D object poses.

Continuous updates improve efficiency and ensure execution consistency. *For example, in the peg-in-hole task, when the robot inserts the blue object into the green one, our system updates the scene graph by modifying the "on" relation to "inside" without reconstructing the entire graph.*

5.5 Execution History

The execution history maintains a log of skill executions, condition verification results, and environmental changes. Instead of explicit failure logging, which assumes perfect execution state knowledge, our approach captures execution traces via changes in the scene graph that help infer failures and inconsistencies.

- **Skill execution records**: Logs executed skills with timestamps.
- **Precondition and postcondition verification**: Tracks whether preconditions were met before execution and if postconditions held afterward.
- **Scene graph updates**: Records object positions and relationships before and after execution to analyze deviations.

For example, in the peg-in-hole task, if the blue peg is placed on top of the green hole instead of inside, the execution history logs the "Place" skill execution with its timestamp. The system records that the precondition was satisfied (e.g., the peg was grasped), but postcondition verification fails as the peg's spatial relation does not match the expected "inside" condition. The scene graph update reflects this deviation, showing the peg as "on" rather than "inside" the hole.

This structured history enables real-time adaptation by detecting execution anomalies, allowing the system to refine failure handling based on observed task progression.

6 Experiments and Results

We evaluate our failure recovery framework through both simulation benchmarks and real-world experiments. In simulation, we use benchmark tasks from REFLECT [13] in AI2-THOR [14], assessing how our system adapts to predefined failure cases. For real-world validation, we implement our framework on a robotic platform to evaluate its effectiveness in handling failures in physical environments.

6.1 Simulation Experiments

We evaluate our framework on REFLECT benchmark tasks, where failures occur during execution and are corrected post-execution using hierarchical summaries and scene graphs [25]. However, REFLECT lacks real-time adaptation, as failures are only detected and corrected after task completion.

Our approach instead uses a reactive planner and BTs to dynamically generate execution policies, enabling real-time monitoring and immediate failure correction. Unlike REFLECT, which reconstructs a new scene graph per execution, our system continuously updates it. Additionally, while REFLECT relies on LLM-generated post-execution corrections without correctness guarantees, our reactive planner ensures correctness through structured preconditions and postconditions.

We successfully applied our framework to all REFLECT benchmark tasks, achieving a 100% success rate across multiple runs. Real-time monitoring was sufficient, making pre-execution checks unnecessary. The *Verifier* ensured execution correctness, while the *Suggestor* resolved missing preconditions. Unlike REFLECT, which evaluates explanation, localization, and replanning success, we assess overall task completion. Since failures are proactively verified and reactively corrected during execution, post-execution replanning is unnecessary, reducing reliance on retrospective reasoning.

Key differences between REFLECT and our approach are summarized in Table 6.1.

Table 6.1: Qualitative Comparison of REFLECT and Our Approach

Feature	REFLECT	Our Approach
Execution Plan	Manually designed	Reactive BT
Failure Handling	Post-execution	Real-time
Scene Graph Update	Reconstructed post-execution	Maintained incrementally
Failure Detection	Post-execution	Real-time
Plan Correction	LLM-generated	Reactive BT

6.2 Real-World Experiments

For real-world validation, we deployed our framework on an ABB YuMi robot equipped with an RGB-D camera. We assessed its failure recovery capabilities across three tasks:

- **Peg-in-hole**: Inserting a peg into a hole with varying initial placements.
- **Object Sorting**: Sorting objects by color into designated locations.
- **Drawer Placement**: Placing an object inside a drawer.

Table 6.2: Comparison of Failure Recovery Baselines

Metric	Pre-execution	Reactive	Pre-execution + Reactive (Ours)
Task Success Rate	31.25%	100%	**100%**
Failure Detection Rate	31.25%	100%	**100%**
Failure Identification Rate	100%	100%	**100%**
Correction Success Rate	100%	100%	**100%**
Skill Suggestion Accuracy	50%	100%	**100%**

Failures were introduced by modifying object placements, adding obstructions, or altering task constraints. Additionally, human intervention was used to induce failures during execution.

6.2.1 Baseline Approaches

We compared our approach against two baselines to assess the benefits of integrating pre-execution and reactive failure recovery mechanisms:

- **Pre-execution**: Check for plan verification [12].
- **Reactive**: Detect and correct failures during execution.
- **Our Approach (Pre-execution + Reactive)**: Combine pre-execution validation and real-time monitoring to prevent and correct failures dynamically.

Table 6.2 provides a quantitative comparison, showing that *Pre-execution + Reactive* achieves the highest performance. *Reactive* only approach matches its failure handling but incurs higher computational cost due to more VLM queries, additional skills, and longer execution times. For instance, without pre-execution checks, a robot may begin execution only to discover a missing object mid-task, requiring backtracking and reactive correction. In contrast, pre-execution checks efficiently catch static or predictable failures (e.g., a blocked hole), avoiding wasted actions. However, they miss dynamic issues revealed only during execution (e.g., a hidden drawer obstruction). Reactive monitoring handles such failures by verifying conditions step-by-step. Although, the reactive only method is robust, they are expensive. Combining both modes is more efficient.

6.2.2 Evaluation Metrics

We evaluated our framework's ability to detect, identify, and correct failures across 16 pre-recorded failure cases, repeating each experiment 10 times. To assess false positives, we also ran each task 10 times without introducing failures.

We measured the system's accuracy in detecting failures, correctly identifying their root causes, and successfully correcting them. Additionally, we analyzed the proportion of pre-execution failures handled versus those requiring real-time intervention and assessed the accuracy of skill suggestions.

Table 6.2 summarizes the performance across these metrics. Our framework achieved a perfect 100% accuracy on all tasks, demonstrating strong failure recognition and reasoning capabilities. No false positives occurred when running the tasks without failures. Given that failure recovery systems are designed to achieve near-perfect accuracy, these results align with expectations. Future work should focus on evaluating the framework on more complex benchmarks to further assess its scalability and robustness.

6.2.3 Ablation Studies and Summary of Findings

To evaluate key components, we conducted ablation studies by selectively removing elements and analyzing their impact on failure recovery.

- **VLM vs. LLM**: Removing vision input weakens spatial and scene-aware failure detection, limiting object relation reasoning. Success drops from 100% (2 images) to 98% (1 image) and 95% (no images), assuming scene graph accuracy, which is not always guaranteed.
- **Scene Graph Contribution**: Assists spatial reasoning and removes scene ambiguity. Without it, success drops to 91.25%, highlighting its role in structured failure prediction.
- **Execution History Effectiveness**: Omitting execution history tracking did not significantly impact results, as we observed similar success rates with and without it. However, this does not imply that execution history is ineffective; its benefits may become more evident in more complex benchmarks.

Our findings confirm that combining pre-execution and reactive failure handling improves task success. Pre-execution checks prevent plan failures, while real-time monitoring improves adaptability. VLM-based reasoning strengthens failure detection and correction,

and scene graphs with execution tracking improve system reliability by maintaining structured environmental context. These results validate our framework's effectiveness in autonomous failure recovery across diverse robotic tasks.

7 Conclusion and Future Work

This paper presented a unified failure recovery framework integrating VLMs, a reactive planner, and Behavior Trees (BTs) for pre-execution failure detection and reactive recovery in robotic execution. By incorporating a scene graph for structured perception and execution history for real-time monitoring, our approach dynamically adapts to failures, minimizing execution disruptions. Experiments with an ABB YuMi robot and in simulation showed that combining pre-execution and reactive strategies outperforms either alone. Ablation studies confirmed the value of VLMs, scene understanding, and execution summaries for reliability.

Although, our evaluated tasks are short-horizon, they feature dynamic and occluded conditions, making them realistic for recovery testing. Future work will address more complex, multi-step tasks in cluttered environments to assess scalability. We also plan to benchmark against models like ReplanVLM to evaluate trade-offs in accuracy, robustness, and interpretability, as few baselines support both long-horizon planning and structured failure recovery.

In the future, we aim to enhance our framework by integrating video and audio inputs for improved context-aware task monitoring. We plan to fine-tune open-source multi-modal models for failure handling, reducing computational costs and improving efficiency. Additionally, we will leverage Vision-Language Action (VLA) models for autonomous skill generation with structured preconditions and postconditions, ensuring quality through static and integration checks. To extend real-time monitoring, we will incorporate holding conditions for proactive failure checking during execution. These advancements will enhance autonomous failure recovery, making robotic systems more adaptable and self-sufficient.

8 Acknowledgements

We thank Jialong Li for valuable discussions. This work was supported by the Wallenberg AI, Autonomous Systems, and Software Program (WASP) through the Knut and Alice Wallenberg Foundation and by Vinnova (NextG2Com, ref. no. 2023-00541). Experiments were partly conducted at ABB Corporate Research Center, Västerås, Sweden, with financial support from WASP. Generative AI tools were used for editing, including grammar and sentence structuring.

References

[1] Malin Löfving et al. "Evaluation of flexible automation for small batch production". In: *Procedia Manufacturing* 25 (2018). Proceedings of the 8th Swedish Production Symposium (SPS 2018), pp. 177–184. ISSN: 2351-9789. DOI: https://doi.org/10.1016/j.promfg.2018.06.072. URL: https://www.sciencedirect.com/science/article/pii/S2351978918305912.

[2] Ruikai Liu et al. "Autonomous Robot Task Execution in Flexible Manufacturing: Integrating PDDL and Behavior Trees in ARIAC 2023". In: *Biomimetics* 9.10 (2024). ISSN: 2313-7673. DOI: 10.3390/biomimetics9100612. URL: https://www.mdpi.com/2313-7673/9/10/612.

[3] Ekansh Sharma et al. *Adaptive Compliant Robot Control with Failure Recovery for Object Press-Fitting*. 2023. arXiv: 2307.08274 [cs.RO]. URL: https://arxiv.org/abs/2307.08274.

[4] Ruichao Wu, Sitar Kortik and Christoph Hellmann Santos. "Automated Behavior Tree Error Recovery Framework for Robotic Systems". In: *2021 IEEE International Conference on Robotics and Automation (ICRA)*. 2021, pp. 6898–6904. DOI: 10.1109/ICRA48506.2021.9561002.

[5] Shoki Kobayashi and Takeshi Shibuya. "Reinforcement Learning to Efficiently Recover Control Performance of Robots Using Imitation Learning After Failure". In: *2022 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*. 2022, pp. 1147–1154. DOI: 10.1109/SMC53654.2022.9945538.

[6] Jonathan Booher et al. *CIMRL: Combining IMitation and Reinforcement Learning for Safe Autonomous Driving*. 2024. arXiv: 2406.08878 [cs.LG]. URL: https://arxiv.org/abs/2406.08878.

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

[8] Faseeh Ahmad et al. "Adaptable Recovery Behaviors in Robotics: A Behavior Trees and Motion Generators (BTMG) Approach for Failure Management". In: *2024 IEEE 20th International Conference on Automation Science and Engineering (CASE)*. IEEE. 2024, pp. 1815–1822.

[9] 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 (TCPS)* 6.2 (2022), pp. 1–36.

[10] A. Marzinotto et al. "Towards a Unified Behavior Trees Framework for Robot Control". In: *2014 IEEE International Conference on Robotics and Automation (ICRA)*. 2014 IEEE International Conference on Robotics and Automation (ICRA). 2014, pp. 5420–5427. DOI: 10.1109/ICRA.2014.6907656.

[11] Jonathan Styrud et al. “BeBOP–Combining Reactive Planning and Bayesian Optimization to Solve Robotic Manipulation Tasks”. In: *2024 International Conference on Robotics and Automation (ICRA)*. IEEE, 2024.

[12] Faseeh Ahmad, Jonathan Styrud and Volker Krueger. “Addressing failures in robotics using vision-based language models (VLMs) and behavior trees (BTs)”. In: *arXiv preprint arXiv:2411.01568* (2024). Accepted at European Robotics Forum (ERF) 2025.

[13] Zeyi Liu, Arpit Bahety and Shuran Song. “REFLECT: Summarizing Robot Experiences for Failure Explanation and Correction”. In: *arXiv preprint arXiv:2306.15724* (2023).

[14] Eric Kolve et al. “AI2-THOR: An Interactive 3D Environment for Visual AI”. In: *arXiv* (2017).

[15] Shunki Itadera and Yukiyasu Domae. *Motion Priority Optimization Framework towards Automated and Teleoperated Robot Cooperation in Industrial Recovery Scenarios*. 2024. arXiv: 2308.15044 [cs.RO]. URL: https://arxiv.org/abs/2308.15044.

[16] Ruichao Wu, Sitar Kortik and Christoph Hellmann Santos. “Automated Behavior Tree Error Recovery Framework for Robotic Systems”. In: *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2021, pp. 6898–6904.

[17] FN Jusuf et al. “Review on defenses against common cause failures on digital safety system”. In: *AIP Conference Proceedings*. Vol. 2374. 1. AIP Publishing. 2021.

[18] Yumeng Lei et al. “Artificial Intelligence Planning of Failure Recovery Strategies in Discrete Manufacturing Automation”. In: *2023 IEEE 19th International Conference on Automation Science and Engineering (CASE)*. IEEE. 2023, pp. 1–8.

[19] Lucas VR Alves and Patrícia N Pena. “Secure recovery procedure for manufacturing systems using synchronizing automata and supervisory control theory”. In: *IEEE Transactions on Automation Science and Engineering* 19.1 (2020), pp. 486–496.

[20] Jiafei Duan et al. *AHA: A Vision-Language-Model for Detecting and Reasoning Over Failures in Robotic Manipulation*. 2024. arXiv: 2410.00371 [cs.RO]. URL: https://arxiv.org/abs/2410.00371.

[21] Yinpei Dai et al. *RACER: Rich Language-Guided Failure Recovery Policies for Imitation Learning*. 2024. arXiv: 2409.14674 [cs.RO]. URL: https://arxiv.org/abs/2409.14674.

[22] Cristina Cornelio and Mohammed Diab. *Recover: A Neuro-Symbolic Framework for Failure Detection and Recovery*. 2024. arXiv: 2404.00756 [cs.AI]. URL: https://arxiv.org/abs/2404.00756.

[23] Oualid Bougzime et al. *Unlocking the Potential of Generative AI through Neuro-Symbolic Architectures: Benefits and Limitations*. 2025. arXiv: 2502.11269 [cs.AI]. URL: https://arxiv.org/abs/2502.11269.

[24] Xin Zhang and Victor S. Sheng. *Neuro-Symbolic AI: Explainability, Challenges, and Future Trends*. 2024. arXiv: 2411.04383 [cs.AI]. URL: https://arxiv.org/abs/2411.04383.

[25] Zeyi Liu, Arpit Bahety and Shuran Song. *REFLECT: Summarizing Robot Experiences for Failure Explanation and Correction*. 2023. arXiv: 2306.15724 [cs.RO]. URL: https://arxiv.org/abs/2306.15724.

[26] Yanjiang Guo et al. *DoReMi: Grounding Language Model by Detecting and Recovering from Plan-Execution Misalignment*. 2024. arXiv: 2307.00329 [cs.RO]. URL: https://arxiv.org/abs/2307.00329.

[27] Jiaqi Wang et al. *Large Language Models for Robotics: Opportunities, Challenges, and Perspectives*. 2024. arXiv: 2401.04334 [cs.RO]. URL: https://arxiv.org/abs/2401.04334.

[28] Enshen Zhou et al. *Code-as-Monitor: Constraint-aware Visual Programming for Reactive and Proactive Robotic Failure Detection*. 2024. arXiv: 2412.04455 [cs.RO]. URL: https://arxiv.org/abs/2412.04455.

[29] Xinglin Chen et al. "Integrating intent understanding and optimal behavior planning for behavior tree generation from human instructions". In: *arXiv preprint arXiv:2405.07474* (2024).

[30] Michele Colledanchise and Petter Ögren. *Behavior trees in robotics and AI: An introduction*. CRC Press, 2018.

[31] Matteo Iovino et al. *A Survey of Behavior Trees in Robotics and AI*. 2020. arXiv: 2005.05842 [cs.RO].

[32] Jonathan Styrud et al. *Automatic Behavior Tree Expansion with LLMs for Robotic Manipulation*. 2024. arXiv: 2409.13356 [cs.RO]. URL: https://arxiv.org/abs/2409.13356.

[33] 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)*. 2019, pp. 8839–8845. DOI: 10.1109/ICRA.2019.8794128.

[34] Jason Wei et al. "Chain of thought prompting elicits reasoning in large language models". In: *Advances in neural information processing systems* (2022).

[35] Shilong Liu et al. *Grounding DINO: Marrying DINO with Grounded Pre-Training for Open-Set Object Detection*. 2024. arXiv: 2303.05499 [cs.CV]. URL: https://arxiv.org/abs/2303.05499.

[36] Nikhila Ravi et al. *SAM 2: Segment Anything in Images and Videos*. 2024. arXiv: 2408.00714 [cs.CV]. URL: https://arxiv.org/abs/2408.00714.