Digital Shadows for Industrial Robotic Agents in the World Wide Lab

Von der Fakultät für Mathematik, Informatik und Naturwissenschaften
der RWTH Aachen University zur Erlangung des akademischen Grades
eines Doktors der Naturwissenschaften genehmigte Dissertation

vorgelegt von

Mohamed Behery, M.Sc.
aus
Dschidda, Saudi-Arabien

Berichter:
Prof. Gerhard Lakemeyer, Ph.D.
Univ.-Prof. Dr.-Ing. Christian Brecher

Tag der mündlichen Prüfung:
05. Dezember 2024

Keywords: Digital Shadows, World Wide Lab, Industrial Robotics, Behavior Trees.

In accordance with the requirements of the degree of Doktors der Naturwissenschaften in the Fakultät für Mathematik, Informatik und Naturwissenschaften, I present the following thesis entitled,

Digital Shadows for Industrial Robotic Agents in the World Wide Lab

This work was performed under the supervision of Prof. Gerhard Lakemeyer, Ph.D. I declare that the work submitted in this thesis is my own, except as acknowledged in the text and footnotes, and has not been previously submitted for a degree at RWTH Aachen University or any other institution.

Mohamed Behery

"Perpetuation with purpose is the way to victory."

James Luceno
Darth Plagueis, 2012

Acknowledgements

Many people helped make this happen. Some through advice and guidance, others through emotional support, and some mostly by sitting through my ranting monologues. I hope I am expressive enough to give credit where credit is due.

I would like to thank my professor and supervisor Gerhard Lakemeyer for taking me in as a student and for giving me his support for over half a decade. I would also like to thank Christian Brecher for being my second reader and his group for opening their lab for our joint experiments. Many thanks go additionally to Bastian Leibe for agreeing to join my examination committee and Wil van der Aalst for agreeing to chair the committee.

I am also grateful to my current and former colleagues for providing a friendly work environment. I appreciate the relaxed and friendly world state created by Tarik Viehmann, Qihui Feng, Till Hofmann, Hayyan Helal, and Daxin Liu. Even though we were never at the KBSG at the same time, I really appreciate the one discussion I had with Christoph Schwering, where he told me the story of his Ph.D. and how he was able to finish after all, this gave me the motivation I needed at the time to keep going. I would like to thank Martin Liebenberg for always taking the time for our burger-based discussions and being ready to listen to my rants. Thanks to Leany Maaßen for making life easier by always giving me the time and guiding me through the sea of administrative tasks that accompanied most of my work at the KBSG. Special thanks go to Stefan Schiffer for guiding me through the intricacies and dynamics of academia early on in my career. The extended discussions we had helped shape my academic persona. I also express my gratitude to Philipp Brauner for guiding me through my newly acquired leading position in the cluster of excellence. I am grateful for all *quick* calls we used to manage the work stream, brainstorm, or write. I appreciate all the fruitful discussions I had with everyone from the cluster, whose enumeration is only precluded by the lack of space. I only hope my impact on their lives and careers is as positive as their impact on mine. I'm thankful to all my students, with whom the discussions were always interesting.

I have been working on this for a long time. I could not have made it without the huge support I received from my friends and family. I would like to thank Camellia Ibrahim for always lending a listening ear and never offering me coffee, I hope she never does. I am also grateful to her for introducing me to Lisa Mochalski, who is a great source of motivation and fun. They are both great friends, who certainly make life more enjoyable. I would also like to thank my friends, Abdelrahman Abdelkawy, Mostafa Ashraf, Ahmad Elkhashap, Moumen Elteir, Mohamed Emad, and Mohamed Shaaban, for all the discussions we had about life, career, and many other things. Many thanks also go to Rana Sawahel, Seif and Adham Raslan, Hana Elkabbany, Nadeen and Mirna Helmy, Rowan Elmasry, Ingy Sherif, Hoda Soliman, and Rofaida Daabis for all the support, motivation, and positivity I received from them throughout the years.

Last but not least, I'm grateful for my brothers' and my parents' continuous, abundant, and unwavering support.

Abstract

The digital transformation of production has become imperative to increase efficiency, adaptability, and innovation required to face the rapidly changing market requirements, supply chain disruptions, and global competition. In addition to initiatives such as Industry 4.0 and the Industrial Internet of Things (IIoT), the Internet of Production (IoP) aims to establish a worldwide network of research and industrial partners through the World Wide Lab (WWL). Such a network requires standardized communication protocols, which need to remain flexible to accommodate the different standards, use cases, and contexts. To this end, the WWL uses Digital Shadows (DSs) to represent the different entities: parts, products, machines, processes, and companies. DSs are context and purpose-specific representations of entities in the WWL, which enables them to replace the omnipresent Digital Twins by representing different aspects of the denoted entities. The thesis offers a road map for achieving the WWL as an application on top of the IoP.

The thesis focuses on industrial robotics processes, which involve teleoperated robots or human-robot collaborative assembly and are not fully integrated in the WWL as they lack a standard DS for task representation. To bridge this gap, the thesis proposes using Behavior Trees (BTs), which offer a flexible, modular, and composable task representation for industrial robotics processes. BTs emerged in the video game industry and were adopted by the roboticists but do not yet fulfil all the requirements to be a standard DS. The presented work offers a plan to fulfill these requirements.

The thesis introduces a framework for extracting and creating DSs for robot teleoperation tasks in industrial processes. It starts by collecting data to extract discrete actions. After that, it learns a library of BTs that represent the commonly executed behaviors. This allows a user to compose more complex behaviors using the learned BTs. The different modules of the framework are evaluated qualitatively and quantitatively. The thesis also proposes and demonstrates three BT extensions increasing BT flexibility to accommodate more use cases. They allow a BT to handle highly dynamic production environments, where it interacts with heterogeneous agents such as humans or robots with different capabilities or knowledge.

The dissertation concludes with a discussion that evaluates the applicability of the proposed framework and extensions to different industrial scenarios giving recommendations for future work.

Zusammenfassung

Die digitale Transformation der Produktion ist notwendig, um Effizienz, Anpassungsfähigkeit und Innovation angesichts der sich schnell ändernder Marktanforderungen, Lieferkettenunterbrechungen und des globalen Wettbewerbs zu steigern. Neben verwandten Initiativen wie Industrie 4.0 und dem ,,Industrial Internet of Things" (IIoT) verfolgt das ,,Internet of Production" (IoP) das Ziel, ein weltweites Netzwerk von Forschungs- und Industriepartnern über das ,,World Wide Lab" (WWL) zu etablieren. Dieses Netzwerk erfordert standardisierte Kommunikationsprotokolle, die flexibel genug bleiben sollen, um die verschiedenen Standards, Anwendungen und Kontexte unterzubringen. Zu diesem Zweck nutzt das WWL ,,Digital Shadows" (DSs), um die verschiedenen Entitäten wie Teile, Produkte, Maschinen, Prozesse, und Unternehmen abzubilden. DSs sind kontext- und zweckspezifische Repräsentationen von Entitäten im WWL, was ihnen ermöglicht, die weit verbreiteten ,,Digital Twins" durch gezielte Darstellungen unterschiedlicher Aspekte der Entitäten zu ersetzen. Diese Dissertation bietet einen Leitfaden zur Erzielung des WWL als Anwendung des IoP.

Die vorliegende Dissertation befasst sich mit industriellen Robotikprozesse, die teleoperierte Roboter oder der Montage mittels Mensch-Roboter-Kollaboration umfassen und nicht vollständig im WWL integriert sind, da ihnen ein standardisierte DS zur Aufgabenbeschreibung fehlt. Um diese Lücke zu schließen, stellt diese Arbeit die Verwendung von ,,Behavior Trees" (BTs) vor, die eine flexible, modulare, und zusammensetzbare Aufgabendarstellung für verschiedene industrielle Robotikprozessen anbieten. BTs sind in der Spieleindustrie entstanden und wurden später von Robotikern übernommen, aber sie erfüllen noch nicht alle Voraussetzungen, um Standard-DS der Produktion zu werden. Die Arbeit präsentiert einen Plan zur Erfüllung dieser Voraussetzungen.

In dieser Dissertation wird ein Framework zur Extraktion und Erstellung von der DS teleoperierter Roboter in industriellen Prozesse vorgestellt. Der Ansatz beginnt mit der Datenerhebung, um diskrete Aktionen zu identifizieren. Anschließend wird eine Bibliothek von BTs aufgebaut, die häufig verwendete Verhaltensmuster abbilden. Das ermöglicht dem Nutzer, durch den Gebrauch der erlernten BTs, komplexere Verhaltensweisen zu kombinieren und zusammenzusetzen. Die verschiedenen Module des Frameworks werden sowohl qualitativ als auch quantitativ evaluiert. Darüber hinaus werden drei Erweiterungen der BTs konzipiert und demonstriert, die deren Flexibilität erhöhen, um zusätzliche Anwendungsfälle abzudecken. Diese Erweiterungen ermöglichen die Anwendung in hochdynamischen Produktionsumgebungen, in denen eine Interaktion mit heterogenen Akteuren wie Menschen oder Robotern mit unterschiedlichen Fähigkeiten und Wissen erforderlich ist.

Diese Dissertation schließt mit einer Dikussion ab, die die Anwendbarkeit des vorgestellten Frameworks und der BT-Erweiterungen in verschiedenen industriellen Szenarien evaluiert und Empfehlungen gibt für zukünftige Forschungsarbeiten.

Contents

List of Figures

List of Tables

List of Algorithms

List of Acronyms

AC	Active Constraint	105
ADL	Activity of Daily Living	114
AI	Artificial Intelligence	146
AMP	Advanced Manufacturing Partnership	28
API	Application Programming Interface	120
APM	Actions per Minute	72
BCI	Brain Computer Interface	106
BT	Behavior Tree	119
CBFBT	Control Barrier Function Behavior Tree	47
CBF	Control Barrier Function	47
CLIPS	C Language Integrated Production System	97
CNC	Computerised Numerical Control	23
CoE	Cluster of Excellence	146
CPPS	Cyber Physical Production System	28

CRD	Cluster Research Domain	21
DL	Deep Learning	19
DoF	Degree of Freedom	106
DP	Dirichlet Process	60
DPGMM	Dirichlet Process Gaussian Mixture Model	60
DS	Digital Shadow	119
DSN	Dynamic Sequence Node	122
DSS	Decision Support System	32
DT	Digital Twin	29
eBT	Extended Behavior Tree	18
EE	End Effector	89
FAIR	Findable, Accessible, Interoperable, and Reusable	32
FEM	Finite Element Method	35
FFT	Fast Fourier Transform	80
FOL	First Order Logic	15
FRP	Fibre Reinforced Plastics	23
FSM	Finite State Machine	121

GA	Genetic Algorithm	85
GAN	Generative Adversarial Network	6
GMM	Gaussian Mixture Model	48
GUI	Graphical User Interface	31
HAN	Human Action Node	122
HRC	Human-Robot Collaboration	121
HRI	Human-Robot Interaction	120
HTML	Hyper Text Markup Language	27
HTN	Hierarchical Task Network	121
IIoT	Industrial Internet of Things	29
IM	Input Mapping	105
IoP	Internet of Production	145
IoT	Internet of Things	31
IQA	Information Quality Assessment	45
IR	Industrial Robot	2
KB	Knowledge Base	97
KBSG	Knowledge-Based Systems Group	145

LD	Linked Dataspaces	31
LfD	Learning from Demonstration	85
LSTM	Long Short Term Memory	61
LTS	Labelled Transition System	105
MARL	Multi-Agent Reinforcement Learning	5
MIP	Mixed Initiative Planning	121
ML	Machine Learning	117
MMR	Match Making Rating	72
MST	Maximally Specific Tree	121
MVP	Minimum Viable Product	120
NPC	Non-Player Character	12
OCELs	Object-Centric Event Log	36
OEM	Original Equipment Manufacturer	116
PDDL	Planning Domain Definition Language	121
PHAST BT	PHAse Switching Teleoperation Behavior Tree	83
RB	Rule Base	97
RCLL	Robo Cup Logistics League	97

RL	Reinforcement Learning	45
ROS	Robot Operating System	42
SCAN	Shared Control Action Node	123
SCT	Shared Control Template	105
SGD	Stochastic Gradient Descent	70
SME	Small and Medium-sized Enterprise	85
SMF	Salient Motion Feature	61
STRIPS	Stanford Research Institute Problem Solver	116
TAM	Technology Acceptance Model	36
TSC	Transition State Clustering	59
TSLSTM	Temporal Sliding LSTM	61
UAV	Unmanned Aerial Vehicle	105
URI	Uniform Resource Identifier	27
VA	Variable Autonomy	84
WWL	World Wide Lab	119
WWW	World Wide Web	119

Chapter 1

Introduction

Industrial automation research has been on the rise in recent years. The use of robots in industry ensures a repeatable task performance and a predictable product quality in many domains [Baier et al., 2022]. However the closer industrial robotics gets to mainstream production, the more complex such tasks get and the higher the reliability requirements are. This is particularly true in dynamic environments where the robots have to share their work-space with other robots or with human workers [Baier et al., 2022, Trinh et al., 2023b]. Task execution and monitoring in dynamic environments, especially those shared with humans is one of the open topics in robotics research targeted by many practitioners of automation and the Industrial Internet of Things (IIoT) to fulfill the vision of Industry 4.0.

One initiative that attempts to fulfill this vision is the Internet of Production (IoP), which is a Cluster of Excellence (CoE) at RWTH Aachen University in Germany. The IoP has the vision of connecting industrial partners across the globe into a World Wide Lab (WWL). It plans to create an infrastructure, including standard data structures, algorithms, and tools to support a network with high traffic and heterogeneous nodes that represent different aspects of the production processes from the machine sensors, to shop floor, to the company's policies and monetization.

Part of the IoP's vision is the inter-company communication and information exchange answered by the WWL [Behery et al., 2023d, Liebenberg, 2021, Behery et al., 2023b], which uses Digital Shadows (DSs) [Bauernhansl et al., 2018] to represent entities such as companies, machines, and products [Liebenberg, 2021]. DSs are task and purpose specific projections of the entities [Behery et al., 2023b], unlike the widely used and almost omnipresent Digital Twins (DTs), which offer a full simulation of the entity they represent. This allows DSs to target real time tasks such as quality prediction and plan repair [Liebenberg, 2021]. Moreover, this allows WWL to offer standard interfaces to dynamic and evolving DSs of the different processes and entities within.

Integrating industrial robotics processes into the WWL requires a standard DS for representing these processes and the robots involved [Behery et al., 2023a]. Many languages and robot programming paradigms attempt to handle this in different ways. The challenge lies in combining robotic control with human domain knowledge in a comprehensive, human-readable, and actionable representation. A road map for DS development was recently defined in [Bauernhansl et al., 2018], that goes over the requirements and milestones for DSs. This road map was further studied for industrial robotics applications [Behery et al., 2023a, Behery and Lakemeyer, 2023], where Behavior Trees (BTs) [Colledanchise and Ögren, 2018] were proposed as a task-level DS for assembly processes.

BTs have recently been introduced into the world of robotics as a generalization of previous approaches that exploit human knowledge in design and execution of robot behaviors. BTs have come a long way over the past decade from game Artificial Intelligence (AI) design to the world of robotics and robotic agent control. However, some work is still needed to complete the transition from the game AI domain to industrial robotics, not to mention the WWL [Behery and Lakemeyer, 2023, Behery et al., 2023a]. This is particularly true in multi-agent environments, whether other agents are humans, machines, or robots.

Cooperation and navigating with other agents increases the uncertainty in the environment as well as the overhead of coordination between the agents. The uncertainty, complexity, and communication overhead increase with the number and type of agents present in the environment. Moreover, cooperation with a human partner requires a high safety standard, fault tolerance, and failure handling [Trinh et al., 2023b, ISO/TS 15066:2016, 2016]. This is important in shared work spaces and becomes even more critical in overlapping work-piece manipulation tasks, where the robot and the human have to work on the work piece at the same time [Dammers et al., 2022].

This thesis is an anthology of some of the work done focusing on the integration of Industrial Robots (IRs) in the WWL. The next section highlights the contributions of the thesis. Most of these go in the direction of enabling the integration into the WWL by creating a respective DSs of the given process.

1.1 Thesis Contributions

This thesis begins with an overview of the vision, benefits, and challenges of the WWL and a road map towards its acceptance as a global network of industrial partners. It also discusses the integration of some industrial robotics processes and their adaptation into the WWL. Additionally, this thesis offers extensions to traditional BTs that allow them to handle more use cases that were not necessarily present when BTs were first created for game AI domains and game environments. As BTs made the transition into robotics, the number of BT applications increased and so did the requirements. BTs have so far shown great success in many robotics domains [Iovino et al., 2020, Colledanchise and Ögren, 2018], but industrial scenarios, where the robots have to work alongside humans have more challenges. In this thesis, we bridge the gap between the current state of BT research and development, and industrial robotics use cases within the IoP to fulfill some of the visions and goals of Industry 4.0. The contributions of this thesis answer three main research questions:

1. What is the WWL?
    - What are the benefits and major challenges of such a network?
    - How can we achieve this vision?
    - How can we represent the different robotics tasks in the WWL?
2. How can the WWL integrate processes, which cannot be fully automated?
    - How can we extract models for the human behaviors in these processes?
    - How can we represent these behaviors in the WWL?
    - How to use these models to assist the manual process execution?
3. How can we bridge the gap between BT research and development and industrial use cases?
    - How can we increase the flexibility of BTs work in production environments?
    - How can human workers collaborate with BT-driven agents while maintaining the trust between the human and the robot?
    - Can we use BTs to control a teleoperated robot?

The thesis answers these questions by contributing in three research directions.

1.1.1 The World Wide Lab

We first review the vision of the WWL, and the challenges to achieve this vision. The WWL was originally introduced as part of the vision of the IoP [Brauner et al., 2022, Liebenberg, 2021]. This thesis gives it a road map that tackles both the long- and short-term challenges. This work not only discusses the technical perspective, but also the economical, ethical, legal, and social ones. After that, we visit the state of the art in BT research in terms of the developmental milestones for using DSs in production domains in the context of Industry 4.0. In our work, we also review the requirements for DSs defined in [Bauernhansl et al., 2018] and their integration into the WWL as well as the state of BT research. Additionally, we offer a road map to further develop BTs into a standard task-level representation DS for industrial robotic assembly.

Moreover, we show how certain robot safety functions can be encapsulated in BT nodes stored and shared across company boundaries through the WWL. We proposed a framework where these safety functions can be moved and shared between BTs that represent different processes and performed by different robots.

1.1.2 Industrial Robot Teleoperation

We discuss a framework for teleoperative assistance. Our framework starts with teleoperation data collection and action discretization. It learns the common operator behaviors as well as their Planning Domain Definition Language (PDDL) models and allows the users to compose more complex task descriptions as BTs. A user can then use these learned BTs for conformity checking and runtime assistance during teleoperation. We visit an industrial teleoperation use case and go over the details of the framework's modules and how we evaluate each of them.

1.1.3 BT Extensions for Highly Dynamic Environments

We show how BTs can make use of expert knowledge in addition to real time optimization for assembly tasks. This increases the robustness of BTs while keeping a low workload on the human designer. To this end, we propose three new BT extensions. The Dynamic Sequence Nodes (DSNs), which allow the agent to reorder the tree at run time according state of the environment. The second extension allows BTs to model a collaborative task using $\mathcal{H}$-Nodes. And the third extension, PHAse Switching Teleoperation Behavior Trees (PHAST BTs), allows BTs to represent robot teleoperation tasks.

1.2 Relevant Publications

Major parts of this thesis have been published in Journals, Conferences, Symposia, and Workshops over the past few years. This section gives a brief overview of these publications and their relation to this thesis. To give order to the publications, we examine the production landscape in a top down approach. This thesis presents work which can be distributed on three levels, the *machine level*, the *shop floor*, and the *global scale*. [Behery et al., 2023b] covers the three levels and surveys some of the infrastructural work done in the CoE till the time of its writing.

1.2.1 Machine and Robot Level

Since most of the use cases described in the thesis involve robotics, we denote by the machine level any contribution that either optimises a robot's decision process, generates its trajectory, or processes the data it generates. On this level, we published some work related to the metal forming use case, where a robot arm is typically teleoperated using a joystick to transport a metal ingot. [Behery et al., 2020] presents an approach where we discretize the continuous teleoperation data into discrete actions as a step towards learning the operational behavior in order to detect errors and deviations from that behavior at a later stage. A subsequently supervised thesis shows that, given sequences of state-action pairs, we are able to learn a library of composable common behaviors as well as PDDL action models for these behaviors [Krömker, 2020]. Additionally, towards robot teleoperation, [Behery and Lakemeyer, 2022] describes a method for intention recognition in robot teleoperation tasks. This is needed in order to provide the operator with assistance in shared autonomy scenarios. Moreover, [Behery et al., 2023e] provides an activity representation aimed at teleoperation activities, where the user defines the activity as a specific class of BTs. In this BT class, the action nodes define how the system assists the user in teleoperating the robot.

The work done on the machine level also includes applying Mixed Initiative Planning (MIP) to BTs [Behery et al., 2023f]. In this work we discuss a new type of control flow nodes for BTs, that can reorder its own children, modifying the tree at run time in order to make local optimizations. This allows the supervisor to abstractly define a robot's activity, and the robot to ground and modify this at run time according to the dynamic world state.

Some work also went into the automation of labour intensive tasks, such as composite material manufacturing. [Dammers et al., 2023] studies the application of scene prediction using depth cameras as an attempt to learn how a human operator performs one of the sub-tasks. A subsequent thesis applies imitation learning in order to teach a robot how to perform this task from an expert worker's demonstrations [Wolf, 2023]. On another front, we also examine performing machining operations using a robot arm instead of traditional machining tools to exploit the larger workspace offered by robots. In [Trinh et al., 2022a], we model the dynamics of the robot during such an operation using transformer networks. Additionally, [Daabis, 2024] studies the application of Multi-Agent Reinforcement Learning (MARL) for pick and place tasks. We look into the combination of pick, place, and trajectory generation tasks in a single policy, which results in comparable performance to the state of the art methods studying Reinforcement Learning (RL)-based trajectory planning.

1.2.2 Process and Shop Floor Level

On the shop floor level, we group the work, where different robots or machines interact with each other or with human coworkers and study the interaction between different products and machines in an assembly line. [Baier et al., 2022] surveys several Human-Robot Collaboration (HRC) use cases within the CoE and presents a new taxonomy and classification for the tasks involved. One of the main concerns in HRC is the safety of the human partners. [Trinh et al., 2023a] presents a CoE-centric survey of safety in HRC scenarios across several use cases. Further [Trinh et al., 2023c] shows how we combine a BT with off the shelf computer vision techniques for human skeleton detection in order to conform to safety standards in an assembly scenario. Finally, [Behery et al., 2023c] shows how we combine BTs with rule-based systems to dynamically dispatch trees to a robot, which can consequently collaborate with a human partner to work on multiple products simultaneously. In order to predict the performance of a process, we apply process mining techniques to Object-Centric Event Logs (OCELs) [Rohrer, 2022]. This is done by using an auto-encoder to learn representations of the events in the log then using Generative Adversarial Network (GAN) to predict the next event with its related attributes. Another approach to process optimization was demonstrated in [Gannouni et al., 2020], where we apply reinforcement learning for the scheduling of a plastic extrusion process. We first learn a reward associated with the different item pairs, then use this reward to learn a policy for scheduling incoming item orders, which reduces waste and increases throughput.

1.2.3 Global Scale and the WWL

At the top level, the global scale, we discuss a vision for the WWL presented in [Behery et al., 2023d]. Ideally, this integrates all production lines and plants into a single network, an IoP, analogous to the relation between the World Wide Web (WWW) and the *Internet*. With a focus on robot processes, [Behery et al., 2023a] argues for using BTs as a standard DS for representing robotic tasks in the WWL. We further present a more specific use case in [Behery and Lakemeyer, 2023] of using BTs with Control Barrier Functions (CBFs) to ensure safety and represent safety requirements in the WWL. This also encourages code-reuse in such scenarios, because CBFs are commonly used to ensure safety in different control tasks and can be reused, encapsulated into BT nodes, and directly integrated into BTs which can then be stored, queried, and exchanged in the WWL. This work places focus on using these nodes as artefacts for exchanging information and safety requirements between roboticists.

1.3 Thesis Outline

This thesis is structured as follows:

- **Chapter 2:**

  Starts with an introduction to DSs used in the WWL. After that, it gives an overview of Robot Operating System (ROS), a robot control framework, then goes over the fundamentals of BTs and their background. After that, it surveys the current state of BT research, particularly BT extensions and synthesis approaches.

- **Chapter 3:**

  Surveys some of the robotics use cases within the IoP. It gives a brief overview of the work done on these use cases and how the CoE contributes to them. Some of these use cases are revisited in later chapters to discuss the contribution of this thesis to them.

- **Chapter 4:**

  Describes the vision and goals of the WWL. The chapter reviews the benefits and challenges of the WWL and provides a road map to tackle the long and short term challenges. We then motivate the the use of BTs as standard DS representation for industrial robotics applications in the WWL. We also give another road map for pushing BT development further to allow its integration into the WWL. This chapter discusses work that was published in [Behery et al., 2023a, Behery et al., 2023b, Behery et al., 2023d, Behery and Lakemeyer, 2023]

- **Chapter 5:**

  Goes into the details of teleoperation tasks in industrial scenarios. After that we discuss our proposed framework for learning a robot teleoperator's behavior, descretizing the teleoperator's input signals into actions to extract composable patterns allowing us to model new behavior and compare run time commands to the learned behavior for conformance checking and error correction. Some work in this chapter was previously published in [Behery et al., 2020] and some of it was introduced in [Krömker, 2020].

- **Chapter 6:**

  Discusses dynamic industrial scenarios and how we extend BTs allowing agents to interact and collaborate with humans in HRC scenarios from the IoP describing both human *-in-* and *-on-* the loop situations. This chapter discusses three BT extensions: the first extension applying MIP to BTs, the second one allowing BTs to handle HRC tasks, and the third extension allowing users to define and use BTs for robot teleoperation tasks. We discuss the new node types introduced with each extension and how the new types are integrated in BTs. The work in this chapter was published in [Behery et al., 2021, Behery et al., 2023c, Behery et al., 2023e, Behery et al., 2023f]. It also presents some novel work that was not previously published.

- **Chapter 7:**

  Concludes the thesis with a summary and a discussion of the results, strengths, and weaknesses of the presented work in addition to recommendations for future work.

Chapter 2

Background and Related Work

Robot task-level control frameworks have grown in popularity over the past decades. This is particularly useful in Small and Medium-sized Enterprises (SMEs) where the robot operators often lacks programming or robotics experience [Berger and Armstrong, 2022, Perzylo et al., 2019]. Use cases that fit this description are present in both industrial and domestic robotics. Behavior Trees (BTs) have shown great success in many of these use cases since their adaptation by the robotics community [Iovino et al., 2020].

In Chapter 4, we present the World Wide Lab (WWL) with its vision and challenges as well as a road map to achieve its goals. We also present Digital Shadows (DSs) to represent some processes in the WWL and road map to fully integrate. Subsequent chapters demonstrate how BTs are extended to cover more industrial robotics use cases. In order to cover the preliminaries, this chapter gives a brief introduction of DSs and their use in the WWL, then it gives an overview of robot middle ware with a focus on Robot Operating System (ROS). After that, it details how BTs are used as a robot control framework as well as their extensions and synthesis approaches as well as how they integrate with ROS.

2.1 Digital Shadows

Several industries and manufacturing processes heavily rely on Digital Twins (DTs) [Qi and Tao, 2018] to represent and simulate their entities and make predictions about their results. Some processes also rely on them for scheduling their steps. These DTs often contain a lot of (or all) details that model the process' entities, the interactions, and the relations between them throughout the process. They are considered a one-to-one digital representation of the physical process or entity [Bergs et al., 2021]. However, that amount of information and knowledge is sometimes unnecessary and leads to an unwarranted computational complexity [Liebenberg, 2021]. This complexity limits, and sometimes precludes, their use in real-time tasks, such as safety critical situations.

Taking that into consideration, the WWL relies on using DSs instead of the omnipresent DTs. DSs are task- and goal-specific digital representations of their physical counterparts. They are considered projections of different aspects of the process or entity they represent. This way, they require less memory and less computational power to yield their results.

For example, a Finite Element Method (FEM) simulation [Zienkiewicz et al., 2005] divides the entity into a number of finite discrete elements, then does the computation, based on a physical model, to find how it affects the neighbouring elements. A DT for the hot rolling process based on a FEM simulation, divides the metal work piece into such elements, and calculates the heat transfer between neighbouring elements. The running time for this simulation can be somewhere between several minutes and days making it an inappropriate solution for real time tasks, such as plan repair. [Seuren et al., 2012, Bambach and Seuren, 2015] showed that such simulations could be replaced by a DS relying on a fast mathematical model, which significantly reduces the simulation time from 30 to 240 minutes to 50 milliseconds. Another DS, which uses a neural network trained on the fast mathematical model, to generate the production schedule for the process replacing a traditionally combinatorial optimization scheduling problem [Meyes et al., 2018].

The thesis discusses the use of DSs for representing robotics tasks in the WWL. The next sections give a brief overview of robot control frameworks, and how they integrate task representations. Chapter 3 surveys some of the robotics use cases in the Internet of Production (IoP) and how the IoP contributes to these use cases. After that, the thesis goes into the details of the the applicability of the different DSs and integration of these DSs into the WWL.

2.2 Robot Control Frameworks

Several robot control frameworks exist to connect the different components on the different layers of control. In our work, we focus on ROS[1] [Quigley et al., 2009] since it is the de-facto standard [Anandan, 2019] in both academia and industry. Like most robot control frameworks, a robot's components are divided into layers representing levels of abstraction, as seen in Figure 2.1. At the Lowest level, we find components closest to the hardware level, such as motor drivers and robot kinematics and dynamics, which are usually tightly bound to the robot. As we move to higher levels, components become more abstract and platform-agnostic.

Different components are able to communicate in ROS using topics and services. As seen in Figure 2.2, ROS represents components as nodes

[1] ros.org

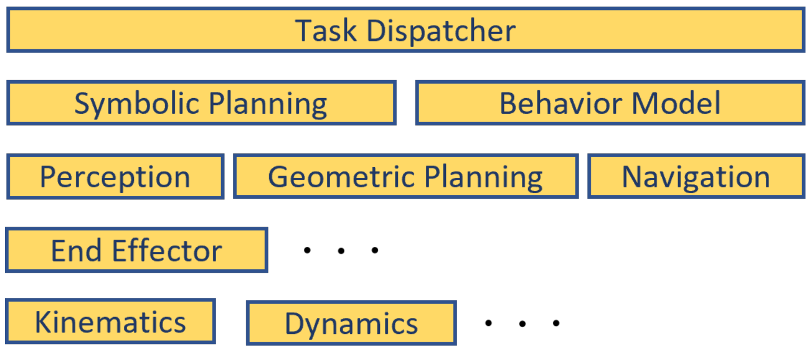


Figure 2.1: An overview of ros robot middle-ware. Different components of the robot software are connected through several channels, topics and services.

in a graph, each of which is able to subscribe to topics and declare its own services and topics. Topics allow a node to broadcast messages while services allow direct peer to peer communications between the nodes. ROS also defines a master node called `rosmaster` that keeps track of the nodes and declared topics and services. To declare a new topic or service, a ROS node defines a new message type that contains all necessary data.

ROS Case Study

For example, we consider a mobile robot with a depth camera and a manipulator whose task is to pick up an object. The depth camera declares a topic for its images, `depth-img`. It would use a message type with the four channels for RGBD, each of which would be represented by a 2D array. At a given frequency, the camera posts new images to a camera topic. It might also define a topic for 2D images where it publishes the RGB channels and another topic where it publishes the depth information. A higher-level perception module would subscribe to one of those topics and process the images to detect the object of interest, publishing its location w.r.t. a world frame in a fourth topic, `object-location`. A robot navigation module can subscribe to the `object-location` topic and ask the lower level kinematics for a configuration that allows gripping the object by calling the provided service. After that it calculates a trajectory to reach that configuration and communicates it to the lower levels components that drive the motors.

Thanks to ROS's modularity, the above system can be complemented with a symbolic planner that chooses the objects of interest according to a plan that achieves a given goal. We can also further enhance this system by using a task dispatcher that decides which tasks are allocated to the robot and which can be aborted. To achieve this, we would need to either exchange the object detection module with a more general detector that detects more objects, or use a more flexible architecture that has several

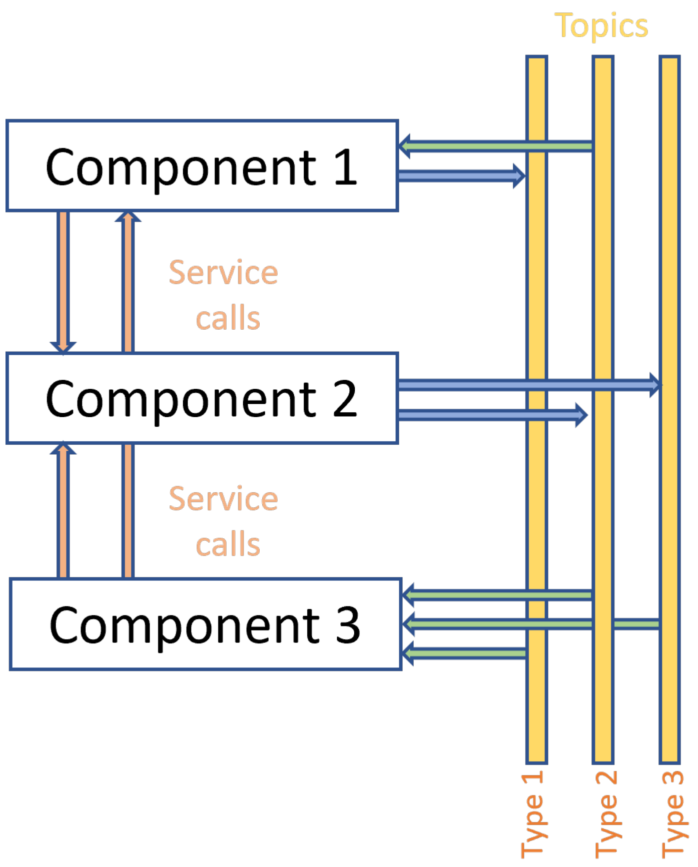


Figure 2.2: An overview of ros robot middle-ware. Different components of the robot software are connected through several channels, topics and services.

detectors and the planner can then query the right one for the right object.

In our work, we use a ROS node that encapsulates the behavior of the robot using a BT. The remainder of this chapter discusses our work on using BTs as a standard DS for representing task level behavior control of industrial assembly robots. We not only exploit the desirable properties of BTs for industrial use cases, we also use them in the WWL as medium for information exchange across the boundaries of an organization and even between practitioners in academia and industry.

2.3 Behavior Trees

BTs are behavior models that originally started as a game Artificial Intelligence (AI) tool used to design Non-Player Character (NPC) behavior in well known games such as Halo 2 [Isla, 2005]. They generalize other frameworks such as Hierarchical Task Networks (HTNs), Finite State Machine (FSM), and teleoreactive programs, among others. BTs were later adapted by the robotics community for these generalizations in addition to other favorable properties that will be covered in this chapter.

A BT is a task switching model that structures the robot's behavior in a tree structure. Leaf nodes represent the atomic actions while the internal nodes control the ordering of these actions. Traditional BTs offer two types of leaf nodes, *action* and *condition* nodes. Action nodes are responsible for actuation, such as navigation or arm motion while condition nodes are responsible for checking whether a specific predicate holds, e.g., *lightOn*. Internal nodes are control flow nodes and are classified into the following four types:

- Sequence: Used for sequential execution of their children, e.g., *openDoor* then *moveInside*.
- Fallback: Their children define alternative execution paths, e.g., *openDoor* or *knock*.
- Parallel: Children are executed in parallel, e.g., *navigate* and *dockArm*.
- Decorator: Used to define custom behavior, e.g., *repeatUntil*.

Extensions to BTs, discussed in Section 2.3.2, introduce more node types to represent different behaviors.

The execution of a BT starts by signaling the root node using a *tick* which is then propagated through the tree towards the leaves. When a node is activated using a tick, they are executed (in case of action and condition nodes) or tick their children according to their types (in case of internal nodes). After that the node returns a status $s \in \{\mathcal{S}, \mathcal{R}, \mathcal{F}\}$ indicating Success, Running, or Failure, respectively, to its parent. The returned status is determined according to the node type as seen in Table 2.1.

Node	Behavior
Action	**execute an action** and return $\mathcal{S}$ or $\mathcal{F}$ depending on the action's success, or $\mathcal{R}$ while it executes the action
Condition	**check a condition** and return $\mathcal{S}$ or $\mathcal{F}$ according to its status and never return $\mathcal{R}$
Sequence	**execute the children in order** from left to right. They return $\mathcal{S}$ iff *all* the children succeed, $\mathcal{F}$ iff at least one fails, and $\mathcal{R}$ otherwise
Selector	implements a **fallback behavior** by executing the children in order from left to right and return $\mathcal{S}$ iff *one* child succeeds, $\mathcal{F}$ iff all the children fail, and $\mathcal{R}$ otherwise
Parallel	**execute the children in parallel**. A node with N children and m threshold returns $\mathcal{S}$ iff m children succeed and $\mathcal{F}$ iff $N - m + 1$ children fail, and $\mathcal{R}$ otherwise
Decorator	used to **implement custom** behavior (e.g., repeat n times) and manipulate the returned status of their child (e.g., negation)

Table 2.1: A summary of node types and their return status in traditional BTs as defined in [Colledanchise and Ögren, 2018]

Definition 2.3.1 (Behavior Tree)**.** A BT is a graph has a tree structure, with nodes that belong to one of the classes: Sequence $\mathbb{S}$, Fallback $\mathbb{F}$, Parallel $\mathbb{P}$, Decorator $\mathbb{D}$, Condition $\mathbb{C}$, or Action $\mathbb{A}$.

A BT $\mathcal{T}$ is defined as a tuple $\langle r, \mathbb{D}, \mathbb{C}, \mathbb{A} \rangle$, such that $r \in \mathbb{S} \cup \mathbb{F} \cup \mathbb{P} \cup \mathbb{C} \cup \mathbb{A}$ is the root of the tree and $\mathbb{D}, \mathbb{C}, \mathbb{A}$ are the sets of decorator, condition, and action nodes, respectively.

Any node $n \in \mathbb{S} \cup \mathbb{F}$ has a non-empty ordered list of children $children(n)$, which defines the order of execution of the children using a successor relation $\succ_n$. A parallel node p has an unordered non-empty list of children $children(p)$ since it executes the children in parallel. It has a parameter m, which defines the threshold of successfully terminating children for p to succeed. A decorator node d has only one child which can belong to any node class, $|children(d)| = 1$. Action and condition nodes are always placed as leaf nodes in the tree and may have an ordered list of parameters that denote objects in the environment. They can also be used as the root of a BT, but in that case, the tree would have no other nodes.

We use a function $tick(n) : n \to \{\mathcal{S}, \mathcal{R}, \mathcal{F}\}$ to represent ticking a node and the returned status to describe whether the node succeeded, is still running, or failed, respectively. We assume that ticking a -non-trivial- BT $\mathcal{T}$ updates the world state by executing the children. To describe this, we use the notation $s' \doteq \mathcal{T}(s)$, where s' is the new world state after executing $\mathcal{T}$ in state s. The definition of the world state is domain specific, we assume it is a vector containing the relevant predicate and function symbols, which an agent could use to describe the world. For example, it may contain a function symbol $pose : \varnothing \rightarrow \mathbb{R}^6$ that returns a 6D pose of the robot[2].

To describe the conditions encapsulated by condition nodes, we use First Order Logic (FOL). In our work we assume that the robot has an underlying finite Knowledge Base (KB) containing information about the environment, which is update in real time using the robot's perception module(s). Depending on the architecture of the agent, this KB might be a database, a dictionary (commonly named a blackboard), or any other form of data storage. The assumption of an underlying KB containing information about a finite number of objects in the environment in addition to the closed world assumption [Raymond, 1978] allows the robot to only keep track of the relevant information and limit the reasoning to this information and what can be deduced from it. i.e., objects unrelated to the current task are not included in the KB and are not included in the variable assignment when attempting to evaluate a query and are thus not considered when evaluating the FOL expressions. For example, if a domestic robot's task is to prepare coffee in the kitchen, it is not concerned with objects that don't belong to the kitchen environment, such as the bed, and therefore doesn't use it when matching predicates such as $clear(x)$ when trying to find room to place a coffee mug.

Using FOL for the condition nodes allows the use to exploit its expressive power by using quantifiers as well as function and relation symbols involving multiple terms. Moreover, under the above assumptions, especially the underlying finite KB, SAT becomes decidable allowing us to test -at design time- whether tree branches could be visited at run time, depending on whether the condition of their respective guarding branch [3] is satisfiable. This offers assistance during manual tree design by warning the designer of branches that will not be visited. It also helps during tree synthesis by pruning the branches with unsatisfiable guards.

2.3.1 Example Behavior Tree

This section presents a common use case for industrial robotics, where a closed loop robot performs pick and place tasks to move products from one conveyor belt to another. As seen in Figure 2.3, the blue box to the right drops cubes of red, green, and blue colors on black conveyor belt. When they reach the robot, it moves them to the white conveyor belt. The setup has a detection window marked in red, and two tracking windows marked as green and yellow. Cubes in the green areas are the ones reachable by the robot on both conveyor belts.

[2] Location and orientation in the Cartesian space.

[3] The guard g of a branch b acts as a precondition to ticking b, its the subsequent sibling, such that b is ticked iff $tick(g) = \mathcal{S}$

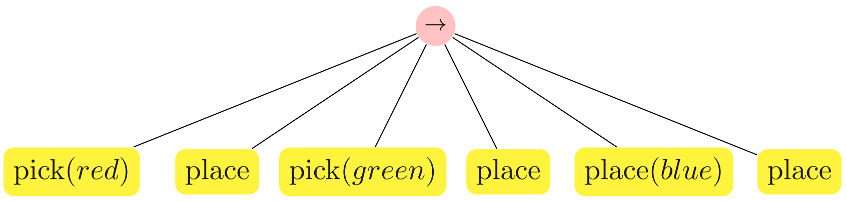


Figure 2.4: A simple BT for an industrial pick and place task.

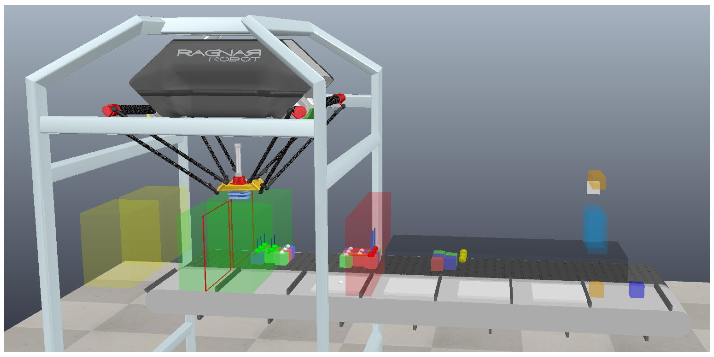


Figure 2.3: A robot with a closed kinematic chain picks products from one conveyor belt to anther. The products are dropped on the first belt at random and should be placed in a 9×9 grid.

In such a scenario, the level of autonomy can vary. One solution is to hard code the locations of the source and target positions for the cubes. This method is common in some applications, such as automotive, which makes up to 80% of robot installations [Hägele et al., 2016]. It does not require sensors or feedback loops between the environment and the robot. Such open-loop control forms about 35% of industrial robotics use cases. However, this is error prone and does not react to the errors or inaccuracies that can happen changing the speed of the belts or the relative locations of the cubes on the belt. Additionally, changing the sizes or shapes of the cube requires reprogramming and re-calibrating the system, which can be time consuming and error prone.

We can construct a simple BT, as seen in Figure 2.4, that picks the cubes in the order, red, green, and blue, and places them in their respective target locations on the second belt. The tree uses two atomic actions, pick and place as follows:

- pick(x): picks a cube with color x from the black belt
- place: places the currently held cube at its target position

As long as the tree is ticked, the robot will keep performing these tasks. The tree will fail as soon as one of these actions fails, which could mean that a cube of a certain color was not located.

For a more responsive tree, we can use a composition of fallback, sequence, and decorator nodes as seen in Figure 2.5. Here we introduce the condition node:

- has(x): which returns $\mathcal{S}$ if the color x is detected on the black belt and $\mathcal{F}$ otherwise.

In this composition, if a color is not detected, the tree ticks the next branch to handle the next color, because the decorator node $\neg$ inverts the output of the condition node, returning $\mathcal{S}$. This causes the parent node to return $\mathcal{S}$ to the root node, which ticks the next branch.

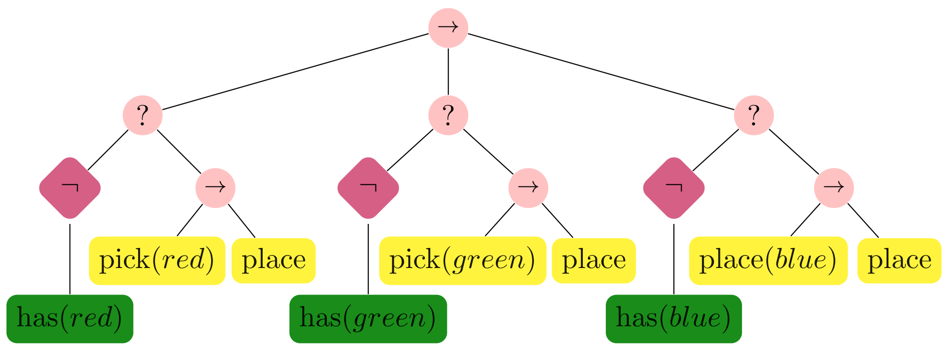


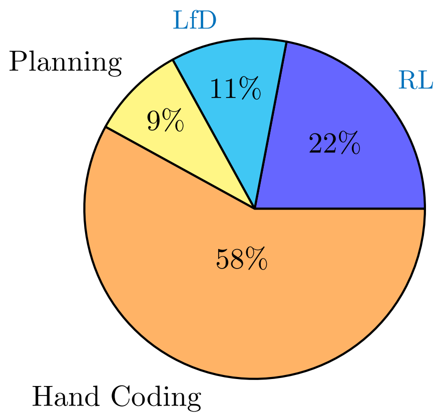
Figure 2.5: A BT that reacts to missing cubes in the pick and place tasks.

This use case can be studied from different perspectives such as task planning, path planning, perception, and safety.

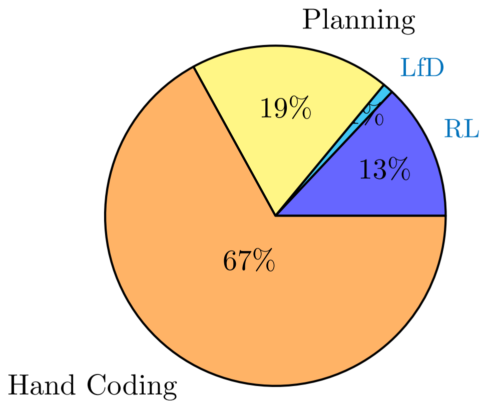
2.3.2 Behavior Tree Extensions and Generation

Over the last decade, as BTs made the transition from the game industry into academia and from game AI into robotics, they gained a lot of traction as their research split into several directions and application domains [Iovino et al., 2020, Colledanchise and Ögren, 2018]. Countless extensions have been proposed to cover more use cases and many synthesis approaches have been developed focusing on different aspects. Additionally, some validation techniques have been introduced to apply BTs in safety-critical applications and tasks. This section gives a brief overview on the state of the art in BT research and development, a more detailed systematic survey is given in [Iovino et al., 2020].

As seen in Figures 2.4 and 2.5, when we add more requirements to the system, the complexity of the BT can increase drastically. This complexity increase is still better than the alternatives, HTN and FSM among others, because increasing the number of nodes in a tree requires less state transitions to handle the change compared to increasing the number nodes in an FSM. However, it can still become difficult to design, modify, and introspect the trees as they scale up to handle more complex tasks, especially by a human worker or supervisor. This called for a number of extensions that



(a) Distribution of BT creation methods based on data from [Iovino et al., 2020].



(b) Distribution of BT creation methods for robotics applications based on data from [Iovino et al., 2020].

Figure 2.6: Approaches for creating BTs, showing the overall distribution in Figure 2.6a and in robotics applications as seen in Figure 2.6b.

allow BTs to cover more use cases that were not covered by traditional BTs, or give them more flexibility and robustness to cover more corner cases that compromised safety or task performance without affecting the readability or modularity of the trees.

[Csiszar et al., 2017] introduced new control flow nodes, parallel sequencer and parallel selector, which tick all their children in parallel. The former only returns $\mathcal{S}$ iff all the children have successfully terminated and the latter returns $\mathcal{S}$ if at least one child was successful. Additionally, they introduce event nodes which respond to exogenous events, which can interrupt execution and ticking new BTs. They also focus on the usability by allowing the programmer to add labels to control flow nodes, which facilitate navigation, debugging, and troubleshooting. [Rovida et al., 2017] introduced Extended Behavior Trees (eBTs) which use conditions to allow optimization of the BT leading to reductions in execution time. Motion Generators were also encapsulated in BT leaves in [Rovida et al., 2018] to reduce programming complexity and exploit the modularity of BT to define robot skills. Furthermore, BTs were used to implement multi-robot systems in [Colledanchise and Natale, 2018, Colledanchise et al., 2016]. More extensions were introduced to BTs to allow new behaviors, such as pause and resume mechanisms [El-Ariss et al., 2021] inspired by HTNs.

The above approaches, among others, focus extending the functionality of BTs, while a developmental bottle neck remains in the programming and creation of these trees. In many cases, the programming complexity of the trees increases with the number of available node types. Several recent approaches learn BTs use Reinforcement Learning (RL), or Learning from Demonstration (LfD), while others rely on more traditional planning approaches such as backward chaining [Colledanchise et al., 2019] or even a mixture of learning and planning [Iovino et al., 2021]. While some approaches learn or synthesize a BT according to some specification or demonstrations, others are given a BT and are able to rebuild it. Figure 2.6a shows the distribution of tree creation approaches, both in robotics use cases (Figure 2.6b) as well as in BT application in other domains (Figure 2.6a). We can see that in both cases, hard coding is the most prominently used approach, with a higher market share in the case of robotics, which can be attributed to the higher safety and reliability requirements in those use cases. This calls for:

1. Reducing the mental effort required to design a BT
2. Enhancing tree validation and verification
3. Eliminating the obstacles hindering tree synthesis and learning approaches

Some approaches modify a BT using a Q-value, for some branches in the tree and reorder the tree accordingly. The Q-value is commonly used in Q-learning [Watkins, 1989] algorithms as a utility value for each state. [Dey and Child, 2013] calculates Q-values for the lowest level sequence nodes and uses them as actions in RL. [French et al., 2019] allows fallback nodes to do the learning and executes the children with the highest Q-values first. And [Zhang et al., 2017] calculates the Q-values for children of fallback nodes and adds condition nodes to only execute the children when they are the best options. Another application of RL in BT design and synthesis is learning parameters of a given BT [Mayr et al., 2021] turning the BT into a generalisable policy, able to transfer from simulation to a physical robot. Another learning approach is using Genetic Algorithms (GAs) to learn a BT [Colledanchise et al., 2018]. The tree is later pruned to remove the unused branches.

Such approaches require training time, and access to training data, a simulation, or a fitness function to assess the fitness of the trees in a given generation, all of which are not always available or not feasible in industrial applications. Additionally, in most cases, Deep Learning (DL) and RL policies are unexplainable black-box approaches with little to no guarantees of correctness or safety which are often avoided in industrial domains.

These downsides motivated research in the direction of planning-based BT generation. The most commonly used of these approaches is PlanSys2 [Martín et al., 2021], which generates symbolic plans for a given Planning Domain Definition Language (PDDL) domain and transforms them into BTs. This maintains the guarantees of symbolic plans while exploiting the usability, in debugging, introspection, and monitoring offered by BTs. Other approaches, such as [Colledanchise et al., 2019] *grow* a tree starting from a goal condition. To grow this tree, they list the actions that can satisfy this condition then start using the preconditions of these actions as a new goal condition using backward chaining till an executable tree is generated.

However, planning-based approaches require PDDL domain models that show all the parameters, preconditions, and effects of all the actions. This is not always available in industrial scenarios, especially in SMEs, which often lack the robotics and programming expertise required for such modeling. The next chapters show our contributions in industrial robotics use cases in smart manufacturing leading to the IoP.

Chapter 3

Industrial Robotics and Where to Find Them

Different industrial revolutions allowed manufacturing to take huge leaps forward using the newly introduced technologies. The first industrial revolution introduced the steam engine to move the industry from manual labor to machine-based manufacturing. This was followed by the introduction of electricity and then computerized and automated manufacturing through embedded systems [Hopmann et al., 2023]. The fourth industrial revolution, "Industry 4.0", is still ongoing aiming to move the industrial landscape towards smart robust manufacturing. Several initiatives around the globe present multiple approaches to achieve this goal [World Economic Forum and McKinsey & Company, 2021, Industrial Value Chain Initiative, 2019, Brauner et al., 2022]. This involves increased data collection through sensors, increased communication between machines involved in a process, and increased communication between the machines and the decision makers on different levels.

The Internet of Production (IoP) is one of these approaches, which originates from the Cluster of Excellence: the Internet of Production at RWTH Aachen University in Germany. It aims at changing the global industrial landscape into a connected "*lab-of-labs*" called the World Wide Lab (WWL). The WWL, is a network of production entities, such as, blue- and white-collar workers, sensors, machines, industrial processes, manufacturing plants, and companies. Each of these entities is represented by one or more Digital Shadows (DSs). This section gives an overview of the robotics use cases and demonstrators present in the IoP. Some of these will be revisited in later chapters to discuss the thesis contributions in more detail.

The IoP structure is split into four Cluster Research Domains (CRDs), each focusing on a different production aspect, with several demonstrators. Some of the demonstrators handle robots as agents in the process, meaning they allow the robot(s) to reason and make decisions that affect the production process and interact with the environment containing other robots and human workers. An example for this is the assembly processes in the factory of the future. Others treat the robot as a substitute for machines to exploit their flexibility and cost reduction, they focus on increasing the robot's accuracy and repeatability, such as performing machining using a cobot.

This chapter reviews some robotics use cases within the IoP. Some of the concepts and use cases mentioned in this chapter were discussed in [Behery et al., 2020, Baier et al., 2022, Trinh et al., 2022a, Behery et al., 2023b, Behery et al., 2023d, Trinh et al., 2023a].

3.1 Future Assembly

This use case focuses on variable and flexible assembly production lines. These production lines are essential to face the shifting supply and demand that result from market fluctuations, supply chain disruptions, or natural disasters, e.g., pandemics [Hiscott et al., 2020, Kluge-Wilkes et al., 2023]. Having a flexible assembly line allows manufacturers to switch to new products or product variants depending on customized specifications with little costs adapting to the new paradigm of manufacturing, batch size one [Hu, 2013]. In this paradigm, assembly processes are modular and the shop floor resources are mobilized to allow full flexibility of production.

To build such an assembly scheme, robotic agents controlling the resources need the ability to make real-time decisions autonomously after coordination. This covers several levels of control and autonomy, from low level motion control, to navigation and collision avoidance, to the task-level control for individual agents, which is managed by the process control and planning [Kluge-Wilkes et al., 2023]. This thesis does not study the flexible production lines holistically. It focuses on some aspects such as the multi-product and human robot collaborative assembly, where robots work in highly dynamic environments collaborating with other robots or human coworkers. We showcase our work in this area in Chapter 6.

3.2 Metal Forming

This use case is split into several industrial processes, metal casting, open- and closed- die forging, as well as hot- and cold-rolling. Casting is a process where molten metal is poured into a mold to take its shape. Forging and rolling are used to change the geometry and structural properties of a metal ingot. The work piece is heated to a certain temperature, then moved to the forge or the rolling machine. We focus on open-die forging, where a robot arm is used to move the work-piece between the furnace and the forge. Then the process proceeds according to a pass schedule, where the arm moves the work piece into and out of the forge. In each pass, the forge presses the die onto the ingot to gradually change its shape. If the temperature of the work piece drops below a specified threshold, it should be re-heated in the furnace. The pass schedule specifies the number of these passes as well as the bite length, temperatures, and other attributes. Some approaches target generating and optimizing pass schedules [Recker, 2014, Wolfgarten et al., 2019], Others focus on reducing the generation time [Liebenberg and Jarke, 2020], while others try to learn the operator's behavior in order to assist them during the teleoperation of the arm. Many of the attributes in the process are only quantifiable through destructive tests. That is why the experience of the human operator plays a significant role in the design and execution of the pass schedule.

3.3 Fibre Reinforced Plastics Manufacturing

Fiber reinforced composite materials are a cornerstone of the production of parts in many industries such as aerospace, aviation, and automotive [Zhang et al., 2016]. Fibre Reinforced Plastics (FRP) are formed using fibers combined with polymer matrix material resulting in a new material with desirable properties such as high strength and stiffness to weight ratio [Prashanth et al., 2017]. The production of these composite materials relies primarily on manual labour by expert workers as it requires high level of dexterity to handle the fiber materials, especially during the hand-layup process. In this process, layers of the textile materials are placed on a mold and resin is applied between the layers till the required number of layers is reached [Seydibeyoglu et al., 2017].

This process is labour intensive and commonly used in the manufacturing of FRP, it has become a bottle neck in the production making it a target for automation approaches [Shirinzadeh et al., 2007, Shirinzadeh et al., 2000]. The IoP studies several approaches to automate the process, by using specialized robot end-effectors allowing the robot to handle the fibers [Kordi, 2009] and through the application of machine learning and scene flow estimation to learn the motion of the operators and replicate it by a cobot [Dammers et al., 2023], to name a few.

3.4 Machining and Subtractive Manufacturing

These two use cases have traditionally been performed using specialized machines that perform certain tasks, e.g., Computerised Numerical Control (CNC) machines. There is, however, a trend of introducing cobots to perform these tasks in order to reduce costs and increase the workspace size, allowing the manufacturing of larger work-pieces.

Cobots are not inherently suited for such tasks, as they are more malleable than the specialised machines, which have higher rigidity. They are not as accurate as they cannot stand the high forces resulting from these processes, e.g., machining [Schneider et al., 2024]. Work on the introduction of robotics to this task, includes modeling of the robot's dynamics. This involves modeling the relations between the different physical forces, that act on the different joints [Trinh et al., 2023d, Trinh et al., 2022b, Trinh et al., 2024]. This work includes the application of the recently introduced transformer models as well as the physical models to reduce errors and increase interpretability.

Some work also investigates the use of mobile robots for processes such as laser material processing [Kaster et al., 2023]. They examine the applicability of using mobile robots equipped with laser-based tools that can perform tasks such as cutting. Work is also being done on increasing the accuracy of pose estimation and trajectory following capabilities of the robots to be comparable to traditional machines [Walderich et al., 2024].

3.5 Welding and Additive Manufacturing

[Kerber et al., 2018] studies robotic fabrication in additive manufacturing to allow bigger workspaces, facilitating the production of more complex structures.

Robotic welding has also become a common use case for industrial robotics [Liu et al., 2023]. Work towards reliable robotic welding that can compete with the skilled welders has become imperative to combat the lack of skilled labour facing the market. To this end, research in this area leans towards increasing versatility and adaptability of autonomous welding systems [Biber et al., 2024]. Other work also aims at path planning for arc welding, which takes the robot's kinematics into consideration [Schmitz et al., 2021].

However, autonomous welding in practice remains rather primitive, requiring constant oversight, especially in Small and Medium-sized Enterprises (SMEs) [Universal Robotics, 2022]. Robots have to be programmed to follow a given trajectory [Silva et al., 2003]. This programming is often done using build in trajectory learning approaches created by the Original Equipment Manufacturer (OEM). These trajectories often need to be hardcoded due to the additional lack of sensors. The environment needs be highly controlled as the robots are unable to adapt to the dynamic environment due to the limited reasoning and reactivity of the robots, these methods are therefore only useful in small lot sizes [Kah et al., 2015, Mining Safety, 2017].

3.6 Others

Other use cases exist and are surveyed in [Hopmann et al., 2023] in the context of the IoP. These use cases focus not only on robotics, but also on several approaches to fulfill the vision of the WWL as a contribution towards a new industrial revolution leading to a connected manufacturing. They include communication protocols, standard data formats, optimization techniques, and explainable Artificial Intelligence (AI), among other things. The next chapter discusses the integration of these processes into the IoP and the WWL.

Chapter 4

The Road to the World Wide Lab and Integrating the Digital Shadows of Robot Tasks

This chapter gives an overview of the World Wide Lab (WWL), its benefits and vision, and gives a road map for tackling its short- and long-term challenges. After that we describe why and how we plan to use Behavior Trees (BTs) as Digital Shadows (DSs) for robot task representation in the WWL to integrate robot-based processes with a focus on assembly processes. We also give a road map to enable such standardization and unlock the full potential of using DSs. The work we discuss in this chapter was published in [Behery and Lakemeyer, 2023, Behery et al., 2023a, Behery et al., 2023b, Behery et al., 2023d, Trinh et al., 2023c].

4.1 The World Wide Lab

The networking of computers commenced in the 1950s, progressing toward the formation of the globally interconnected network, known now as the *Internet*. However, the Internet's monumental leap emerged in the 1990s, marked by the conception and development of the *World Wide Web (WWW)* as its mainstream application [Berners-Lee, 1989]. This was followed by the introduction of HyperText, Hyper Text Markup Language (HTML), and Uniform Resource Identifiers (URIs) [Berners-Lee and Cailliau, 1990], crucial tools that standardized the communications between the connected devices. Similarly, the Internet of Production (IoP) aims for interconnected industrial devices, such as sensors and machines, that often lead to the generation of industrial big data. Nevertheless, a network of this magnitude demands an expansive, resilient, and adaptable infrastructure, which not only focuses on the technical aspects of this interconnection, but also on the legal, economical, ethical, and societal ones.

The WWL aims to provide a toolbox for sharing, aggregating, and using the available data across different connected entities. It also promises a set of tools, methods, and data standards to process and analyse the data.

This work gives a review over the challenges of the WWL. We cover technical milestones leading to an infrastructure able to carry the WWL providing communication protocols and ensuring safety and security of the data. We additionally go over the challenges from humanities point of view, ethical, legal, and societal aspects as well as the economical aspects that face the feasibility and maintenance of the WWL. This thesis gives the short and long term road maps and recommendations for achieving the vision and aims of the WWL.

This Section first reviews the different approaches and initiatives towards Industry 4.0. After that, it describes the different aspects of the WWL detailing the vision and challenges of each. Analogous to the transformative influence of the WWW on the internet, the proposed WWL interconnects different industrial entities and partners within an IoP.

4.1.1 Digitalization and Industry 4.0

Since the start of the fourth industrial revolution over a decade ago in Germany as part of its government's high-tech strategy [Xu et al., 2021], and the introduction of Cyber Physical Production Systems (CPPSs) [Vogel-Heuser et al., 2012], several countries and entities have started their own initiatives. Originating from the United States, Advanced Manufacturing Partnership (AMP) encourages collaboration among industry, academia, and government to improve manufacturing competitiveness and accelerate innovation and technology transitions to the marketplace [Molnar, 2017]. It aims at advancing manufacturing technologies and processes in several directions including development and deployment of materials, energy-efficient processes, and advanced robotics, among others [White House Office Of The Press Secretary, 2011]. The United States also presented the "Manufacturing USA" initiative. Manufacturing USA Comprises various institutes, that aim to advance manufacturing capabilities through research, development, and deployment of cutting-edge technologies. Each institute specializes in different technology areas like 3D printing, advanced materials, and more [Shivakumar and Cohen, 2017].

On the other hand, China's road map for technological innovation revolves around "Made in China 2025" [Wübbeke et al., 2016], which use a top down approach to smart manufacturing, aiming to encourage enterprises towards more risk. Additionally, the Chinese industrial landscape started leaning towards automation after their worker shortages following the world economic crisis in 2008 [Wübbeke et al., 2016]. Further reforms that accompany this initiative include policy changes which allow foreign access to the market and modern production facilities. The Chinese government plans a strategically interlinked battery of industrial policies affecting large industrial sectors such as automotive and energy sectors. The policy reforms and the lean towards smart manufacturing and automation trickle down, backed by the government, to the different manufacturing parties with different sizes such as large- and small-enterprises which belong to both, government and private sector [Wübbeke et al., 2016]. Several other countries also presented other initiatives:

- Strategy for Scientific and Technological Development of the Russian Federation
- Japan's Industrial Strategy 2030
- Australia's Advanced Manufacturing Growth
- Saudi Vision 2030
- Dubai Industrial Strategy 2030

They all share a vision of increasing the applications of Artificial Intelligence (AI) and automation in addition to the integration of Industrial Internet of Things (IIoT) into the technical level of their respective industrial landscape. They also push for more communication within process components and in the supply chain.

Approaches that try to achieve the Industry 4.0 vision usually focus on using connected, communicating, intelligent machines to achieve robust, self-organizing reliable CPPSs [Xu et al., 2021]. Industrial initiatives try to fulfill this vision by using smart manufacturing systems, often associated with data collection and increased communication between machine and process components [Brauner et al., 2022]. This increases the process complexity due to the overhead in collecting, transferring, and processing the data [Hopmann et al., 2023]. The IoP tackles this overhead by several approaches:

- Using DSs instead of the widely used Digital Twins (DTs), which helps move some of the computations closer to the edge, reducing the time needed for data transfer.
- Less dependence on finite element simulations, in favor of purpose specific DSs to help in real time decisions, such as plan repair [Liebenberg, 2021, Liebenberg and Jarke, 2020], zero-defect manufacturing [Powell et al., 2022], and real-time robot behavior optimization.

However, they usually focus on a single layer of production, either communication within a company, or within the supply chain as seen in Figure 4.1. The communication within a company goes in two axes, the vertical one where communication goes between the upper level management of a company all the way to the low level sensors. It passes by everything in between, such as the process level and the shop floor level. This supports process automation and optimization, data collection and analysis, and the

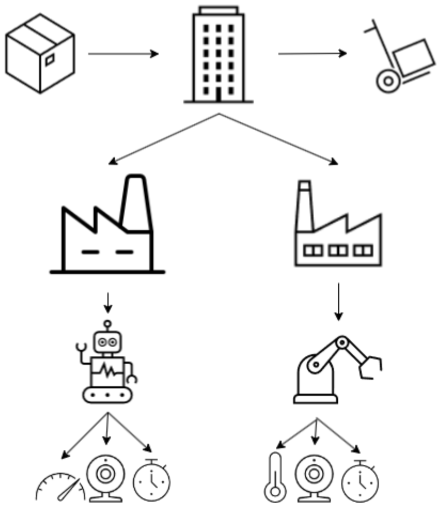


Figure 4.1: In the age of smart manufacturing, companies communicate in horizontal and vertical axes with the supply chain partners and the low level machines, respectively.

creation of decision support systems to help machine operators, supervisors, and managers [Pennekamp et al., 2019, Kagermann, 2015]. The horizontal communication axis handles the communication within the supply chain, allowing a company to reason about and optimize its logistics. The next Section discusses how the WWL plans to push the envelope by adding an extra communication dimension allowing a company to communicate with other, potentially competing companies.

4.1.2 Vision and Benefits of the WWL

In order to become the WWW equivalent of the IoP, the WWL goes beyond company and supply chain communication, by introducing the cross-company communication axis allowing potentially competing companies to communicate and share information as seen in Figure 4.2. To that end, it offers a set of tools, methods, and communication protocols to enable controlled global communication between the different industrial entities at different levels of the production processes. An infrastructure is provided to aid the digitalization efforts present in different Industry 4.0 initiatives. The envisioned infrastructure should not only allow the integration of sensors, machines, and processes, it should also allow their DSs to communicate with each other's within and beyond the borders of a single company. Designated partners should be able to query these DSs to extract information that helps replicate results or benchmark products and processes. The WWL ultimately becomes a lab-of-labs which provides a communication medium for querying DSs as well as exchanging data, knowledge, models, and software artefacts, analogous to a search engine in the WWW [Brauner

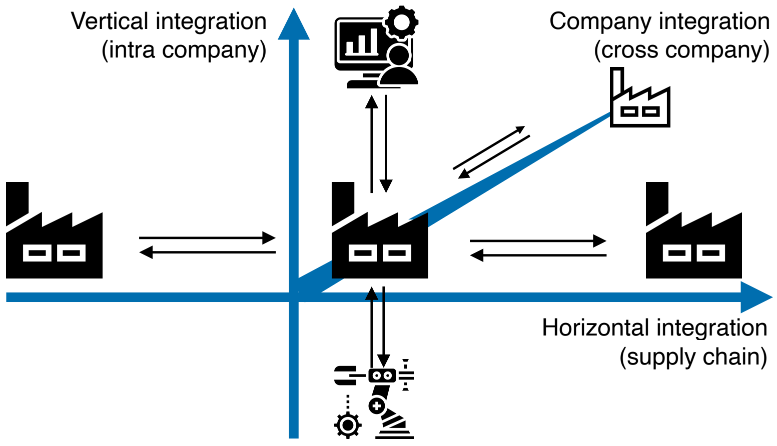


Figure 4.2: The three dimensions of communication offered by the WWL. Most other initiatives only focus on the horizontal and vertical integration.

et al., 2022]. The scientification of the industrial landscape offers savings of time and effort required for machine calibration and model training in addition to potentially more diverse datasets that result from the aggregation of different data available at the disposal of the different partners. This initially requires high usability on two fronts:

1. Machine integration through standardized interfaces and Application Programming Interfaces (APIs)
2. Graphical User Interfaces (GUIs) to support human workers holding different roles in extracting and inserting knowledge as well making use of the decision support systems in place

Some approaches to digitalization such as the International Data Spaces [Otto and Jarke, 2019] and GAIA-X [Braud et al., 2021] share goals with the WWL, namely controlled, secure, and sovereign data exchange and communication. However, the WWL sets itself apart from other initiatives by crossing domain boundaries on a global scale. It transcends the industry-specific focus often associated with applications of similar approaches. Additionally, unlike other Internet of Things (IoT) initiatives such as Linked Dataspaces (LD) [Curry, 2020, Almeida et al., 2020], the WWL exploits the structure of the connected entities using the process-based relations. This allows us to group different entities that interact with the same process, product, or machine to either apply traditional Machine Learning (ML) approaches or more advanced process mining techniques to make predictions, recommendations, or decisions. Furthermore, instead of relying on the widely spread DTs simulating different processes, the WWL is a network of DSs, this potentially moves the computations closer to the edge reducing the communication overhead and offering real-time computation capabilities for real time decision making such as re-planning or plan repair.

Finally, human workers, both blue and white collar, are integrated into the WWL as data sources and end-users. As data sources, they can define processes and insert knowledge in terms of process models and ontologies [Kluge-Wilkes et al., 2022]. On the other hand, their data is used –responsibly– to build human DSs [Mertens et al., 2021], which can be used to evaluate performance and make quality predictions and recommendations. They are also end-users as they receive the recommendations from Decision Support Systems (DSSs) and decide accordingly to make changes to a running process in order to increase quality, recover from defects, or otherwise enhance performance.

4.1.3 Challenges Facing The Lab-of-Labs

Due to the scope of the WWL's vision, we categorise the challenges into five aspects: technical, social, legal, economical, and ethical. The technical aspects form an infrastructure for the WWL. They provide the building blocks of implementation and enforcement of solutions to the legal, ethical, and economical challenges, which lead to acceptance by governments and enterprises of different sizes, as well as social challenges which target individual acceptance. The economic and financial challenges further include the costs and return on investment involved in maintaining such a global network. Solving the economic challenges leads to financial feasibility and stability of the WWL. A great deal of these challenges can be solved by monetizing the utility of the WWL and creating new business models on top of it. Additionally, the social challenges are step stones towards acceptance by the general public. All challenges are interrelated and interdependent, as seen in Figure 4.3, increasing the complexity of tackling them.

Technical Challenges

Technical challenges include performance requirements to enable and support real-time capabilities of the different connected entities as well as infrastructuring in terms of standardization, security, and data transfer. Standardization and regulation are needed to ensure Findable, Accessible, Interoperable, and Reusable (FAIR) data [Wilkinson et al., 2016]. This ensures the discoverability and connectivity of sensors and actuators as well as machines in a lab or production site [Bodenbenner et al., 2023]. On the other hand it overcomes the diversity of the datasets available in such labs and production sites, which may have different data formats, structure, and labeling due to the use of machines from different Original Equipment Manufacturers (OEMs). This increases the complexity and overhead of data aggregation and transfer [Khan et al., 2017], hindering the procurement of suitable data for training ML models. To mitigate these issues, we need unified data exchange formats with standard labeling, structure, linkage, and storage. More technical challenges arise in order to implement solutions for the remaining social, legal, economical, and ethical challenges.

Legal Challenges

Due to the high level of competition in the different industrial and manufacturing domains, companies of different sizes, and to a great extent research institutions and labs, are reluctant to share production data, even for research, in fear of losing the competitive edge. The legal challenge here lies in regulating the data exchange to ensure that data is used responsibly and within the bounds and permissions set by its creators. These bounds are to be ensured both legally and technically. To overcome this fear, the WWL needs to encourage participation through an initial peer-to-peer communication and data exchange, where an entity can select who gets access to their data. Each data producer should also have control over a set of parameters that collectively control, not only who gets access to its data, but also which data is shareable, according to purposes, granularity, and semantics, and define different access levels for different types of data and different data consumers.

Social and Ethical Challenges

Smart factories tend to collect huge amounts of worker data, both for performance and action analysis [Mertens et al., 2021, Pütz et al., 2022]. The analysis of such data under the labels of *Scientific Management* and *Taylorism* has resulted in increased safety, productivity, and economic importance [Mitcham, 2005]. However, the data comes from permanent and ongoing monitoring of these workers, this is often accompanied by a risk of data misuse leading to a reduction in worker autonomy. Therefore, to preserve these results, it is crucial to preserve the autonomy and privacy of the workers involved.

Data is often used to compare production sites and processes involving manual labour to learn new methods of resource allocation and identify optimal work setups. To enable this comparison on a larger scale, beyond company borders, an ethical requirement for the WWL is the protection of personal data of the workers. Moreover, data consumers are required to use the data responsibly and data producers, both companies and workers, should be guaranteed this responsible data usage. The WWL must ensure that data shared through it is being used responsibly and that scientific ingenuity is credited appropriately. This leads back to both legal and technical requirements to ensure data provenance and sovereignty and control of data producers over their data.

Economical Challenges

Similar to most platform and crowd-driven solutions, the WWL also faces the *Cold Start* problem. This refers to the relation between network utility and the size of the community. The return on investment for the different partners only starts to show once the network is big enough. This problem resides on top of the list of economical challenges facing the WWL. The

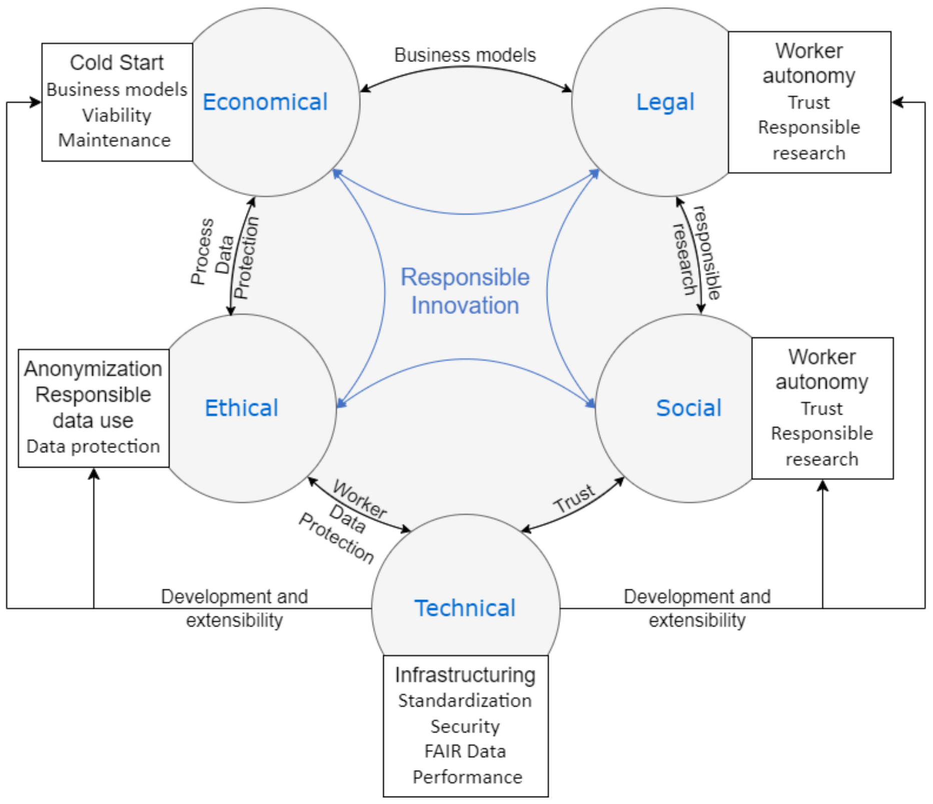


Figure 4.3: Illustration of the interwoven technical, legal, economic, social, and ethical challenges of building the WWL.

WWL needs to ensure its own viability by ensuring that it is well maintained and sustained. This can be done by using open-source platforms and standards as well as new business models that encourage a dedicated entity or entities to take over the maintenance responsibility in exchange for monetising the utility of the WWL. The economical challenges can be summed up in ensuring that the benefits of joining the WWL outweigh the overhead of integration and the risk involved in sharing data. This can vary depending on many factors as well as the goals and expectations of the different partners.

4.1.4 Road map to Acceptance

The challenges facing the WWL come from different aspects and therefore require coordinated efforts in different disciplines and even more interdisciplinary projects to tackle. Realising the full potential of the WWL is a long term endeavor that can be achieved in several working phases. This section gives an overview of the work done so far as part of the IoP towards the vision of the WWL as well as short- and long-term plans to reach a state of industry-wide acceptance of the WWL.

Existing Infrastructural Work

Several surveys cover the previous work towards these goals [Becker et al., 2021, Brecher et al., 2023, Brauner et al., 2022] from different perspectives. The surveyed work mostly covers the creation and use of DSs of different processes for different purposes. For example, [Liebenberg and Jarke, 2020] shows the creation and use of a DS for a metal forming process instead of a full Finite Element Method (FEM) simulation and how this allows real time re-planning. Additionally, [Liebenberg, 2021] shows a GUI for sharing and querying DSs between different users of different roles, they demonstrate this on the use case of metal forming and plastics processing. Several communication protocols and data standards were also introduced for data storage and querying, the main aim is to ensure data sovereignty and suitability for different purposes such as training ML models as well as analysis for process mining applications [Mann et al., 2020, Pütz et al., 2022, Berti et al., 2023, Ghahfarokhi et al., 2021]. Further work was done to include human machine operators into the WWL either working hand in hand with artificial agents in collaboration or by creating a human DS that uses human data to predict process performance [Mertens et al., 2021, Mertens et al., 2022]. These, among other work, handle a great deal of the technical challenges. They constitute the main building blocks of a WWL.

Proof of Concept

A proof of concept, in the form of a Minimum Viable Product (MVP), is needed in the immediate future, that not only integrates and groups the different building blocks together, but also demonstrates the infrastructure of the WWL. We define the minimum requirements for such an MVP as follows:

1. Integrating different processes from different industrial domains. As mentioned above, most of the building blocks implemented so far target specific use cases and rarely demonstrate transfer-ability across use cases.

2. Provides a set of APIs that allow the easy integration of new processes and industrial entities.

3. Demonstrate the different modes of communication between the different entities in the WWL.

4. Uses standard data formats and communication protocols for storage and communication between DSs.

5. Follows FAIR data principles, ensuring that the data is Findable, Accessible, Interoperable, and Reusable.

6. Ensures data provenance, protection, and anonymity.

The MVP should further be agile and extensible to allow the integration of the remaining requirements in the long term. These minimum requirements are needed to demonstrate some of the benefits of such an industrial network to give incentive to potential partners in order to gradually grow the network.

Further Down the Road

In the long term, we need to further integrate the non-technical requirements. These requirements, help with the acceptance of the WWL as a communication framework. To that end, we need to involve experts from the different disciplines, economics and humanities. Once the technical requirements are satisfied, we need more incentives to grow the network and face the cold start problem. New business models need to be investigated, that initially make use of peer-to-peer connections through the WWL. These should offer a sandbox that showcases the benefits on a small scale while at the same time, giving companies and partners full control over their data and access rights motivating more openness and transparency.

Providing a toolbox that facilitates interacting with standard and open-source data formats facilitating the conversion from the different formats, used by different OEMs to store generated data, to selected standards (e.g., Object-Centric Event Log (OCELs)). The WWL should exploit the existence of such standards by providing tools for pre-processing data before training ML models.

Finally, there should be dedicated entity/entities that maintain and sustain the WWL. These are needed to fund and maintain the hardware and software components that form the network. The existence of such entities highly depends on the monetization potential and the business models realizing it.

Last but not least, we need to look into the acceptance of the WWL as a new technology both on the corporate level and the individual level. In the former, we will focus on the return on investment and in the later, we investigate trust and safety. This can be studied through the Technology Acceptance Model (TAM) [Silva, 2015]. However, we should keep in mind that the two sides are related since the corporate acceptance starts by individual acceptance [Bach et al., 2016, Horton et al., 2001]. According to [Paredes-Gualtor et al., 2017], corporate technology acceptance is a 6-step process as follows:

1. **Current State Definition:**

   The MVP, should help companies view all the available features and building blocks so they can assess it with respect to their business models and use cases

2. **Requirements Definition:**

Potential partners should have access to the API, which allows them define what their respective systems need for integration into the WWL.

3. **Providers Analysis:**

   This targets analysing the services and products from different providers. In the WWL case, we aim to have a single global overarching network of industrial entities.

4. **Providers Assessment:**

   Companies should be able to assess the different providers in terms of usability, feasibility, and benefits. In the WWL, we aim for a singular provider, but companies can still use the MVP to compare the WWL to traditional communication schemes in terms of these attributes.

5. **Decision:**

   The decision to adapt a new technology requires studying and weighing multiple factors based on use cases and business models. Many approaches exist for such decisions, surveyed in [Pantelić et al., 2016], but are outside the scope of this thesis.

6. **Migration:**

   This will initially happen using the MVP, which will evolve gradually to include more features and fulfill more requirements. Eventually, it should pass the peer-to-peer communication stage as it gathers momentum and populate its network.

The remainder of this chapter focuses on one of the building blocks of the WWL, creating DSs for industrial robotic tasks.

4.2 Digital Shadows of Production in the WWL

As opposed to the more prevalent and almost omnipresent DTs, the WWL uses DSs to represent different aspects and entities of industrial processes. Each entity in the WWL is represented by a DS making the WWL a network of DSs. The set of possible entities includes, but is not limited to, processes, machines, products, or agents.

DSs are context- and task-specific representations of an entity. They have been used in several scenarios and for different purposes, which include process modeling and task execution and monitoring among other things. They are usually integrated into systems used for process planning, data analysis, zero-defect production and plan repair, and decision support systems [Brauner et al., 2022, Liebenberg and Jarke, 2020, Becker et al., 2021].

Traditionally, digitalization was approached using DTs. These are one-to-one models and simulations of their physical entities. For example,

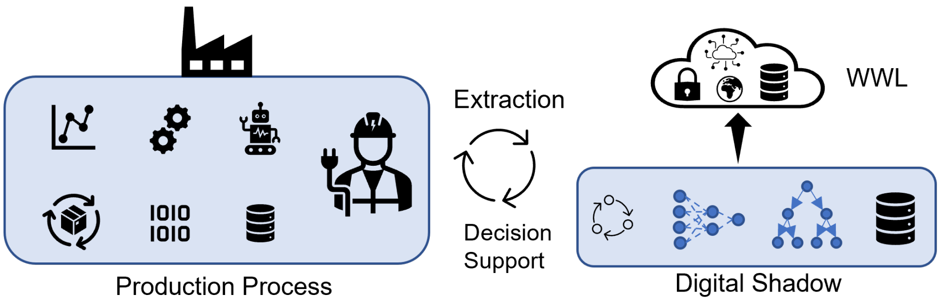


Figure 4.4: The architecture proposed by [Liebenberg and Jarke, 2020]. Human operators add, introspect, and query DSs of task representations of different agents.

FEMs [Zienkiewicz et al., 2005] were used to simulate a metal rolling process [Seuren et al., 2012, Bambach and Seuren, 2015]. However, since FEMs often include descritizing the simulation subject into finitely many elements and then processing them, it can be computationally expensive and time consuming. Depending on the process and the simulated object(s), such a simulation can take days, precluding their use in real-time situations such as defect prediction and plan repair [Liebenberg, 2021].

The WWL overcomes this by using DSs instead of DTs. Since DSs are projections of processes and entities, they are able to abstract away from the details and focus on a given aspect, reducing the amount of computation required. Their main purpose is usually to model the behavior of an asset [Becker et al., 2021]. A DS might also be an analytical model or a neural network modeling this behavior by predicting a property of the asset or classifying the process according to some criteria. Compared to DTs, they are more suitable for handling real time problems and edge-computation scenarios.

As proposed by [Piller and Nitsch, 2022], the WWL uses them not only for modeling industrial process, but also for cross company sharing of knowledge. Figure 4.4 shows an overview of the proposed architecture of the WWL [Liebenberg, 2021]. The WWL allows developers to create, share, store, and query DSs through web interfaces or an API. The WWL is able to offer different interfaces depending on the user's access level and role. A data analyst is able to query DSs related to data visualization and analysis to extract insights. A process supervisor is able to examine process data to examine where bottle necks may be.

More examples exist, where DSs are used for industrial applications inside and outside the WWL [Schuh et al., 2021, Ladj et al., 2021, Bergs et al., 2021, Becker et al., 2021].

[Bauernhansl et al., 2018] presents a set of requirements, open challenges, and a road map for DSs in production. In this road map, a DS can be in one of four complexity levels, each of which is further split into four stages. Some of these stages are already realized by many models but some of them are open problems that require further investigation. We can see below that some of the described complexity levels are missing their first stages. This is because the missing stages are still open problems in their respective fields with no immediate solutions.

Level 1. Information Linkage Supply and Demand

At this level of complexity, defining and linking information suppliers and consumers, i.e., the DS has to have an interface to its environment allowing it to connect to other entities.

Stage 1: Automated Registration

There has to be a manual method to link and define interfaces for each entity in the system. Moreover, the system should allow the entities to automatically register, informing the DS of their interfaces as well as the semantics of the messages that can be exchanged.

Stage 2: Information Linkage

Automatic linking of information producers, consumers, and processors should be possible without human intervention. This requires a directory that registers each components, e.g., `rosmaster` that keeps track of the different ros nodes and their services.

Stage 3: Automated Semantic Analysis

The DS should be able to automatically analyze the semantics of the services provided by other entities to supply information.

Stage 4: Data Valuation

At this stage, the DS should treat the information provided as a resource and able to valuate it. This is particularly important in cases where the information suppliers and consumers reside in different companies.

Level 2. Information Flow Control

Linking information processing services to data suppliers and information consumers

Stage 1: Automated Linking of Control Services

This stage allows automatic linking of control elements and services for providing or consuming data. This requires components to be registered in directory services. However, the semantic analysis of these components and the services and data they provide is still manual.

Stage 2: Support of Different Control Logics

The DS should support different logics that allow the exchange of data, information, and knowledge such as pushing and pulling.

Stage 3: Automated Adaptation of Control Logic

The DS should be able to react to the shifting information supply and demands and flexibly adopt new information flow control logics.

Level 3. Information Quality Control and Feedback

The ability to inspect and evaluate data quality and give feedback. The DS should be able to filter information according to a given quality criteria and ultimately search for information suppliers that provide higher quality data.

Stage 2: Only Use Data of Given Quality

At this stage, the DS should be able to compare the data and information quality to a given threshold based on given criteria some information will pass to the rest of the system while others will not.

Stage 3: Feedback Regarding Data Quality

At this stage, the DS passes all the information regardless of quality but gives some information regarding the quality of the transfered data.

Stage 4: Automatically Optimise Information Quality

Given that the provided information has varying quality and that the DS is able to assess the quality and give feedback, at this stage, it should be able to maximize the information quality supplied to the system by making decisions regarding the data suppliers to choose those providing highest quality data when available.

Level 4. Self Optimisation

The ability to recognise unused data and information suppliers, eliminate or compress them, and make proposals for alternatives all in the direction of self optimization of the DS and therefore the entire system.

Stage 2: Automatically Eliminate Unused Data Sources

The DS should be able to disconnect and delete any data source that remains unused. It should be able to reason before doing this based on some deletion compliance criteria.

Stage 3: Propose new data sources

At this stage, the DS is able to make suggestions to expand the information supplier pool by automatically proposing new information suppliers.

Stage 4: Automatic Discovery and Connection with Other DSs

This stage takes the proposals even further by allowing the DS to automatically connect and exchange information with other instances of DSs.

As shown in [Colledanchise and Ögren, 2018] and [Iovino et al., 2020], BTs have crossed several milestones allowing them to become a reliable behavior model for several types of robotics applications, ranging from domestic service to military ones. In our work, we show that BTs in their current state of development with the available extensions and synthesis approaches are fit as a standard DS for industrial robot behavior, conceptually ready for integration into the WWL.

The difference between a behavior model and a behavior model DS is that a DS is a projection of this model that can be shared, queried, and composed, in addition to the requirements and development stages defined mentioned above. Our work shows where BTs stand with respect to the development road map defined by [Bauernhansl et al., 2018].

We propose extending the framework proposed in [Liebenberg, 2021] by allowing users to share BTs in the WWL as seen in Figure 4.5. This allows users with different roles, such as managers, engineers, and process supervisors to create, share, and reuse DSs represented as BTs. This also enables agents to infer the capabilities and objectives of other agents and communicate with their DSs. The next section goes over the details of this proposal.

4.3 Behavior Trees as a DSs for Task Modeling

After the success shown by BTs as an activity representation for robots and artificial agents, we propose using them as a standard DS representation of industrial robot tasks. Enhanced by the extensions proposed over the past decade, they have already crossed several milestones on the DS road map proposed by [Bauernhansl et al., 2018]. This section goes over the stages of DS development showing how far BTs have come with respect to the road map through their several desirable properties as well as their extensions.

4.3.1 BTs as Behavior Models in the WWL

This section shows the state of BT development with respect to the developmental road map introduced by [Bauernhansl et al., 2018]. We go over the different stages of development and levels of complexity showing how far BTs have developed. Multiple approaches simplify BT programming for

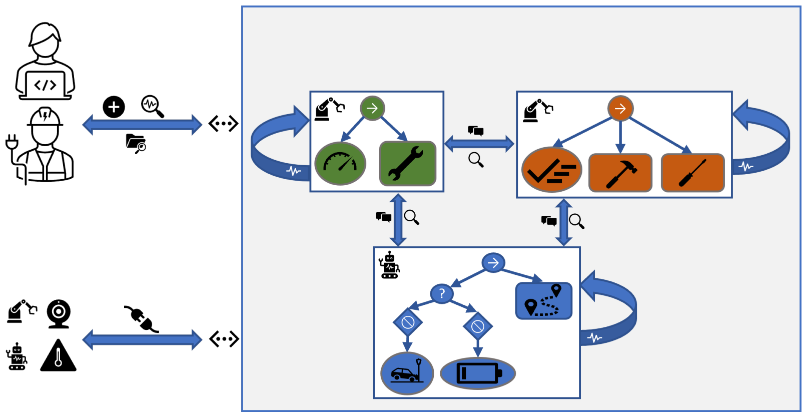


Figure 4.5: An overview of the proposed DS model based on the requirements in [Bauernhansl et al., 2018] integrated in the WWL architecture [Liebenberg and Jarke, 2020]. DSs are able to automatically self-optimise as well as register and communicate together through API. Human operators and managers can add, introspect, and query the DSs of agents.

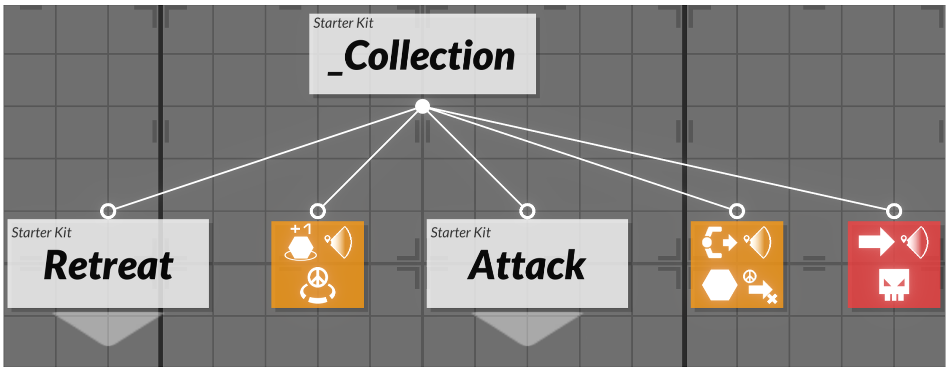


Figure 4.6: BT editor used in [Gladiabots, 2019] to model behavior of battle robots.

end-users in different domains for different applications [Iovino et al., 2020]. These approaches include [Gladiabots, 2019], which uses an in-game interface allowing players to edit and create BTs and execute them by a swarm of robots in combat as seen in Figure 4.6. Additionally, a general purpose open source editor, shown in Figure 4.7, allows developers to create, edit, and export BTs in addition to defining new action and condition nodes. This section revisits the different milestones mentioned in Section 4.2 elaborating where BTs stand as a task-level action representation framework relative to each development milestone and complexity level.

Level 1. Information Linkage Supply and Demand

Although BTs don't intrinsically support the automatic discovery of and connection to sensors and systems, the robotic ecosystems and middle-ware such as Robot Operating System (ROS)

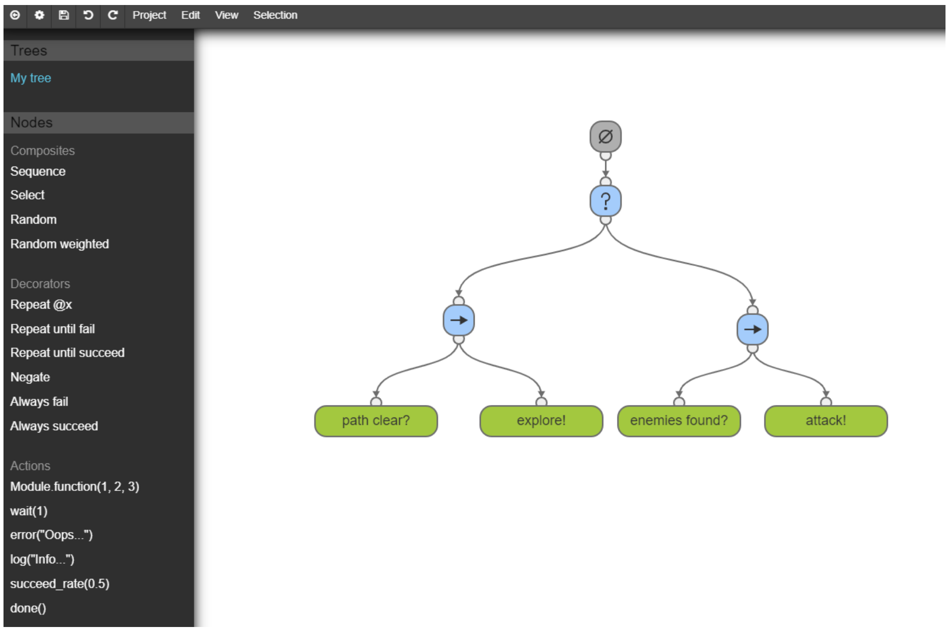


Figure 4.7: An opensource general purpose BT editor [1]

allow robot components to publish and subscribe to topics systematically. Additionally, ontologies can encode information about the available sensors in a system as well as different action affordances by different objects and agents [Kluge-Wilkes et al., 2022]. At run-time, an agent can query an underlying ontology and automatically connect to appropriate sensors and actuators, based on the task at hand. We have seen similar behaviors tree synthesis approaches, where goal conditions are expanded into condition and action nodes at run-time using backward chaining [Cai et al., 2021]. These nodes are able to subscribe to topics as well as send and receive messages from other components of the system.

By combining a BT with an appropriate framework and an underlying knowledge base, such as an ontology, we can manually and automatically discover and connect to different components. This covers the first development stage of the first level complexity for a DS.

In the second stage of development, a DS should be able to provision, log, process, and archive data and information autonomously. This is achieved in the commonly used BT frameworks and implementations (e.g., `py_trees` [2] and `py_trees_ros` [3]),

[2] https://github.com/splintered-reality/`py_trees`
[3] http://docs.ros.org/en/kinetic/api/`py_trees_ros`/html

which offer a blackboard and are able to subscribe to ROS's topics, and parameters for logging, processing, and decision making.

Automatically analysing the semantics of the processes involved in the information exchange has not yet been discussed in the BT literature. However, many process mining approaches are able to learn process models and use them to make inferences. It would be possible, in principle, to make the connection between these approaches, but for now this stage of development is left open.

The fourth and last stage of development involves putting a price on the exchanged information between companies and divisions within a company. This is out of the scope of our work, as it involves interdisciplinary studies as well as practitioners and experts of different fields. This type of communication is one of benefits of the WWL. We leave this question to future work, when BTs have been fully integrated into a MVP of the WWL. Only then would companies have a clear view of the kind and value of information they can share. These can range from raw sensor and meta data to sub trees and even full behavior models, that can be plugged on top of running systems as a plug-and-play models by connecting to the underlying robot controllers and drivers.

Level 2. Information Flow Control

The first stage of this complexity level has already been long standing in both BTs and several robot middle-wares, such as ROS. Manually linking control systems has been a feature of ROS as well as other robot control frameworks ever since its introduction as both a commercial and a research framework. Due to its modularity ROS can hand the control of the robot to any of its nodes, which offer several layers of abstraction as well as control flows. This also applies to BTs since their introduction as a game character control framework, due to their modular structure, which allows the agent to choose which components or actions to perform. In the context of robotics, the leaf nodes control the robot's actuators depending on the state of the environment and the task at hand [Colledanchise and Ögren, 2018]. These nodes can be exchanged through the WWL encapsulating the entire behavior of a robot or only parts of it to guarantee safety or perform sub-tasks.

The second stage focuses on the ability of the DS to handle different types of control flow logics. This can be seen in several BT extensions, which allow the tree to build itself at run time [Colledanchise et al., 2019] to satisfy goals, allow Human-

Robot Collaboration (HRC), as discussed in Chapter 6, or multi-agent interaction [Wang et al., 2020] among others.

At the third and final stage of this level, the tree should be able to self-adapt to dynamic environment. This has also been demonstrated by expanding trees at run-time to satisfy goals [Colledanchise et al., 2019], using Reinforcement Learning (RL) and Genetic Algorithm (GA) to synthesize trees [Colledanchise et al., 2018], optimize trees at run time [Merrill, 2019].

We can see that BTs have developed beyond this complexity level with all its stages. The challenge remains in integrating the different extensions and approaches in a unified framework. This can be done by choosing a standard set of extensions with non-conflicting semantics, or allowing the users to pick and choose which extensions to use through an interface and using a unified format for storing the trees.

Level 3. Information Quality Control and Feedback

The third level of complexity had no first development stage defined in [Bauernhansl et al., 2018] since automatic assessment of data and information quality is an open problem with no immediate solution. The road map of [Bauernhansl et al., 2018] defines a vision for the three remaining stages. In the second stage, a BT should be able to decide whether the incoming data meets certain quality standards, which is not an inherent ability of BTs. However, since BTs have access to the available data streams, they can either use off-the-shelf anomaly detection approaches to decide on the noise ratios and whether the data sources are reliable enough. The third stage builds on this allowing the trees to collect and use data regardless of its quality and later analyse it, allowing decision makers to draw conclusions and make the final decisions regarding reliability. The fourth, and last, stage focuses on allowing a BT to autonomously choose new information sources (e.g., sensors), again depending on the quality and reliability of the source. Some BT synthesis approaches [Colledanchise et al., 2018, Iovino et al., 2021, Colledanchise et al., 2019] are close to this, but instead of focusing on information quality, they focus on task completion. However, they can be combined with Information Quality Assessment (IQA) approaches (surveyed in [Becerra et al., 2021]) to cross this milestone. This is particularly doable since BTs offer an alternative to online re-planning, thanks to their reactive modular nature.

Level 4. Self Optimisation

The focus of this complexity level is to give self-optimization and modification capabilities to the DS. In the context of BTs, this would mean modifying the structure and the actions to execute as well as the conditions to check. The first stage of development is still unrealised and was therefore not defined in [Bauernhansl et al., 2018].

The second stage handles the selection and elimination of unused data sources, such as sensors and other agents. This was demonstrated in several BT-based approaches, which can prune the tree [Colledanchise and Ögren, 2018, Jones et al., 2018, Colledanchise et al., 2018] after their creation by removing unused branches. For example, if we build a tree for a mobile robot to execute a certain task, this tree would contain branches for navigation and obstacle avoidance. These branches would be unused if the tree is applied for a fixed-base robot.

The third stage focuses on automatic information acquisition. This means that a DS should be able to extend its list of information sources and query them when needed. In a BT-driven robot, this would mean that the robot is able to add a new branch in the tree that queries certain sensors, or decide to communicate with and query another agent that is able to provide some previously unknown information about the world. This was described in several BT synthesis approaches which are able to grow the trees at run time [Iovino et al., 2021, Colledanchise et al., 2019, Cai et al., 2021]. Some of these approaches seed the tree using a condition node describing a goal state, then incrementally add action nodes that satisfy some conditions of the goal state, with guarding condition nodes that ensure the preconditions of these actions are satisfied before executing them. The guard nodes act as new intermediate goals, then the process is repeated till an executable tree is created. Other approaches add a learning node in the tree. This node is an action node that uses a genetic algorithm to select a new branch to extend the tree, then append a new learning node at the end. This way, when the agent reaches an unseen state, it can grow the tree to handle the new state.

The fourth and final development stage allows the DS to form new connections with other DSs or agents. Although this has not been part of the work on BTs yet, a multi-agent BT approach already exists, where the tree can request help from other agents by assigning tasks to them in auctions [Wang et al., 2020]. In this approach, the agent is not able to automatically discover new agents, but agents are able to broadcast new goals, and other agents are able to decide, based on their own capabilities whether they can achieve it and make bids accordingly. All agents in [Wang et al., 2020] are assumed to know and communicate with each other. To achieve this level of complexity, BTs

> should be augmented with underlying knowledge bases that can dynamically register new agents along with their capabilities. This allows an agent to discover other agents, along with the action affordances as well as the information they can provide.

Even though BTs have crossed several milestones bringing them closer to becoming a standard DS for robotic assembly processes, they still need to cover more ground in order to fulfill all the cells of the matrix defined by [Bauernhansl et al., 2018]. Some of these cells still need a solution to open problems or input from experts in several fields, such as information quality assessment as well as information valuation. Others have been demonstrated either by tree synthesis approaches or tree extensions that emerged over the past few years [Iovino et al., 2020]. The next section shows a use case describing the use of BTs in the WWL as well as using Control Barrier Function (CBF) as a DS of safety and how this can be integrated into the WWL.

4.3.2 Digital Shadows of Robot Safety

We further propose a DS for safety functions that can be used as a plug and play solution for requirements, safety functions, and benchmark. This not only alleviates engineering efforts involved in code reuse, but also bridges the gap between different entities in the WWL and potentially between academia and industry. This section details how these functions can be integrated in the WWL.

CBFs [Ames et al., 2019] have long been used for safety of dynamic systems. A CBF is a function $h : R^n \to R$ that assesses the safety of the world state. Given an n-dimensional state description vector $\mathbf{x}$, it returns a heuristic value rating the safety of $\mathbf{x}$ classifying it into safe and unsafe state sets, $\mathcal{S}^+$ and $\mathcal{S}^-$, respectively. Such that $h(\mathbf{x}) \geq 0$ iff $\mathbf{x} \in \mathcal{S}^+$ and $\mathbf{x} \in \mathcal{S}^-$ otherwise.

[Özkahraman and Ögren, 2020] have recently shown that CBFs can be combined with BTs and used as safety guards for the system by using Control Barrier Function Behavior Trees (CBFBTs). They show that this combination, not only ensures the safety of the system, but also allows a BT-agent to pursue multiple goals with predefined priorities.

The WWL can maintain a pool of condition nodes, each encapsulating a CBF that is able to evaluate the safety of a given world state. Each node has an API, which states the topics needed by the node as well as a documentation explaining the semantics and limitation. When the node is integrated into a tree, it is parameterised with the topic names needed for its operation. At run time, it would connect directly to the specified topics acting as a safety guard to the branches executing the core tasks as seen in Figure 4.8.

For example, given the use case of gear assembly, seen in Figure 4.9, where a robot arm is tasked with assembling a gear. There are several

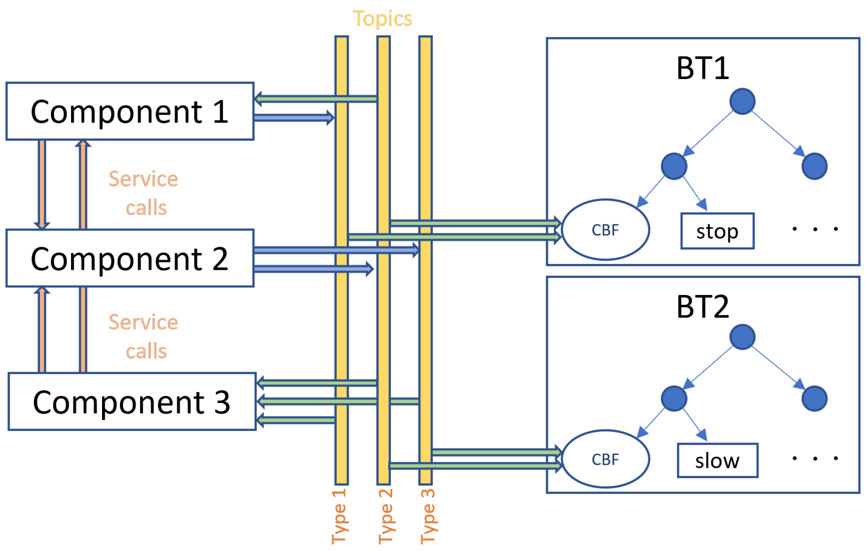


Figure 4.8: Integrating CBF nodes into ROS using BTs. They can subscribe to topics that provide messages defined in their APIs.

safety requirements to consider, especially when humans are involved:

1. Avoiding collisions:

   Avoiding collision with the environment, which is not always possible in a dynamic environment. In case a collision has occurred, the robot needs to react by stopping immediately to prevent injuring a shopfloor worker or damaging itself or the work pieces.

2. Dropping an object:

   When an object slips out of the gripper, the robot should react by either adjusting its grip and picking the object again if it has already dropped it, or picking a new instance of the object if it has become unreachable as a result of dropping.

A naïve BT to execute this task is shown in Figure 4.10. The tree contains a sequence node that executes the actions in sequence without checks or conditions. It moves the gripper to the location of a nut, picks it, and then moves it to the respective bolt, where it is placed. A more sophisticated CBFBT is shown in Figure 4.11. It contains condition nodes that encapsulate a CBF for detecting anomalies in the data stream. This can either be a clustering approaches, statistical approach such as a Gaussian Mixture Model (GMM) [Reynolds et al., 2009], or a filtering-based approach such as [Ding and Goshtasby, 2001]. These are examples of domain agnostic anomaly detection methods, they can simply, and interchangeably, be connected to a given topic as seen in Figure 4.8 and detect abnormalities in the data stream. The designer can choose what the abnormality entails

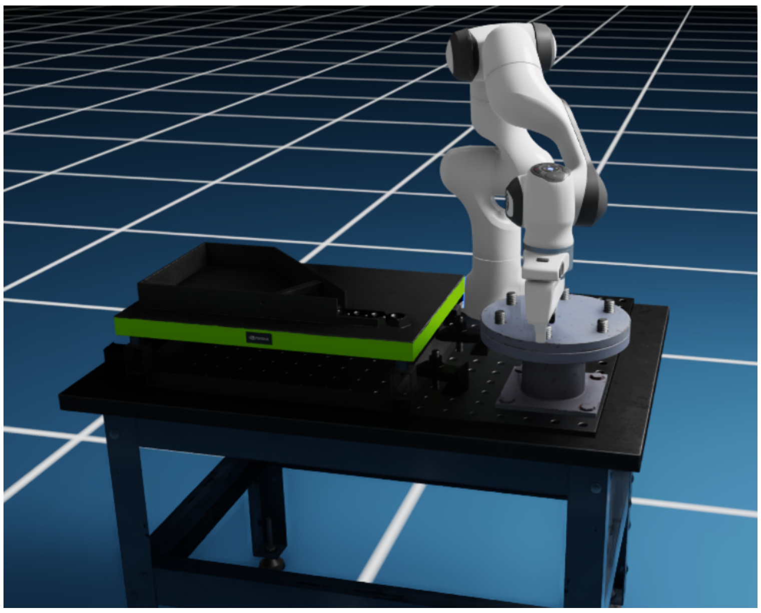


Figure 4.9: Assembling a set of nuts and bolts using Franka Emika robot arm. The robot is provided a set of nuts in the container to its right, which should be assembled onto the bolts to its left. All nuts are identical, the robot needs to pick one and assemble it onto the next available location.

in the given domain and the given data and how the robot should react in the respective cases.

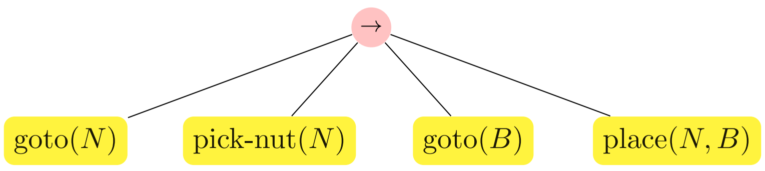


Figure 4.10: A simple BT to assemble nuts and bots using a robot arm. The tree sequentially executes actions to move and pick the nut, then move and place it.

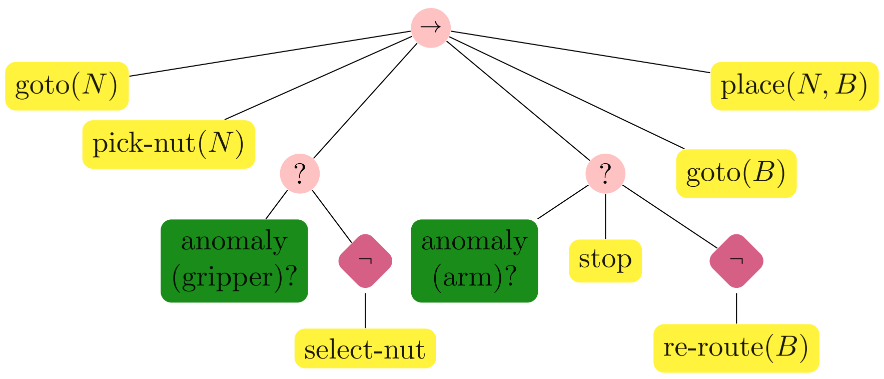
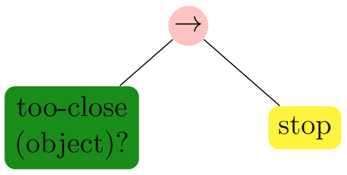
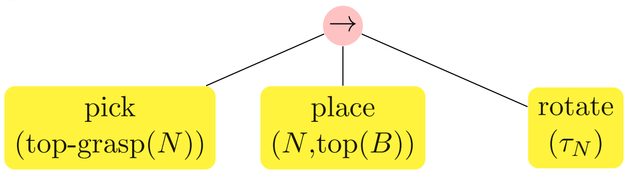


Figure 4.11: A CBFBT to assemble nuts and bolts including safety checks and guarantees. The tree contains condition nodes that detect anomalies in certain joint torque sensors (gripper and arm). In case a an anomaly is detected in the gripper, this indicates that the nut has been dropped, in which case the robot needs to select a new nut. The ¬-decorator is an inverter, which will cause the selector node to fail and thus restarting the tree. The second selector child has a similar behavior, when an anomaly is detected in one of the arm joints, this indicates a collision, the tree stops, and creates a new route. After that, the decorator node causes the selector to fail, which in turn causes this branch of the tree to fail and restart.

A vision of the WWL is to maintain a storage of DSs which can be complete trees, branches, or single nodes. This storage allows knowledge sharing as well as artefact exchange between different users. CBF nodes can be used as safety requirements and can be used as safety guarantees. Trees and tree branches can model different safety behaviors defined by operational safety experts, as seen in Figure 4.12a or be used as models for the assembly of different parts, defined by the parts' OEM, as seen in Figure 4.12b. Since the BTs seen in Figure 4.12 are robot-agnostic, they can be used as stand alone branches that are integrated directly into other trees that either need such behavior for increased safety, or need to assemble this specific nut as part of their product.



(a) In case an object is too close, the robot should stop.



(b) The nut should be grasped from above, placed on top of the bolt, and then rotated with a given torque, τ_N.

Figure 4.12: Two stand alone behaviors that can be stored and retrieved in the WWL by different users in different domains.

Despite the attention that BTs have received and the efforts invested into them, more work needs to be done in order to fulfill the vision of the WWL for using them as a standard representation in industrial tasks. Because the WWL has usability requirements in addition to the technical specification and power of representation required as a DS. The importance of usability is further increased, particularly in the case of Small and Medium-sized Enterprises (SMEs), which often lack robotics and programming experience [Berger and Armstrong, 2022, Perzylo et al., 2019]. The next chapters, describe a few BT extensions that target usability and readability of BTs allowing to represent tasks that were previously not supported as well as reduce the number of nodes needed to represent a given task using traditional BTs.

Chapter 5

Digital Shadows for Robot Teleoperation

Despite the industrial landscape's lean towards automation and repeatability, some manufacturing tasks did not make the leap to full automation. They rely heavily on human expertise and are still performed manually, using machines and robot arms that are fully operated by human workers. The automation of these tasks and processes has so far been infeasible, either because they require precise and fine grained manipulation of the work-pieces, or lack non-destructive quality tests of the work piece in order to take in-process decisions. In manual execution of the process, these decisions are taken by the human worker based on their experience.

This chapter discusses a framework we propose to assist the operators of these machines throughout the process execution. The framework targets the process of open-die forging, a common step in metal forming, used to change the geometry and micro structure of metal ingots as described in Section 3.2. The next sections give an overview of the framework followed by the details of the individual components forming it. Parts of the framework discussed in this chapter were published in [Behery et al., 2020], the rest is discussed in [Krömker, 2020].

5.1 Framework Overview

Our framework, shown in Figure 5.1 has five main steps, data collection, temporal action localization, action model inference, behavior library learning, and conformance checking. Data collection is a common step in data-driven industrial processes, in this step, we collect the data from the manual execution of these processes.

This includes the teleoperation data, where we capture and store the operator's commands, given using a joystick as well as the position of the metal ingot as the operator moves it during the process using a robot arm as shown in Figure 5.2. We annotate the data manually to locate, classify, and discretize the actions performed by the operator. After that, we learn

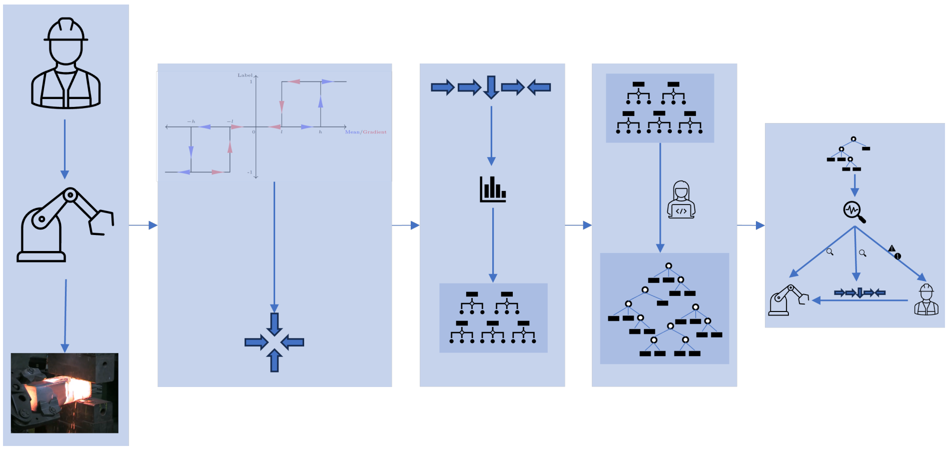


Figure 5.1: An overview of our proposed framework. Starting from data collection and annotation, using discretization and model learning, we build a BT library, that can be used to evaluate the operator's performance and offer support.

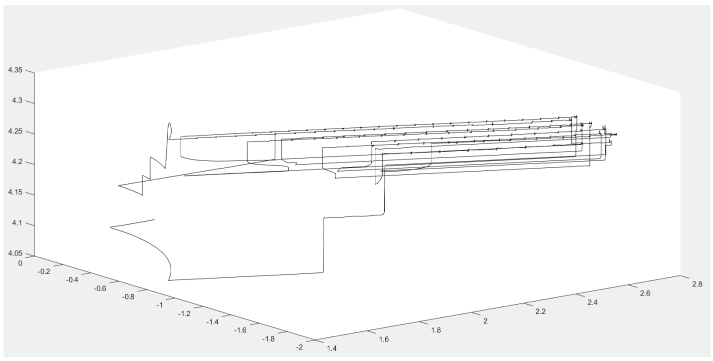


Figure 5.2: The gripper's trajectory during a sample execution of the open-die forging from our dataset. Units on the three dimensions are in Meters

the patterns of actions performed throughout the process and identify the most common ones, then infer their Planning Domain Definition Language (PDDL) models, their preconditions and effects. This gives us a library of commonly used actions and their symbolic models. We can use these to compose new behaviors, or learn the operator's behavior model and evaluate the behavior for conformance at runtime.

5.2 Action Extraction

To extract the primitive actions from the data, we model a function $f : \Theta \rightarrow \langle a_x, a_y, a_z \rangle$, whose input $\Theta \in \mathcal{C}$ is a sequence of joint configurations and the output is a 3-tuple of labels, each $a_i \in \{-1, 0, 1\}$ representing motion in the depth, horizontal, and vertical axes. The labels $\{-1, 0, 1\}$ represent

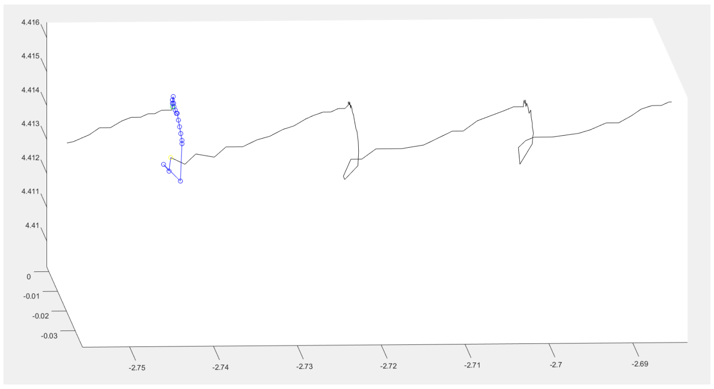


Figure 5.3: Undesired movements of the robot's end effector, which can be caused by noise in the input. Units in the three dimensions are in Meters

motion in the negative direction, idling, and motion in the positive direction relevant to the respective axis of Cartesian space, respectively. Our work in uses Hysterisis thresholding, commonly used in the Canny edge detector [Canny, 1986] to detect changes in the actions of the operator. We compare our work to other approaches which rely on modern neural networks used for action recognition in image sequences (videos) [Lee et al., 2017]. We also compare this work to unsupervised approaches used to cluster actions during surgery [Krishnan et al., 2017]. Moreover, the data we used is available for replication [1].

5.2.1 Data Acquisition

The process is performed using a heavy-duty 6 Degrees of Freedom (DoFs) forging robot from GLAMA Maschinenbau GmbH. The robot features both sliding and rotating joints as shown in Figure 5.4. During the process, the arm moves the metal ingots between a furnace and a forging press by SMS Group GmbH, capable of forces up to 630 tons. We recorded the joint configurations, with frequency of 50Hz and accuracy of $\pm 0.05°$, from 5 executions of the forging process. We also calculate the corresponding end-effector poses, depicted in Figure 5.2, using the forward kinematics solver provided by GLAMA Maschinenbau GmbH.

One execution of the process, e.g., illustrated in Figure 5.2 is represented as a sequence of approximately $15,000$ joint configurations, starting with the furnace withdrawal and ending with the billet deposition for cooling.

Table 5.1 shows the different parameters used for the forging processes with their respective ranges used during the data collection. The next section explains how we used this data to evaluate our discretization approach and to train a state-of-the-art action recognition model for comparison.

[1] https://github.com/behery/open-die-forging-teleop-dataset

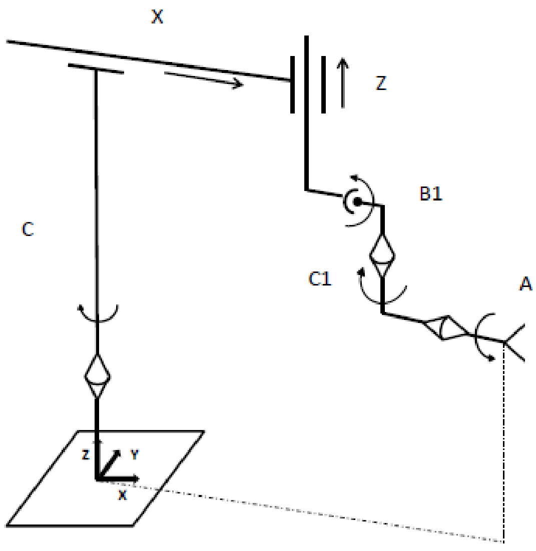


Figure 5.4: Diagram of the robot arm used for open-die forging annotated with the joint names.

5.2.2 Action Discretization Using Hysteresis Thresholding

To discretize the actions, we need to locate where the operator input signals change. Change alone is not an indicator because the operator uses an analogue joystick as an input device. This joystick can have some noise and unnecessary or unintentional movements caused by an operator's shaky hands as seen in Figure 5.3. Therefore, we do not rely only on the input velocity commands alone, but on a mixture of velocity commands, velocity gradients, and end-effector positions.

Inspired by the Canny edge detector [Canny, 1986], which relies on applying hysteresis thresholding to pixel gradients, we apply the thresholds to the end-effector velocity and gradients. The algorithm used in this work takes the high and low thresholds as part of its input as illustrated in Algorithm 1.

We first calculate the differences between every two subsequent configurations θ_t and θ_{t-1}, at times t and $t-1$ respectively, in each dimension $\hat{\theta}_t = \theta_t - \theta_{t-1}$. We smooth the data in each dimension using a box filter, which calculates the unweighted average of a neighbourhood of radius k, where k is a hyper-parameter of the approach that can be tuned to increase or decrease the smoothness of the curve. We apply the thresholding to the smoothed curve, $\bar{\Theta}$ to locate the start of an action, when the difference between the velocity and the mean neighbourhood velocity is higher than the high threshold h. The end of an action is marked when the difference between two data points is lower than the low threshold l. Using the initial

Parameter	Values	Notes
Forging Direction	{feeding, withdrawal}	decides which end of the billet is forged first during the process
Material	{C45, 42CrMo4}	mild steel and heat treatable steel respectively
Temperatures	[1150 °C, 1200 °C]	furnace temperature to heat the metal before forging
Height Reduction	[3, 22] %	
Bite Width	[20, 110] mm	distance between each stroke
Number of Passes	[5, 8]	
Number of Strokes	[93, 302]	
Operator	{1, 6}	years of experience
Start Geometry	{80x80, 150x150}	mm^2
End Geometry	{45x45, 50x50, 80x80}	mm^2

Table 5.1: The parameters used in the forging processes to record the data.

smoothing and the two thresholds h and l allows us to distinguish noisy values from slow or small actions. This is visualized in Figure 5.5, where the mean is shown in blue and the gradient is shown in red and the action labels are plotted on the Y-axis.

Algorithm 1 discretises and labels the actions using 0 to denote idling or no change in a given axis, and $+1$ and -1 to denote motion in the positive and negative directions respectively. Line 2 creates a box filter with a total length of $2k + 1$ time steps. We then calculate the gradient $\hat{\Theta}$ in Line 3 and smooth it using the box filter in Line 4 using the convolution operator $\circledast$ to get the average difference across the neighbourhood. We assume the data recording and processing starts from an idle state and therefore label the first action as 0 in all dimensions as seen in Line 5.

The thresholding process can be seen between Lines 6 and 18. It relies on two thresholds, h, which is used to decide whether a new action has started, and l which is used to detect whether it continues. If the average change at time step t, $\bar{\theta}_t$ is higher than h, we label this point as 1 indicating an action and backtrack till the start of the action; where the gradient $\hat{\theta}_t$ is less than l as seen in Function *labelAndBacktrack* on Line 20. We mirror this for actions in the negative direction. As long as $\hat{\theta}_t \geq l$, the label remains as 1. A change in direction is detected at time step t where $\hat{\theta}_t \leq l$ *and* $\bar{\theta}_t \leq h$, in this case, we label the the action as -1 to indicate an action in the opposite direction. Otherwise, if $\hat{\theta}_t < l$, we label the action as 0 to indicate no movement on the respective axis.

The rest of Algorithm 1 shows how we mark the start and end of the actions. When $\bar{\theta}_t > h$, we label the action as 1 and backtrack to the start

Algorithm 1: Hysterisis thresholding used in our approach. Action labeling for each of the 3 axis of the Cartesian space. For readability, we assume all variables are accessible in all 3 functions.

```
1  Function main:
       Input: $\Theta \leftarrow$ a sequence of end-effector positions
              $k \leftarrow$ neighbourhood radius
              $\hat{t} \leftarrow$ temporal stride
              $(h, l) \leftarrow$ high, low thresholds, respectively
       Result: $A \leftarrow$ equivalent action sequence
2      $f \leftarrow$ box-filter$(2k + 1)$
3      $\hat{\Theta} \leftarrow \{\hat{\theta}_t | \hat{\theta}_t = \theta_t - \theta_{t-\hat{t}}\}$ where $\theta_t \in \Theta$        ▷ Gradient
4      $\bar{\Theta} \leftarrow f \circledast \hat{\Theta}$        ▷ Smoothed gradient
5      $a_t \leftarrow 0 \; \forall_{t<\hat{t}}$
6      for $t$ from $\hat{t} + 1$ to length$(\Theta)$ do
7          if $a_{t-1} = 0$ then
8              if $\hat{\theta}_t \geq l$ and $\bar{\theta}_t \geq h$ then
9                  labelAndBacktrack$(t, 1)$
10             else if $\hat{\theta}_t \leq -l$ and $\bar{\theta}_t \leq -h$ then
11                 labelAndBacktrack$(t, -1)$
12             else
13                 $a_t \leftarrow 0$
14             end
15         else
16             resumeLabeling$(t, a_{t-1})$
17         end
18     end
19     return $A \leftarrow \{a_t | 0 \leq t \leq length(\Theta)\}$

20 Function labelAndBacktrack:
       Input: $i$, $label \leftarrow$ the given index and label
21     $a_i \leftarrow label$
22     $i' \leftarrow i - 1$
23     repeat
24         $a_{i'} \leftarrow label$
25         $i' \leftarrow i' - 1$
26     until $i' = \hat{t}$ or $label * \hat{\theta}_{i'} \leq l$;

27 Function resumeLabeling:
       Input: $i$, $label \leftarrow$ the given index and label
28     if $label * \hat{\theta}_i \geq l$ then
29         $a_i \leftarrow label$
30     else if $label * \hat{\theta}_i \leq -l$ and $label * \bar{\theta}_i \leq -h$ then
31         $a_i \leftarrow -label$        ▷ Mirroring for opposite direction
32     else
33         $a_i \leftarrow 0$
34     end
```

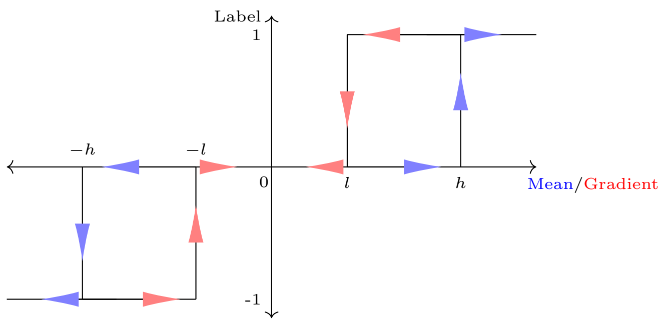


Figure 5.5: Action labeling with hysteresis thresholding

of the action at time step t_s, labelling the actions from t_s to t and then continue labelling the actions from t to t_e, where the action ends, when $\hat{\theta}_{t_e} < l$ as seen in Function *ResumeLabeling* on Line 27. For actions in the opposite directions, this is mirrored by using $-h$ and $-l$ as the high and low thresholds, respectively.

5.2.3 Evaluation of Action Discretization

To evaluate our approach, we compare it to two approaches used for similar tasks. One of them is an unsupervised approach used for action clustering [Krishnan et al., 2017], and the other is a supervised and an unsupervised learning approach used for human action recognition [Lee et al., 2017].

Comparison to Transition State Clustering

The industrial use case described here is similar to that of surgical robotics, where a surgeon teleoperates a robot arm. In order to offer appropriate assistance, [Krishnan et al., 2017] introduce Transition State Clustering (TSC), a clustering algorithm that clusters the velocity commands given by the surgeon to move the robot's end effector.

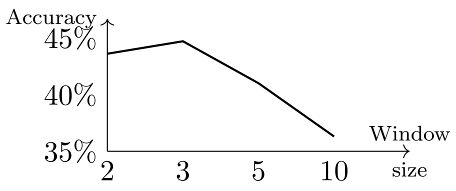
TSC takes as input a set of expert demonstrations $D = \{d_i\}$, where each d_i is a sequence of world states, in our case d_i is a sequence of end effector poses, but can also contain further sensory features that aid in the segmentation. They output a segmentation over the set of demonstrations and a transition state function that indicates the world states where a transition occurred between the segments.

In TSC, segmentation is modeled as a function $S : D \to (A)$ that assigns each state in the demonstration to a segment (in this case, an action $a \in A$). Additionally, they return the set of states where a transition occurs between the actions $\Gamma = \bigcup_i^N \{o_{i,t} \in d_i | T(d_i)_t = 1\}$, where $T : D \to (\{0,1\})$ is a transition indicator function. It is a mapping function over the states in a demonstration sequence, which returns 1 for each state, where a transition occurs and 0 otherwise.

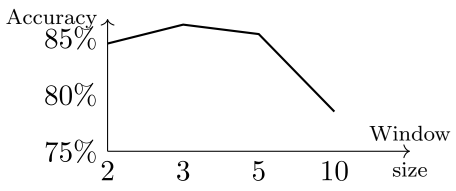
TSC is a segmentation function S which returns the transition states and the segments for each state in a given demonstration consistently across all demonstrations. They perform Dirichlet Process Gaussian Mixture Model (DPGMM) [Görür and Rasmussen, 2010] by sampling the number of segments m and their probabilities ϕ, from Dirichlet Process (DP) [Ferguson, 1973]. Then they sample from a categorical distribution to obtain the parameters of the normal distribution. This is applied with expectation maximization several times throughout the process to obtain a hierarchical segmentation of the demonstrations. At the end, they prune clusters that contain less samples than a given threshold, ρ to reduce the resulting clusters.

We applied this approach to our data using only the end effector poses, since our data did not contain any sensory information. We evaluate this using [Wu et al., 2015] similar to how TSC was evaluated in [Krishnan et al., 2017]. The segmentation accuracy is calculated as the true positives TP normalized by the number of data points n, $SA = \frac{|TP|}{n}$. Where the true positive segments are those that have over 40% overlap with the ground truth segments. We also evaluate this approach based on frame accuracy, which is the number of correctly assigned data points normalised over the number of data points.

We performed ablation studies by varying the hyper-parameters of TSC, the window size, and the pruning threshold, ρ. Figure 5.6 shows how the segmentation and frame accuracies are affected by the sliding window size. Figure 5.7 shows how they are affected by the segmentation threshold. As seen in the figures, the accuracies vary greatly in response to the two parameters. However, the accuracy, both frame and segmentation remain low, with the highest achieved frame accuracies around 90%.

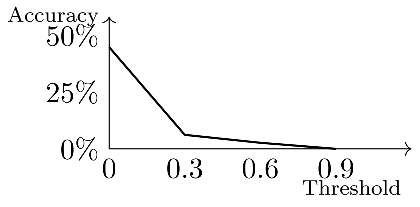


(a) Segmentation accuracy with different windows sizes.

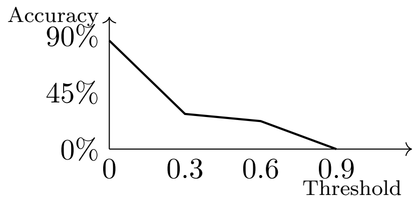


(b) Frame accuracy with different windows sizes.

Figure 5.6: The relation between the segmentation and frame accuracies and sliding window sizes and a pruning parameter of 0.0.



(a) Segmentation accuracy for different pruning values.



(b) Frame accuracy for different pruning thresholds.

Figure 5.7: The relation between the segmentation and frame accuracies and pruning thresholds for a window size of 3.

Our approach shows higher accuracy comparable to state of the art deep learning approaches. Additionally, using this clustering approach is able to group the actions, but does not tell us information about the clusters other than the actions it contains and when the transitions occurred. These pieces of information are provided by the hysteresis thresholding approach in addition to the labels, which can be used to add more knowledge that can be used in later stages of our framework. The next section compares our approach to one of state of the art supervised deep learning approaches.

Comparison to Temporal Sliding LSTM

We compare Algorithm 1's results to a deep learning approach for action recognition. In our work, we compare our approach to Temporal Sliding LSTM (TSLSTM) ensemble [Lee et al., 2017]. For the action recognition task, [Lee et al., 2017] uses skeleton configuration as input, their model is an ensemble of TSLSTMs, which recognize actions of short, medium, and long lengths and vote on the assigned class at the end. Each of the TSLSTMs modules contains several layers of Long Short Term Memory (LSTM) units that slide over the input, seen in Figure 5.8. The original work introducing TSLSTMs, [Lee et al., 2017] uses an ensemble of four TSLSTM networks, in our work, we used only three, since the pose did not add information in our use case. We also skip some of the pre-processing steps that fix scaling and orientation since our robot has a fixed base. We also replaced the Salient Motion Feature (SMF) extraction step with our own hysterisis thresholding to try and boost the results obtained from the network.

As proposed in [Ioffe and Szegedy, 2015], we normalize our attributes using Minmax normalization [Juszczak et al., 2002] to increase the model convergence speed. This is done using Equation 5.1, where $\psi_{i,t}$ is the i^{th} attribute of Θ_t. $min(\Theta_i)$ and $max(\Theta_i)$ are functions that output the minimum and maximum values respectively for attribute Θ_i over the entire dataset. Finally, we use one-hot encoding [Bishop, 2006] to label the actions in the 3 dimensions of Cartesian space.

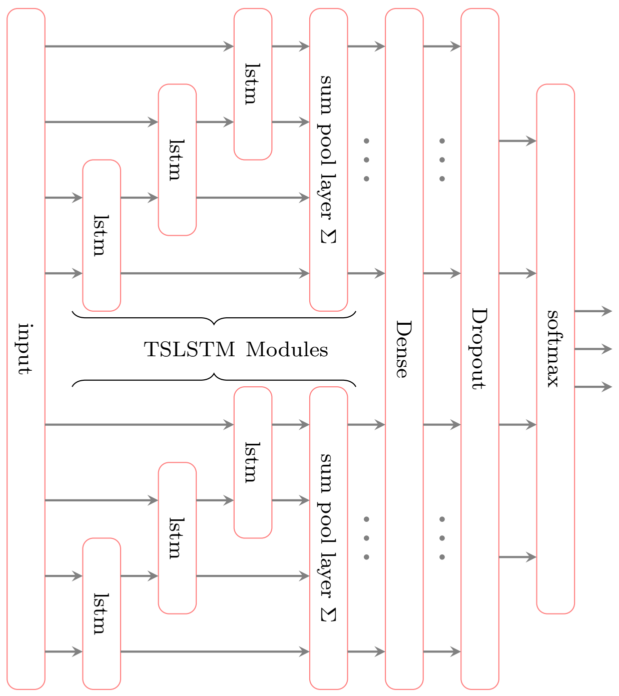


Figure 5.8: The architecture of a single TSLSTM network used in our evaluation.

$$\psi_{i,t} = \frac{\Theta_{i,t} - min(\Theta_i)}{max(\Theta_i) - min(\Theta_i)} \tag{5.1}$$

We further bridge the gap between skeleton-based human action recognition and teleoperated action recognition by handling each axis of motion independently. This is done by creating and training different ensembles of networks for the horizontal, vertical, and depth dimensions and only combine their results after the voting step. This prevents the *softmax* functions of the different axes from interacting, i.e., the normalization is done per axis.

To evaluate the models, we use several metrics for accuracy commonly used in recognition and classification problems in the literature [Godbole and Sarawagi, 2004, Zhang and Zhou, 2007]. The following are the used metrics, where T is the true labels, P is the predicted labels, n is the number of data points, and k is the number of labels. I is an indicator function, such that $I(x) = 1$ iff x holds.

- *Any Match:*

  Measures the percentage of correctly matching labels in the one-hot encoded outputs, as seen in Equation 5.2.

$$\mathcal{A} = \frac{1}{nk} \sum_{i=1}^{n} \sum_{j=1}^{k} I(T_{i,j} = P_{i,j}) \tag{5.2}$$

- *Exact Match:*

  Measures the percentage of correctly labeled data points, taking into consideration the multiple labels denoting motion in the different dimensions, as seen in Equation 5.3. This is normalized over the total number of instances.

$$\mathcal{E} = \frac{1}{n} \sum_{i=1}^{n} I(T_i = P_i) \tag{5.3}$$

- *Partial Match:*

  Counts the number of correctly predicted labels for each dimension independantly as seen in Equation 5.4.

$$\mathcal{P} = \frac{|T \cap P|}{|T \cup P|} \tag{5.4}$$

- *Hamming Loss:*

  Counts the misclassified instances. It considers both incorrect and missing classifications as seen in Equation 5.5.

$$\mathcal{L} = \frac{1}{kn} \sum_{i=1}^{n} \sum_{l=1}^{k} [I(l \in P_i \wedge l \notin T_i) + I(l \notin P_i \wedge l \in T_i)] \tag{5.5}$$

The evaluation results are summarized in Table 5.2. Hysterisis thresholding achieves better results seen in higher Any-, Exact-, and Partial-match ratios as well as lower hamming loss. We attribute this performance to the low input dimensionality, compared to the skeleton-based data commonly used for human actions, which are seen in datasets such as [Shahroudy et al., 2016, Li et al., 2010].

Table 5.2: Evaluations of the two approaches. $\mathcal{A}, \mathcal{E}, \mathcal{P}$ are the Any-, Exact-, and Partial-matching ratios respectively, and $\mathcal{L}$ is the Hamming-Loss.

Metric	$\mathcal{A}$	$\mathcal{E}$	$\mathcal{P}$	$\mathcal{L}$
TSLSTM	0.927	0.695	0.890	0.073
Hysteresis Thresholding	0.970	0.866	0.955	0.029

Another key advantage of the Hysterisis thresholding approach is the transparency compared the to deep learning approach. This is favorable in industrial domains as well as other safety-critical applications where the experts need to understand and change the model's behavior for different and more dynamic goals. Our approach has only 4 hyper-parameters that an expert can tune for different instances of the problem, the temporal stride $\hat{t}$, the box filter radius k, and the thresholds, l and h.

The next section shows the next module in the proposed framework, the action library learner. Given a sequence of actions and world state pairs, it is able to learn action models as well as behavior trees, which can be used to describe bigger more complex behaviors.

5.3 Learning A BT Library

This section describes the pipeline introduced in [Krömker, 2020] to learn commonly used action sequences, infer their PDDL models, and use them to construct Behavior Trees (BTs) that can be used to compose different behaviors. Starting from a trace of state action pairs, $T = (\langle s_0, a_0\rangle, \dots, \langle s_n, a_n\rangle)$, where $s_i = \langle f_0, .., f_{n_S}\rangle$ is a world state vector of length n_S, and $a_i \in A$, which is the set of atomic actions an agent can use to change the world state from s_i to s_{i+1} by changing the value(s) of at least one feature of s_i, this module performs the following four steps:

1. Extraction of the set of common action patterns
   $\mathcal{P} = \{(a_{0,1}, .., a_{0,n}), .., (a_{m,1}, .., a_{m,n'})\}$.
2. Finding and refactoring overlapping and recurring patterns
3. PDDL Model inference for each pattern $p \in \mathcal{P}$, $pre(p)$ and $post(p)$, where $pre(p)$ and $post(p)$ are the sets of pre- and post-conditions of p respectively.
4. Creating BTs of the common patterns

5.3.1 Common Action Pattern Extraction

In order to find common patterns, we need to compare the different action sequences to see where they overlap. We also keep in mind that the action sequences can be of different lengths. We use the Smith-Waterman algorithm [Smith et al., 1981], which is used to extract patterns in DNA sequencing. It finds local alignments between two sequences, $a = (a_1, .., a_{n_a})$ and $b = (b_1, .., b_{n_b})$ of arbitrary lengths, n_a and n_b respectively. The algorithm keeps a similarity matrix H, whose elements are calculated using Equation 5.9, the maximum value of Equations 5.6, 5.7, and 5.8, and 0. These calculate the similarity score for an alignment step, and gap step in the horizontal and vertical dimensions of the matrix respectively. The first row and column are initialised with 0, i.e., $H(i, 0) = H(0, j) = 0$ for $0 \leq i \leq n_a$ and $0 \leq j \leq n_b$. The rest of the matrix is filled using a heuristic similarity function $s : A \times A \to \mathbb{Z}$ and a gap penalty G as seen in Equations 5.6 - 5.8. s is chosen such that $s(a, b) > 0$ iff $a = b$ and $s(a, b) < 0$ iff $a \neq b$.

$$\alpha_{i,j} = H_{i-1,j-1} + s(a_i, b_j) \tag{5.6}$$

$$\gamma_{i,j} = H_{i-1,j} - G \tag{5.7}$$

$$\gamma'_{i,j} = H_{i,j-1} - G \tag{5.8}$$

$$H_{i,j} = max(\alpha_{i,j}, \gamma_{i,j}, \gamma'_{i,j}, 0) \tag{5.9}$$

After the matrix is filled, the algorithm tracks back from the global maximum to find the path of the highest similarity score till it reaches a cell with a score of 0. The value of G controls whether the algorithm prefers to include a gap step over an alignment step. The Smith-waterman algorithm runs in $\mathcal{O}(n_a * n_b)$ and its space complexity can be reduced to $\mathcal{O}(min(n_a, n_b))$ as shown by [Myers and Miller, 1988].

To build a library of most common action patterns, we needed to modify the Smith-Waterman algorithm to return a set of possible alignments instead of only the alignment with the maximum score. This modification was done in [Krömker, 2020] and is illustrated in Algorithm 2. We keep a list of the local maxima, instead of the global maximum and repeat the backtracking step for each of them. After that, we post process these patterns by adding references to other patterns as well as create Maximally Specific Trees (MSTs) [Robertson and Watson, 2015] as discussed in the next section.

Algorithm 2: The multi-alginment smith-waterman algorithm introduced in [Krömker, 2020].

```
1 Function MultiAlignmentSW:
      Input: $A$ $B$ Action Sequences
      Result: $aignments \leftarrow$ set of alignments
2     $ScoringMatrix \leftarrow$ fillScoringMatrix$(A, B)$        ▷ Equation 5.9
3     $localMaxima \leftarrow$ getLocalMaxima$(scoringMatrix)$
4     $alignments \leftarrow \phi$
5     for $m \in localMaxima$ do
6         $traces \leftarrow$ traceback$(m, scoringMatrix)$
7         $aignments$.insert$(traces)$
8     end
9     return $alignments$
```

Pattern References and MSTs

After extracting the common patterns, we measure the frequency of occurrences of these patterns. This is used to determine their reliability, i.e., more frequent patterns are less likely to contain noisy *sub-patterns*. The most common of these are checked for overlap. For any two patterns, p and p', such that $p' \subset p$, we change $p' \cap p$ in p with a reference to p'. For example, checking the two patterns, $p = (a_1, a_2, a_3, a_4)$ and $p' = (a_3, a_4)$, transforms p into (a_1, a_2, p').

These patterns can then be transformed into MSTs as shown in Figure 5.9. We can further refine the trees by using a decorator node for repeating actions, e.g., $q' = (a_5, a_5, a_5)$ can be transformed into the MST in Figure 5.10.

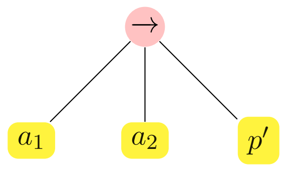


Figure 5.9: A BT created using overlapping patterns, where one is a subset of the other.

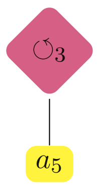


Figure 5.10: Creating a BT from a recurring pattern using a repeat decorator node.

The MST is built from a given pattern by using a sequence root node and adding all actions in the pattern as children of this node directly. Such tree over fits the given pattern, which is desirable for our intended purpose, especially in safety critical tasks. In other applications, these MSTs can be pruned, abstracted, or can be connected as children of selector nodes that choose randomly among their children [Robertson and Watson, 2015]. In our use case, we want to allow robot operators and floor supervisors to define behaviors using a library of MSTs. To ensure safety of these trees, the library must include the pre- and post-conditions of the learned patterns.

5.3.2 Pattern PDDL Model Inference

PDDL action models are commonly used in planning problems to define pre- and post-conditions of actions, *pre* and *post* respectively [McDermott et al., 1998]. These action models have also been used to construct and learn BTs, which are safe by construction [Colledanchise et al., 2019, Cai et al., 2021]. In our setting, we learn these action models in order to create safe atomic BTs, that can be directly integrated as branches to compose complex, but safe, behaviors. Safety comes from the reactivity of the trees, which would be able to check the preconditions of each action before it is ticked. We first go over the basics of PDDL action models before explaining how we learn them for the different patterns in our BTs.

Planning Domain Definition Language

PDDL was introduced by [McDermott et al., 1998] to standardize action models inspired by the STRIPS planner [Fikes and Nilsson, 1971] and its accompanying language. They are the base for most planning languages used today.

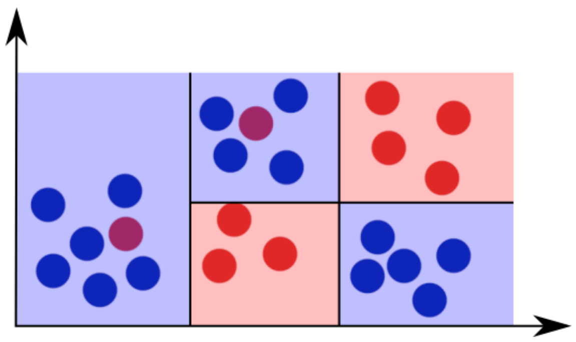
PDDL is used to define the domain dynamics [McDermott et al., 1998], how actions change the world and when they are feasible. It is traditionally split into a domain definition and a planning problem. Domain definitions describe the domain including the available predicates, actions, types, and constants, among other things. For our purposes, we use domain definitions similar to STRIPS, where a problem instance is represented as a tuple $\langle P, A, s_0, G \rangle$, where P is a set of predicates, A is the set of operators (actions), s_0 is the initial world state, and G is a set of goal conditions.

Each action $a \in A$ is modeled using 3 sets pre_a, add_a, del_a preconditions, add-effects, and delete-effects, respectively [2]. For example, an action to install a part p into a target location t in an assembly scenario would be defined as seen in Listing 5.1. The preconditions for this action is that p is *held* by the robot and t is *clear*. After the execution of the action, p is no longer held, and t is no longer clear. Additionally, p is now *installed* and is located at t.

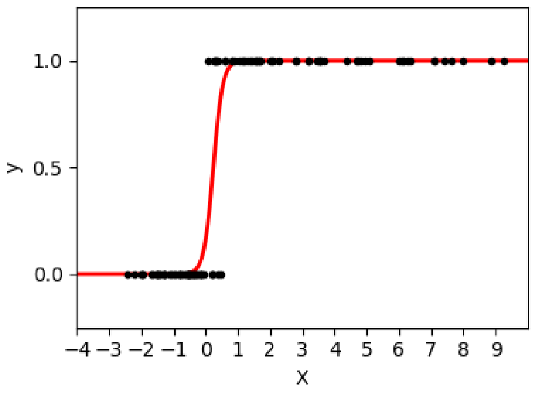
```

    (:action install
        :parameters (
            ?p - part ;the part to be installed
            ?t - location ;target location to install the part
        )
        :precondition (and
            (held ?p)
            (clear ?t)
        )
        :effect (and
            (not (held ?p))
            (not (clear ?t))
            (installed ?p)
            (at ?p ?t)
        )
    )
```

Listing 5.1: PDDL install operator.

[2] *add* and *del* set have these respective names because STRIPS represents a state, s as a set of atomic propositions. Executing an action a *adds* its add effects add_a to s and *deletes* its delete effects, del_a from s.



(a) Hard boundaries learned by decision trees.



(b) Soft decision boundaries of logistic regression.

Figure 5.11: Decision boundaries of decision trees (Figure 5.11a) and logistic regression (Figure 5.11b) used to learn action preconditions.

Action Preconditions

We represent an action a's preconditions as a binary classifier pre_a over the state description vector, such that $\text{pre}_a : S \rightarrow \{-1, 1\}$. We train the classifier given a set of state-action pairs $T = \{\langle s_i, a_i \rangle\}$. For the evaluation, we train two models, Decision Trees and Logistic regression. These two models were chosen for their separation boundaries, which are perpendicular to the axis, which fits our domain needs. Decision trees offer hard decision boundaries, while those of logistic regression are smoother as shown in Figure 5.11.

The precondition classifier uses the state vector, s_i as a feature vector. The classifier for action a, pre_a labels the point with 1 iff a is feasible in s_i, and -1 otherwise. To preprocess the data for each classifier pre_a, we iterate over the dataset, labelling each data point $\langle s_i, a_i \rangle$ with 1 iff $a_i = a$ and -1 otherwise. However, given the nature of the task at hand, multiple actions could be feasible at any given time and the operator chooses only one to perform. This causes a bias in our data because some labels are not as reliable as others. All positively labelled data points can be considered true positives, but negatively labeled ones might have been incorrectly labelled.

To solve this bias problem, we follow [Lee and Liu, 2003], who address it using weighted logistic regression. They use logistic regression and learn the conditional probability of the positive labels. For a classifier over a data set with positive label $Y = 1$ and negative label $Y = -1$, they show that the conditional probability of true positives is greater than 0.5, $P(f(x) = 1|Y' = 1) > 0.5$, and that the conditional probability of false positives is less than 0.5, $P(f(x) = 1|Y' = -1) < 0.5$ allowing us to threshold at 0.5 for the classification assuming that the classification function is powerful enough to represent these probabilities [Lee and Liu, 2003].

Action Effects

In some settings, the effects of durative actions occur over time and could non-linearly affect different state attributes. For example, in the game StarCraft II [3] described in detail in Section 5.3.3 , a player is able to give commands to erect a new building or train new units. To erect a building b, first the required resources r_b are deducted from the player instantly. Later on, after time t_b, the building b is added to a location in the player's territory. Some buildings can also affect the player's capabilities, e.g., building a *Refinery* enables the player to gather gas, other buildings add actions to train certain units or research upgrades.

We model such effects as a series of function sets $\Theta_t^{(a)}$ over a number of time steps. The postconditions for action a at time step t are $\Theta_t^{(a)} = (\theta_0, \theta_1, \ldots, \theta_n)$ where θ_i describes how a updates the i^{th} element of the state vector.

[Krömker, 2020] used two types of these predictors, LSTM networks and linear regression. The LSTM models are more powerful, but the linear regression models are more transparent and are faster to train and query.

Linear Regression Models

We assume that the effects of each action on each state attribute can be approximated by these predictors for each time step after the action execution. The number and types of predictors are domain specific, i.e., we need $|S| * t_{max}$ predictors for each action, where t_{max} is the largest expected delay for an action's effect. t_{max} is a domain specific hyper-parameter, if we choose t_{max} to be too high, we end up with too many predictors which is computationally expensive. If t_{max} is too low, then we might not have enough predictors to cover the long term effects.

We train several predictors, one for the effect at each time step. This can be seen in Equations 5.10-5.12. Where $y_{i,t,k}$ is the effect on the i^{th} attribute of the state vector $v_{t-1,i}$ is the state vector at time step $t-1$.

[3] starcraft2.blizzard.com

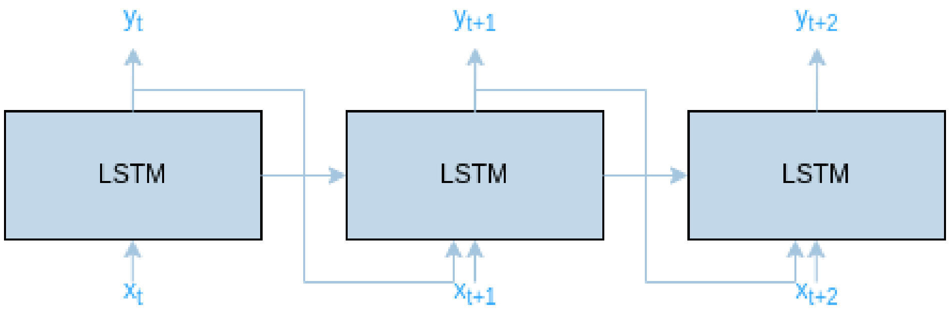


Figure 5.12: Layout of the LSTM network for predicting action effects.

$$y_{i,t} = X_{i,t-1} W_{i,t} \tag{5.10}$$

$$y_{i,t} = \begin{pmatrix} y_{i,t,1} \\ y_{i,t,2} \\ \vdots \\ y_{i,t,n} \end{pmatrix} \tag{5.11}$$

$$X_{i,t} = \begin{pmatrix} 1 & v_{t-1,1}^T \\ 1 & v_{t-1,2}^T \\ \vdots & \vdots \\ 1 & v_{t-1,n}^T \end{pmatrix} \tag{5.12}$$

LSTM Networks

Since LSTM networks are able to reuse information and use their own outputs as feedback, they are capable of predicting the outputs (action effects) at multiple time steps at a time. As seen in Figure 5.12, the output at each time step, y_t, is also used as input to the next along with the corresponding state vector. Unlike the linear regression model, here we need to train fewer LSTM models for the prediction. Using an optimization algorithm such as Stochastic Gradient Descent (SGD), we are able to update the weights of the cell's gate based on the error from the previous example in order to minimize the error (distance between output and prediction).

5.3.3 Evaluation

This section shows the evaluation for common pattern extraction and pattern model learning. We first describe the dataset used for the evaluation, after that we go over the experiments, results, and discussion. Due to the lack of data from industrial use cases for this task, we choose to build a proof of concept using data from the game: Starcraft II.

Data from Industrial Use-Cases

The available data within the Cluster of Excellence (CoE) coming from the different university institutes, is usually abundant but suffers three major issues that preclude their use in this task:

- **Isolated Machine Data:**

  The collected data usually focuses on a single machine used in isolation and does not connect the data points to data from other machines to give us an overview of the process.

- **Acquisition Costs:**

  Each data point can be very expensive to collect because it costs time, energy, and consumable materials that will be processed by the machine.

- **Experimental Data:**

  Data is collected in isolated experiments, and is therefore quite noisy compared to production data collected from live production plants.

Larger more reliable datasets could be collected at high cost, but this requires an infrastructure that allows not only the storage and transfer of the data, but also the connects the data points collected at different institutes but belong to the same process. For example, the process of metal forming often include a casting step, that starts with molten metal to produce metal ingots. These ingots can go through several other processes such as rolling and forging. In the academic landscape, each one of these steps is performed at a different institute by different entities making large datasets hard to collect. To avoid the overhead of data acquisition, we built a proof of concept using the MSC dataset [Wu et al., 2017] described below.

StarCraft II Data

StarCraft II is a military science fiction real-time strategy zero-sum game with partial observability. Due to its complex settings in addition to the success of its predecessor StarCraft I, it attracted researchers from different fields over the past decade. With a large player count and an Application Programming Interface (API) for data collection and game control, it became a benchmark for game Artificial Intelligence (AI) agents [Čertický et al., 2018].

The game features three factions, Terran, Zerg, and Protoss, that have their respective strengths and weaknesses, each is controlled by a player engaging in battle against other players. Players compete for resources distributed over the map. They use these resources to build structures and train units, which can build buildings, fight battles, or gather resources. A player is able to micro manage the units, moving and commanding units individually or in groups throughout the course of the game. However, like many strategy games, players also need to think of the *macro-management* aspects, which include the high level strategy controlling the choice of build orders, resource allocation, as well as the balance between military and economic development.

To build a proof of concept of our framework, we use the MSC [4] [Wu et al., 2017], a dataset for macro-management tasks in StarCraft II. It has over $36,000$ high quality game replays, each is a list of around $10,000$ frames. The replays are sampled at a fixed rate of 8 frames per time step to make sure they contain at most 1 macro action per frame. It contains only games between skilled players, those with Actions per Minute (APM) over 10 and Match Making Rating (MMR) over $1,000$. Both are measurable attributes of players, which are directly proportional to a player's skill.

Each frame is presented as a pair of action and game state observation vector $\langle a, s \rangle$, which contains the frame id, player information, alerts, upgrades, and unit and building information. MSC also offers spatial information, which was not used in our work. The next section discusses the similarity between actions in the StarCraft II domain and industrial domains. After that we detail how we used MSC to train and evaluate our models.

Action Similarity Across Domains

In industrial as well as non-industrial domains, some actions have lingering and long term effects on the world state. For example, in the open-die forging, once the metal ingot comes out of the furnace, its temperature will keep dropping gradually till it reaches room temperature, we call this the lingering effect of the action. Other actions have more traditional effects, which cause an instant and abrupt change to the world state. For example, the action of striking the metal ingot by the dies will compress it changing its geometry.

Such actions also exist in non-industrial domains. StarCraft II is a good example as it contains actions that have effects of both types. For example, the action of mining resources has the lingering effect of steadily increasing a player's resources throughout the game, while the action of training a new unit has the abrupt effect of decreasing resources (the cost of training the unit) and increasing that unit's count at a later point after it had been trained. The diversity of these actions enabled us to use StarCraft II and the MSC dataset as a proof of concept for our models. We model the lingering action effect as an *idle* pattern, which has no preconditions and is feasible at any given time. We only use linear regression to learn these effects. We train it by selecting state sequences that do not contain any actions and calculate the average change over time. In the StarCraft II

[4] github.com/wuhuikai/MSC

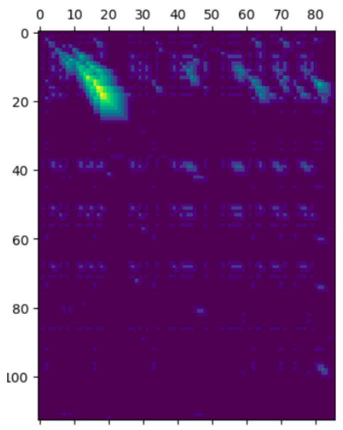
domain, this pattern only affects two state features describing a player's resources, minerals and gas, so we only train models for these attributes. However, this is domain specific, therefore the respective models' complexity and types can be chosen or eliminated based on domain knowledge. The next section goes over the training and evaluation processes.

5.3.4 Learning a BT Library from the MSC Dataset

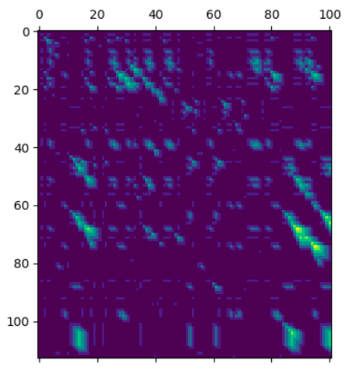
We filter games that contain the *protoss* race as it has the lowest number of features across the races, 316 features and 60 macro-management actions. This reduces the computational time and data requirements. We additionally split the data into training and test set of 83% and 17% respectively. We evaluate our approach using *TensorFlow* [Abadi et al., 2015] for the LSTM networks and *Scikit-learn* [Pedregosa et al., 2011] for logistic regression, linear regression, and decision trees.

Action Patterns and MSTs

Action alignment and similarity matrix compare two action sequences at a time. Figure 5.13 shows the scoring matrix between different action sequences from different games. As seen in Figure 5.13a, the top left corner shows brighter colors indicating that players have similar techniques for starting the game unlike Figure 5.13b. Figure 5.13b shows that players have different opening strategies, demonstrated by the darker areas on the top left corner. However, bright regions can be seen in different areas during the later game stages, indicating common actions.



(a) Action sequence alignments with high similarity in starting pattern.



(b) Action sequences showing different opening patterns but more similarity at later game stages.

Figure 5.13: Visualization of similarity matrix and alignment of action sequence. Brightness indicates higher similarity. Horizontal and vertical axes indicate the time steps, with one action per time step.

From the learned alignments and common patterns, we show some of the generated MSTs. Figure 5.14 shows the most common action sequence pattern. Its high frequency is caused by its low depth and simplicity. In this pattern, the players are building basic units, *Probes*, which are used to gather resources and build buildings. This is important as a starting pattern as well as in other stages of the game. Figure 5.15 shows the MST generated from a common starting pattern. We can see the MST from Figure 5.14 on the bottom right subtree. That pattern comes after players have built basic buildings needed to gather resources.

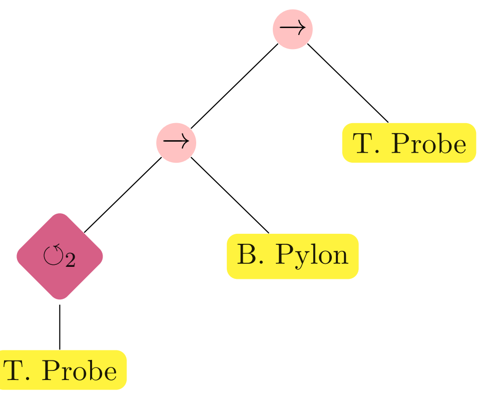


Figure 5.14: An MST generated from the most common action pattern.

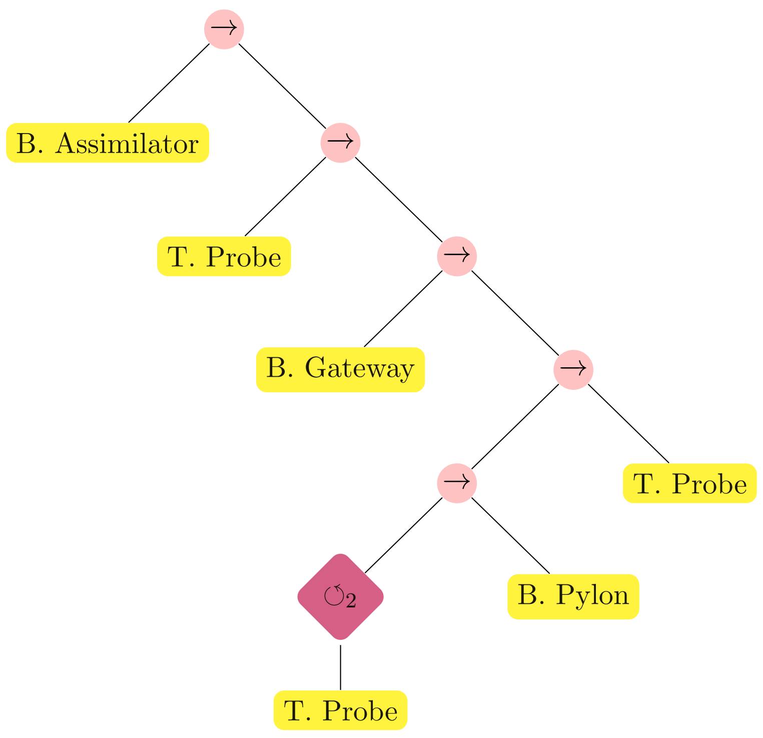


Figure 5.15: The generated MST for a commonly used starting action sequence.

Learning Action Models

This section discusses the evaluation metrics we use for the different models. We use the balanced accuracy, $\bar{\mathcal{A}}$ introduced in [Mower, 2005] to compare the models. $\bar{\mathcal{A}}$ is calculated as the average true positive and true negative labels, as seen in Equation 5.13, where TPR and TNR are the true positive and negative ratios shown in Equations 5.14 and 5.15, respectively.

As seen in Figure 5.17, decision trees had an overall better performance than logistic regression. This may be attributed to the orthogonal class separation offered by decision trees, which aligns with the actual class separation for the action preconditions. Action preconditions in the StarCraft domain are usually functions over the resource features for each player. The balanced accuracy score averaged over all actions for decision trees is 61%, while logistic regression scored 52%. Additionally, two actions, which are the most commonly used ones, were classified by the decision trees with scores 96.5% and 99.5%. In addition to the accuracy, decision trees are preferable to logistic regression for this use case due to their human readability and their significantly shorter training time. Figure 5.16 shows a sample decision tree for the preconditions of the action *Build Stargate*. Each node contains its respective condition and attribute, the gini index, and the positive and negative sample sizes. While the inference is reasonable for the two features, *gas* and *minerals*, it does not include the

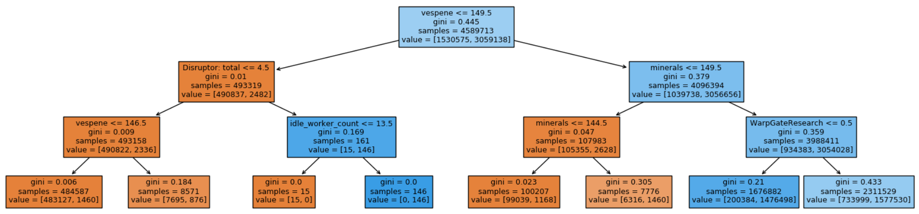


Figure 5.16: Example decision tree for the action *Build Stargate* preconditions. The true preconditions for this action is $Minerals >= 150$, $gas >= 150$ and $Cyberneticscore >= 1$. This shows that the classification is reasonable, but imperfect as the $Cyberneticscore >= 1$.

attribute *Cybernetics Core*, which can be attributed to the lack of negative examples involving that attribute.

$$\bar{\mathcal{A}} = \frac{\mathrm{TPR} + \mathrm{TNR}}{2} \tag{5.13}$$

$$\mathrm{TPR} = \frac{\mathrm{TruePositives}}{\mathrm{Positives}} \tag{5.14}$$

$$\mathrm{TNR} = \frac{\mathrm{TrueNegatives}}{\mathrm{Negaives}} \tag{5.15}$$

To Evaluate the models for action effects, we use the R^2 score, which is calculated as seen in Equation 5.16. It uses the residual sum of squared errors, SS_{res} and the total sum of squares seen in Equations 5.17 and 5.18, respectively, where y_i and f_i are the real labels and the predicted values, respectively and $\bar{y}$ is the average of the predicted values. The R^2 score measures how well the model fits the given data relative to a model that always predicts the mean, $\bar{y}$. The average R^2 scores, averaged over 20 time steps for the linear regression models are plotted in Figure 5.19.

Except for some outliers, the effect models of most actions have an R^2 score of approximately 0.7. The actions, whose models have lower scores are those occurring at very low frequencies, between 600 and 1300, compared to the average frequency of over $30,000$ occurrences. We can see the highest R^2 score, 0.94, was achieved by the models for the *idle* action, which has the highest occurrence frequency.

$$R^2 = 1 - \frac{SS_{\mathrm{res}}}{SS_{\mathrm{tot}}} \tag{5.16}$$

$$SS_{\mathrm{res}} = \sum_i (y_i - f_i)^2 \tag{5.17}$$

$$SS_{\mathrm{tot}} = \sum_i (y_i - \bar{y})^2 \tag{5.18}$$

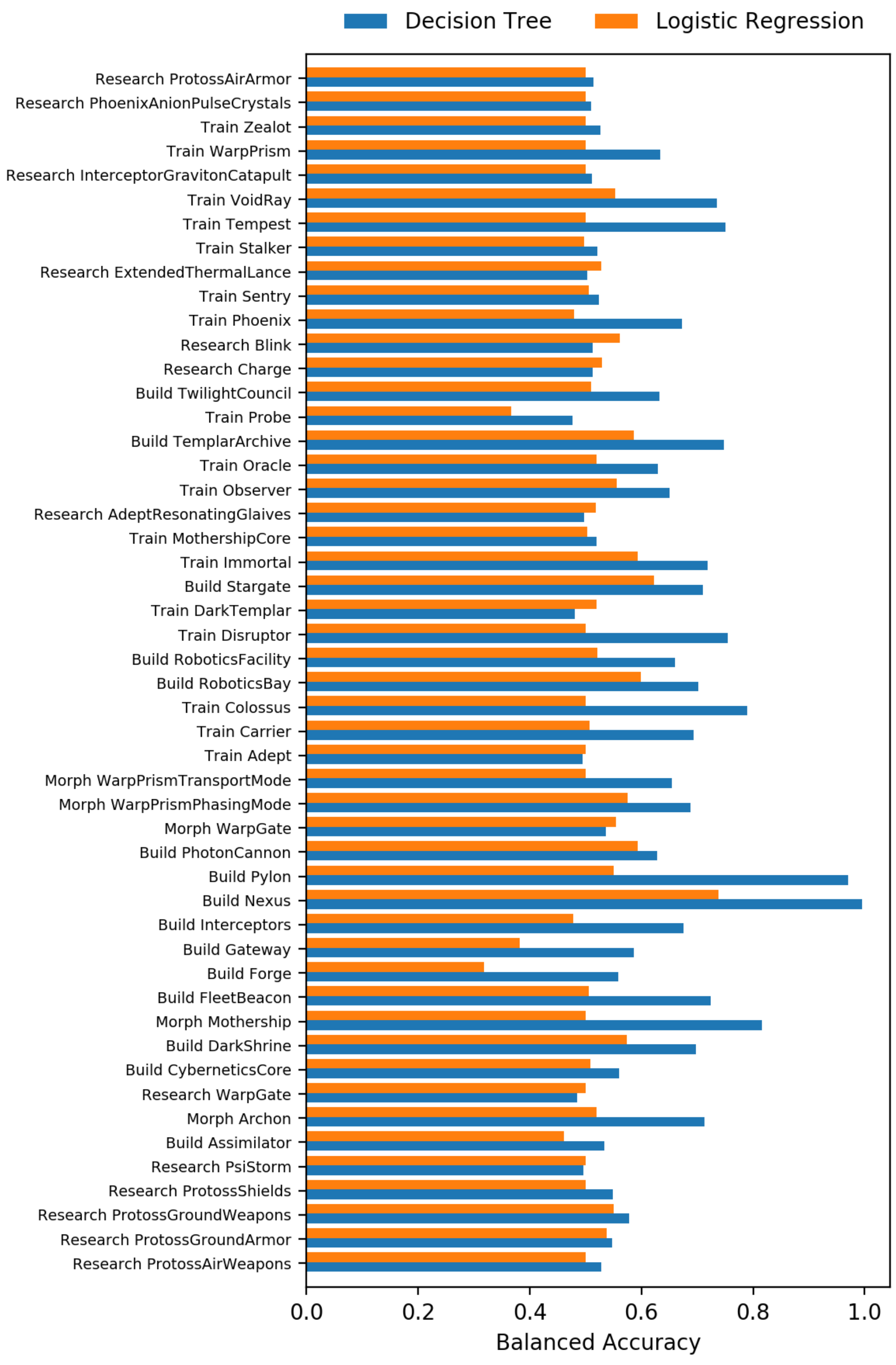


Figure 5.17: Comparing decision trees to logistic regression

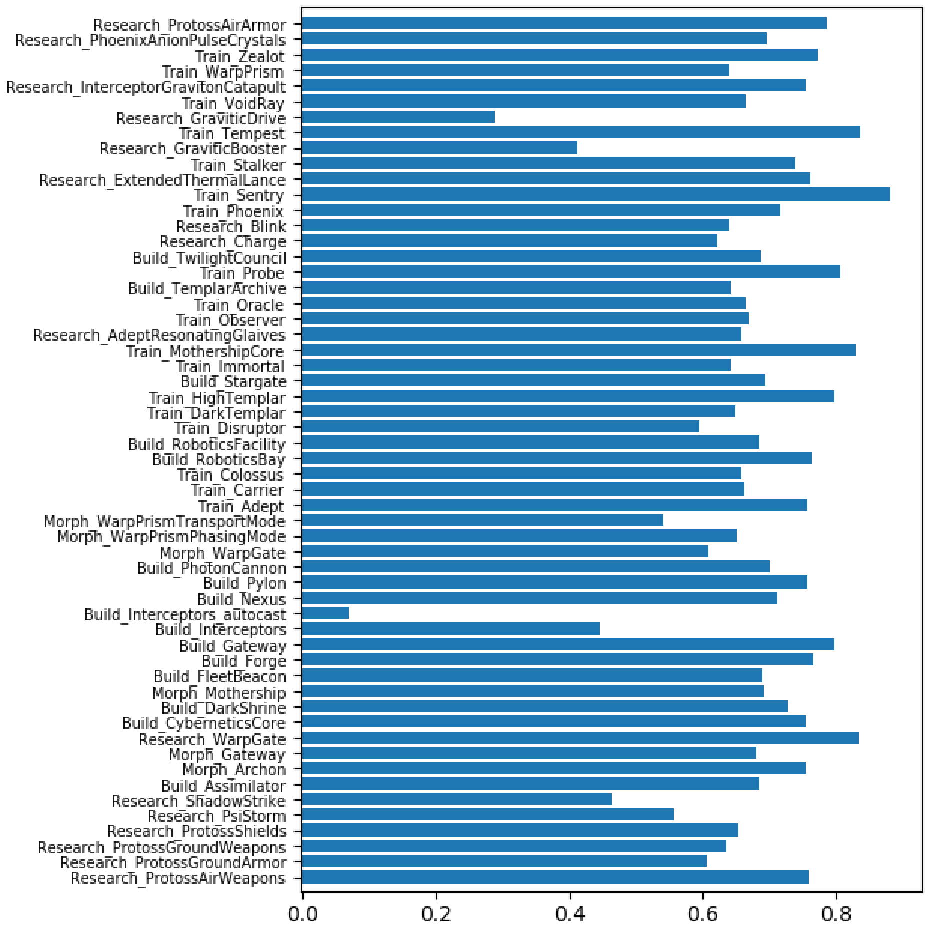


Figure 5.18: R^2 scores using logistic regression to learn the action effects

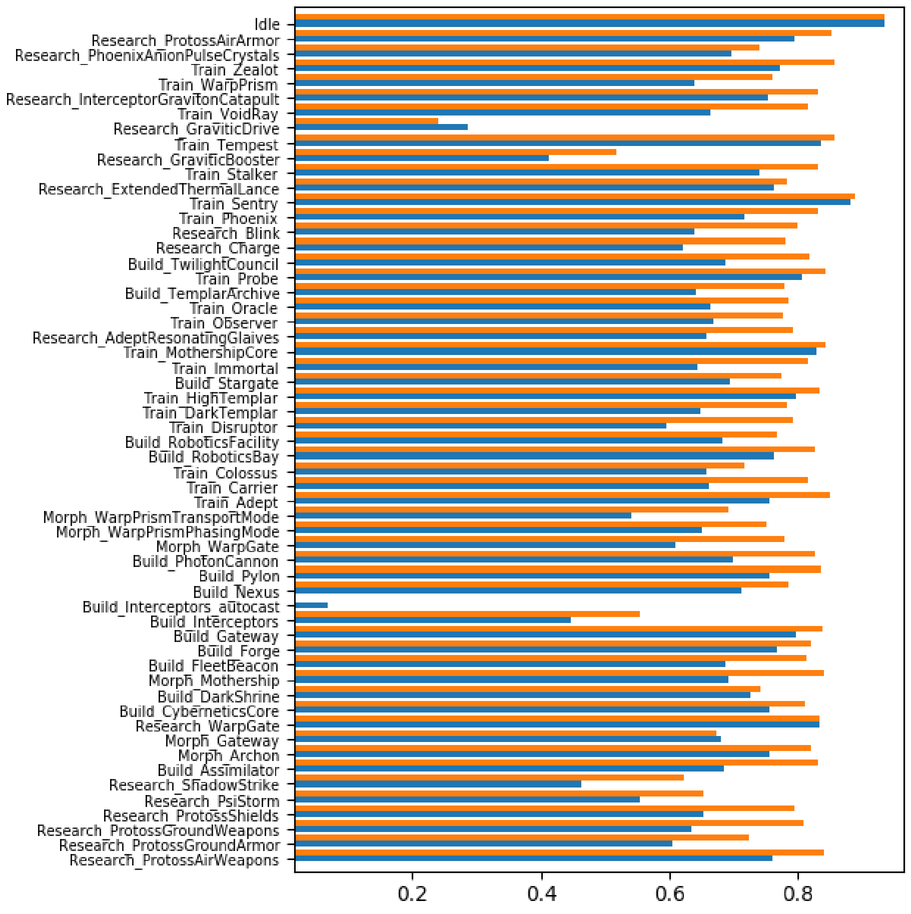


Figure 5.19: Average (blue) and median (orange) R^2 scores for predicting action effects for 20 time steps using Linear Regression Models

On the other hand, the LSTM-based models showed an average R^2 score of 0.9 after being trained for only 100 epochs. As seen in Figure 5.20, most actions have high R^2 scores. This shows that LSTM networks can represent such complex effects quite well, however they are undesirable in industrial situations due to their opacity. They also require a long training time compared to the other transparent models.

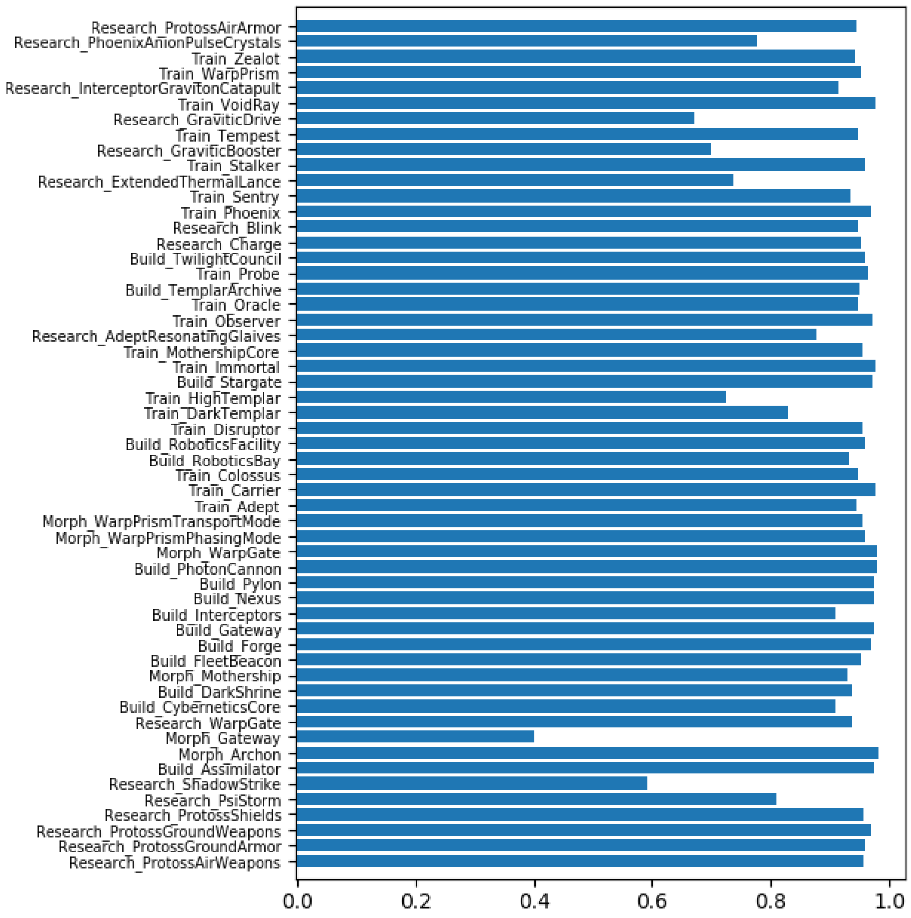


Figure 5.20: The R^2 scores using LSTM networks for predicting action effects.

The work in this module provides a library of atomic BTs, which can be used to compose complex behaviors. The safety issues associated with deep neural networks and LSTM networks are mitigated by the human readability of the generated trees. Involving a human operator on several levels of the framework increases the safety and reliability of the system. Humans are able to read the trees, including the preconditions and effects of each action. They can overwrite or remove these trees if they compromise the overall safety of the system. That being said, the safety of the generated trees could still be increased by switching the LSTM effect models for more transparent models such as Fast Fourier Transform (FFT), however, this is left for future work.

5.4 Discussion

This chapter proposed a framework that extracts and uses Digital Shadows (DSs) for industrial robot teleoperation tasks. Section 5.1 gives an overview of the proposed framework, starting with the data collection, where we collect and store the data resulting from manual teleoperation data. Section 5.2 describes how we descretize this data, starting from the operator's velocity commands used to move the robot's end effector in the Cartesian space into state action pairs. The next part of our framework, described in Section 5.3 extracts the most common state action pairs then transforms those into BTs. The resulting BTs are models of common operator behaviors, which can be used to design and compose different and more complex behaviors. Additionally, they act as a decision support system for the designer because they give insight into the common behaviors. The insight is given in terms of action models, which can reveal information such as preconditions and effects of different actions. They also a measure of the reliability of each of these behaviors by giving information about the sample sizes used to generate their models.

As illustrated in Figure 5.1, the proposed framework has five main steps. In the last step, an operator can use a designed behavior, composed of learned and programmed behavior to create a BT to teleoperate the robot while the framework assists them. This assistance can either take the form of conformance checking as seen in process mining [Zhao and Han, 2022] or the form of shared autonomy control we discuss in Chapter 6. The implementation and empirical evaluation of both options is left for future work. Other modules mentioned in this chapter are evaluated empirically and compared to different state of the art approaches to achieve their respective goals.

Since we aim to apply this in industrial scenarios, we evaluate the approaches qualitatively and quantitatively. The latter involves the execution speed affecting the throughput of the system, e.g., processing speed, while the former include safety, usability, and trust between the operator and the robot. These are indirectly affected by the processing speed through the reactivity of the robot. Nevertheless, they are directly affected by the transparency of the chosen approach [Dammers et al., 2022, Baier et al., 2022, Trinh et al., 2023c].

To this end, we focus on the so-called transparent and explainable AI approaches and compare the performance of our approach to transparent as well as black box approaches. Our hysteresis thresholding approach to action descritization achieved better performance than both alternatives for the action recognition, the LSTM network and TSC. In addition to the higher accuracy, our approach's transparency allows it to be used in more safety critical tasks. It also does not require training time or data, which is usually scarce in industrial applications such as metal forming due to the cost of executing such a process. Additionally, many companies and production plants are reluctant to share their data, which gives them an edge over their competitors, leading to more data scarcity in many industrial domains. Without data-sharing, an industrial partner has to pay the high costs of generating more heterogeneous data or settle for overfit models learned by black box data-driven approaches.

This is particularly relevant to the step of learning a BT library in our framework. This step is divided into multiple tasks, we compare several approaches for each. Some of the approaches are black box approaches showing higher accuracy in addition to faster training and prediction times than their alternatives. However, our framework is a human-*in*- and -*on*-the-loop framework allowing the human to design as well as teleoperate the robot to mitigate and overwrite undesired parts of the learned behaviors. Additionally, we rely on using BTs, which are highly modular. This means that a human designer can isolate and eliminate branches of the tree which are unreliable or untrustworthy. Our framework additionally empowers the behavior designer by giving them information regarding the reliability of different action models. Such information can also be overwritten in real time by the machine operators to ensure the safety of the overall process.

Relying on such human in the loop approaches not only solves the technical challenges, but also proposes solutions to ethical challenges and concerns, namely replacing the human workers with intelligent robots. Our approach empowers the blue-collar workers with more insight into the different aspects of the robot operation and maintains their autonomy during the execution of the process. The next chapter shows several extensions to BTs, which allow them to cover more industrial use cases. Some of the covered use cases allow more collaboration and coordination between the robots and human coworkers.

Chapter 6

Behavior Tree Extensions for Collaboration

Human-Robot Interaction (HRI) covers a wide spectrum of interactions between human and robot coworkers. This spectrum has been studied and surveyed in different contexts [Baier et al., 2022, Goodrich et al., 2013, Otto and Zunke, 2015] offering different classifications of use cases. This classification allows us to focus on different aspects given the class of interaction. For example, if the robot is sharing the workspace with a human, we would need to keep track of where the human is within the workspace and control the speed of the robot accordingly [Trinh et al., 2023b]. In other cases, where the robot is inside a cell, we only need to ensure that no human is inside the cell during execution.

This chapter focuses on two classes of interaction between a robot and a human coworker. Namely human *on* the loop and human *in* the loop. In Section 6.2, we present our Behavior Tree (BT) extension published in [Behery et al., 2023f], enabling Mixed Initiative Planning (MIP) for BTs, where a human supervisor defines an abstract representation of the robot's task, and the robot can modify the tree at run time to increase performance.

Section 6.3 reports on another extension published in [Behery et al., 2021, Behery et al., 2023c] allowing BTs to handle tasks where humans and robots work side by side occasionally passing a work piece between them.

A third extension is discussed in Section 6.4, which introduces PHAse Switching Teleoperation Behavior Trees (PHAST BTs) to represent robot teleoperation tasks without losing modularity and with little impact on the mental effort required to define behaviors using BTs and increasing the modularity and usability of defining such tasks using alternative methods such as Finite State Machines (FSMs). This extension was proposed in [Behery et al., 2023e], this thesis additionally adds a formal proof that PHAST BTs have the same representational power of the state of the art representation for these tasks, which rely on FSM. This chapter presents the details of the proposed extensions as well as case studies emphasising their contribution and impact on BT-based behavior models in industrial use cases.

6.1 Mixed Initiative Planning

MIP [Ferguson et al., 1996] allows the human supervisor to define abstract plans or behaviors, allowing the robot(s) to complete the plans at run time. The robot, or other autonomous agent, can ground variables, reorder action sequences, or add and remove actions to/from the plan. Using MIP is beneficial in situations where human knowledge needs to be combined with robot flexibility. Compared to hard-coded or hand-engineered robot behavior, this planning paradigm allows using the robot's perception to ground variables and make informed decisions at run time based on concrete information such as object locations, intentions of other agents, etc.

This has an advantage over fully autonomous systems, especially those that involve learning, in terms of transparency and flexibility. Transparency allows human coworkers to know the robot's intentions at all times, which increase the trust [Dammers et al., 2022]. On the other hand, these robots are more flexible than fully autonomous systems since they do not require retraining, a human can simply reprogram them to perform a new task or comply with new requirements [Cai et al., 2021].

Several approaches exist for MIPs and Variable Autonomy (VA) in robotics dating back to the 1990s. Some of these approaches allow a human operator to define the plan allowing the robot to ground variables at run time [Ferguson et al., 1996, Methnani et al., 2021, Schmaus et al., 2020]. Others allow the supervisor to use an interface to define action affordances [Hart et al., 2014, James et al., 2015, Birkenkampf et al., 2014] of the different objects in the environment, the robot can then plan and execute their tasks using these newly defined affrodances. In other approaches, the robot creates a plan and the human is able to issue change requests [Myers et al., 2003, Lin and Bercher, 2021] to the proposed plan to modify or reorder actions. Others allow the human to define a robot behavior, which is validated by the system [Howey et al., 2004] making system-initiated change requests. The bi-directional communication offered by MIP systems increases trust and flexibility, allowing them to be used in safety critical applications.

BTs fall naturally under the MIP umbrella since they are mainly designed by a human supervisor allowing them to create an abstract behavior. The robot is able to handle the low level decisions involving navigation or object manipulation. The following sections show how we extended BTs to follow the MIP planning paradigm on the task level instead of the lower levels seen in traditional BTs, increasing the flexibility and readability of the trees and trust in the robot running them.

6.2 Real-Time BT Reordering

This thesis introduces a new node type for BTs, the Dynamic Sequence Node (DSN). Designing the BT for the robot using DSNs allows the BT to reorder itself at run time depending on the world state. This extension adds run time optimization to BTs while maintaining the tree's modularity and readability. Tree modularity is imperative for code reuse, especially for Small and Medium-sized Enterprises (SMEs) that often lack the expertise in programming and robotics [Perzylo et al., 2019, Nguyen and LUU, 2020, Berger and Armstrong, 2022]. Readability, on the other hand, is required at run time to introspect the robot's state and explain its behavior and intention, which can be key factors in increasing the trust of the human coworkers towards the robot [Dammers et al., 2022]. The next section covers approaches that reduce the design effort of BTs and compare them to the extension introduced in this work. After that we go into the details of the DSNs and how they impact the resulting BT during design and execution phases.

6.2.1 Tree Synthesis and Real Time Optimization

A lot of effort goes into reducing BT design time and resources. This is mostly done by either synthesizing the BT using learning, planning, or a combination of both. Several approaches do this using Learning from Demonstration (LfD) [Styrud et al., 2021, French et al., 2019] or Genetic Algorithms (GAs) [Colledanchise et al., 2018, Iovino et al., 2021, Ångström, 2022] to generate the tree. However, training is often time consuming and the produced trees are not necessarily readable or reusable because of their size. Additionally, in case of assembly tasks, changing the process to produce a new variant of a given product, such as cases of lot-size-one production, requires reprogramming or retraining the agent and thus consuming resources in terms of human effort and computation time. This is a major disadvantage to companies that have to quickly respond to frequently shifting market demands.

To spare computational costs, other approaches build the tree at run time using backward chaining [Cai et al., 2021] or by combining learning and planning [Colledanchise et al., 2019, Fu et al., 2016]. These approaches avoid long training and guarantee completeness, however, they focus neither on readability of the tree nor on execution time. Additionally, creating a product variant using these approaches requires a lot of effort to remodel the end product. Especially when producing a variant means changing the assembly steps and not only the goal description.

Approaches for real time optimization of BTs exist but move away from traditional BTs reducing readability of the tree by introducing several new node classes and converting parts of the tree to an Hierarchical Task Network (HTN) [Rovida et al., 2017]. Another approach exists for child reordering, but it requires the behavior designer to create heuristic functions for each selector node, the selector nodes can then choose which of their children to execute depending on the predefined reward function [Merrill, 2019].

6.2.2 Dynamic Sequence Nodes

DSN-enabled BTs are designed and are the deployed on the robot. During execution, the trees control the robot as traditional BTs, with the added ability to modify themselves and optimize the ordering of the DSNs' children depending on the state of the environment as seen in Figure 6.1. At run time, DSNs create a weighted graph of their children and use a shortest path solver on the graph to reorder them. Next, we detail how we avoid infeasible action orders and create the graph.

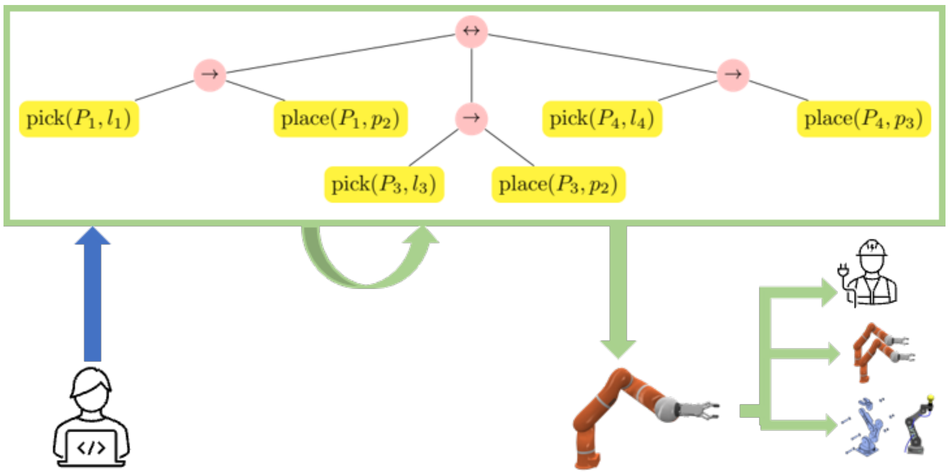


Figure 6.1: A developer designs the tree offline (blue arrow). After that, the tree controls the robot in real time (green arrows), which can interact with human co-workers, other robots, or objects in the environment. Meanwhile, the tree is able to adapt dynamically based on the world state due to the DSN.

6.2.3 Subtree Model Inference

Since our approach reorders the children of the DSNs in the tree, we run the risk of creating action conflicts, i.e., executing an action that disables their next sibling. To avoid this, we use Planning Domain Definition Language (PDDL) [McDermott et al., 1998] action models described in Section 5.3.2 that define the preconditions and effects of each action. PDDL models are usually available either as human knowledge or reside in some preexisting knowledge base. They have now become a standard action modeling language in planning. However, these models are only available for leaf nodes (actions) and not for the subtrees.

Sequence Nodes:

We use Algorithm 3 to infer models for internal nodes given the PDDL definitions for the available action nodes. The model of a sequence node, s, aggregates the add, del, and pre sets of its children. Line 3 updates the add_s effects of the sequence node by iterating over the children, $c \in \text{children}(s)$ in order adding the add effects, add_c, of each and removing the respective del_c effects. Similarly, we update the del_s effects of the sequence node in Line 4 by adding each child's del_c effects and removing its respective add_c effects. For the preconditions of the sequence node, Line 5 adds the preconditions of each child to pre_c if they are not already in the add_s set. This prevents unnecessarily adding a precondition that would be satisfied by one of the children before it is actually needed.

Algorithm 3: Inferring pre- and post-conditions for the sequence node s.

```
   Input: $s \leftarrow$ sequence node
1  $add_s, del_s, pre_s \leftarrow \emptyset$
2  for $c$ in children(s) do
3      $add_s \leftarrow (add_s \cup add_c) \setminus (del_c \cap add_s)$
4      $del_s \leftarrow (del_s \cup del_c) \setminus (add_c \cap del_s)$
5      $pre_s \leftarrow pre_s \cup (pre_c \setminus (pre_c \cap add_s))$
6  end
7  return $pre_s, add_s, del_s$
```

Fallback Nodes:

Inferring a similar model for fallback nodes is more involved. This is due to the uncertainty involved when ticking them. At design time, it is not clear which of the branches will succeed. This means that it's not clear which effects will be applied to the environment and what are the preconditions for this node. We consider the following approaches in this regard, but the choice on which one to use is domain specific.

- **Least Resistance:**

  The smallest set of conditions that lead this node to return `SUCCESS` is added to the node's preconditions and their corresponding effects are added to the node's effects. This set is selected by iterating over the children and selecting the branch with the smallest precondition set as seen in Algorithm 4.

Algorithm 4: Inferring pre- and post-conditions for the fallback node f in the least resistance approach.

```
  Input: $f \leftarrow$ fallback node
1 $add_f, del_f, pre_f \leftarrow \emptyset$
2 for $c$ in $children(s)$ do
3     if $|pre_c| < |pre_f|$ then
4         $add_f \leftarrow add_c$
5         $del_f \leftarrow del_c$
6         $pre_f \leftarrow pre_c$
7     end
8 end
9 return $pre_f, add_f, del_f$
```

- **Maximum Utility:**

  Maximizing the node's *add*-effects visiting all children and selecting the branch with the largest number of *add*-effects $add_f = add_m$, where m is seen in Equation 6.1. Maximizing this set enables more actions to be executed later as PDDL does not allow negative literals in the action preconditions. The approach is seen in Algorithm 5.

$$m = \underset{c \in children(f)}{\operatorname{argmax}} (|add_c|) \tag{6.1}$$

Algorithm 5: Inferring pre- and post-conditions for the fallback node f in the maximum utility approach.

```
  Input: $f \leftarrow$ fallback node
1 $add_f, del_f, pre_f \leftarrow \emptyset$
2 for $c$ in $children(s)$ do
3     if $|add_c| > |add_f|$ then
4         $add_f \leftarrow add_c$
5         $del_f \leftarrow del_c$
6         $pre_f \leftarrow pre_c$
7     end
8 end
9 return $pre_f, add_f, del_f$
```

- **Real Time Model Inference:**

  Another possibility is to infer the model at runtime by doing the inference every time this node is ticked. Before ticking the children, the fallback node goes through the children collecting the model of each child and whether it will be ticked given the current world state as well as how many children have to be ticked before one -symbolically-succeeds. However, this adds a computational overhead and might not be practical in industrial tasks, especially safety-critical ones.

This can be seen in Algorithm 6. But it makes the assumption that all the effects of a child (both add and delete) occur instantly once an action succeeds. This is not necessarily always the case, not only due to the lingering effects of actions mentioned in Section 5.3.3, but also because durative actions can fail at different points during their execution leading to multiple points of failure.

Algorithm 6: Inferring pre- and post-conditions for the fallback node using real time inference approach.

```
  Input: $f \leftarrow$ fallback node
1 $add_f, del_f, pre_f \leftarrow \emptyset$
2 for $c$ in children(s) do
3     if $\bigwedge pre_c$ then
4         $add_f \leftarrow add_c$
5         $del_f \leftarrow del_c$
6         $pre_f \leftarrow pre_c$
7     end
8 end
9 return $pre_f, add_f, del_f$
```

Since the model inference relies on the node order, it's not possible to directly apply this to a DSN or a parallel node. It is therefore not possible to have one of these as a descendant of a DSN in the tree. However, this can be solved by allowing a designer to directly define the model for these branches from the editor.

6.2.4 Real Time Tree Optimization

The main idea behind this approach lies in creating a directed weighted graph of the children of a given DSN, reducing the action ordering problem into a shortest path problem. These problems already have several commercially available solvers as well as well known algorithms that handle them efficiently, particularly practically for small instances. Additionally, recent research has gone into applying neural networks, which are suitable for solving larger instances [Bello et al., 2016, Garmendia et al., 2022].

Graph Creation

To create these graphs, we scan the tree at startup time. For each DSN, N we use Algorithm 7 to create a weighted directed graph. Initially, we start with a fully connected graph $\langle V, E\rangle$, Where V is the set of children $children(N) \cup ee$, ee is an added node denoting the current End Effector (EE) pose, $E = V \times V$ representing possible transitions between the different actions. After that, we iterate over these edges eliminating infeasible transitions and calculating weights for the feasible ones. Transition feasibility of edge $\langle a_1, a_2\rangle$ is determined by checking whether the preconditions of a_2 are invalidated by the effects of a_1 as seen in Lines 3 and 7.

In Lines 4 and 8, we calculate the costs of these transitions as discussed below.

The elimination step above significantly reduces the computational costs of the shortest path by reducing the number of edges, and therefore the solution space. Moreover, this increases safety of the execution because the robot won't be able to execute conflicting action sequences. This is done without hard coding the preconditions or the action sequences, leading to less work load on the tree designer compared to using traditional BTs.

Algorithm 7: Creating the graph of the children of a DSN and returning the shortest path starting at *ee* and visiting all children.

```
   Input: $N \leftarrow$ Dynamic Sequence Node
1  $V, E \leftarrow \langle children(c) \cup \{ee\}, \varnothing\rangle$
2  for $\langle v_1, v_2\rangle \in V \times V$ do
3      if $del_{v_1} \cap pre_{v_1} = \varnothing$ then
4          $w_{v_1,v_2} = cost(v_1, v_2)$                ▷ from Eq. 6.2
5          $E \leftarrow \langle v_1, v_2, w_{v_1,v_2}\rangle$
6      end
7      if $del_{v_2} \cap pre_{v_2} = \varnothing$ then
8          $w_{v_2,v_1} = cost(v_1, v_2)$                ▷ from Eq. 6.2
9          $E \leftarrow \langle v_2, v_1, w_{v_2,v_1}\rangle$
10     end
11 end
12 return shortest-path$(ee, \langle V, E\rangle)$
```

Cost Calculation

To order the children, we use a weight function considering two factors, time and quality of the action.

- **Time:**

  Here we consider the time it takes to execute the action, particularly within a given sequence, e.g., time to pick item x from location l_x by moving the end effector from location l_0. It is possible to integrate the execution time of durative actions, supported by newer versions of PDDL[1].

- **Action Quality:**

  Since we aim at integrating this system with multi-agent and Human-Robot Collaboration (HRC) assembly processes, we give preference to actions that enable more actions. To this end, an action that enables another agent (e.g., human coworker) to execute another action

[1] Versions after PDDL 2.1, which introduced durative actions.

instead of waiting will have higher quality than an action that blocks other agents.

Cost calculation is done recursively based on the node type. We start at the DSNs in the tree and propagate the calculation down to the leaves, then aggregate the calculated costs.

At the leaves, we calculate the cost for executing actions a, a' in that order as seen in Equation 6.2. Where $D(a, a')$ is the Euclidean distance covered when executing the actions, i.e., the distance moved by the robot's EE, which may be the distance between the EE poses, or the translational distance. We normalize this by R, which is the maximum reach of the robot. $I : A, s \rightarrow 0, 1$ is an indicator function determining whether an action $a \in A$ is executable in state s. $Q : A \rightarrow \mathbb{R}$ is the action quality and is calculated as seen in in Equation 6.3. $Q(a)$ is the number of feasible actions after executing action a normalized by the number of actions in our domain.

$$cost(a, a') = \frac{D(a, a')}{R} - I(a) \cdot Q(a) \tag{6.2}$$

$$Q(a) = \frac{|\{a' \mid pre_{a'} \subseteq S \cup add_a \setminus del_a\}|}{|A|} \tag{6.3}$$

To calculate the cost of the condition nodes, we consider the cost of the right sibling C_{rs}. This is because condition nodes are assumed to be instantaneous and never return running. They are also traditionally used as guarding conditions combined with a control flow node to control the execution of their sibling nodes. For example, they can be used to check whether a room door is open before trying to move inside. That's why we calculate it as seen in Equation 6.4, where $\epsilon \in \mathbb{R}$ is an arbitrarily small positive number to keep the condition nodes next to the same sibling after reordering the children.

$$Cost(cond) = C_{rs} - \epsilon \tag{6.4}$$

The costs of control flow nodes naturally depend on their children and their types. Since sequence nodes execute all their children, the cost, as seen in Equation 6.5 only sums the costs of the children, where N is number of children.

$$Cost(Seq) = \sum_{i=1}^{N} cost(C_i) \tag{6.5}$$

The cost of a fallback node follows the logic in [Merrill, 2019], where the fallback nodes execute the child with the highest utility, we return the cost as the cost of the cheapest child.

$$Cost(Sel) = \min_i C_i \tag{6.6}$$

Since parallel nodes tick all their children at the same time and only wait for T nodes to succeed. Their cost is calculated as the mean cost of the children multiplied by the success threshold T. We can further integrate the success probabilities as weights for each child instead of averaging over the children. However, this requires a more intricate action model, that includes sensor and actuator models, which are not always available.

$$Cost(par) = T \cdot \frac{1}{N} \sum_{i=1}^{N} C_i \tag{6.7}$$

Since decorator nodes are not standard, they allow the user to implement custom behavior either over the execution of their single child or manipulate the return value. Their costs are also user defined based on their policy. For example, the cost of a repeat-n decorator can be calculated as $n * cost(c)$, where c is the child, while the cost of a negation decorator would simply be the cost of the child.

6.2.5 Case Study

We now consider an industrial assembly use case. We model a BT to assemble a desk lap, seen in Figure 6.2. We show how the tree would look like using DSNs. We also show how a synthesized tree using [Colledanchise et al., 2019] as well as a traditional BT.

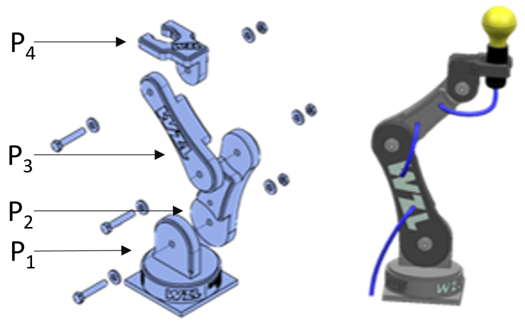


Figure 6.2: The assembly task demonstrator. The robot assembles the different parts and screws them together while a human worker connects the cable and installs the light bulb.

We consider a simplified version of the domain, where the robot has 2 actions at its disposal, *pick* and *place*. The domain has 3 predicates $free$, $holding(o)$, and $at(o, l)$, representing whether the gripper is free, the robot is holding object o, and whether o is at location l.

The PDDL models of the domain actions are given in Table 6.1. For the robot to execute the pick(o, l) action to pick object o from location l, the robot's gripper needs to be free, the object needs to be at location l. As an effect of picking the object, the gripper is no longer free, object

o is no longer at l, and the robot is *holding*(o). Similarly, the place(o, l) requires the robot to be holding object o. It causes the object to be at l, the gripper to be free and no longer holding o.

Table 6.1: Models of the robot's actions

Action	desc.	pre	add	del
pick$(?o, ?l)$	pick object $?o$ from l	free(), at(?o, ?l)	holding$(?o)$	at$(?o, ?l)$ free()
place$(?o, ?l)$	place object $?o$ at $?l$	holding$(?o)$	free(), at(?o, ?l)	holding$(?o)$

The environment contains the different parts of the lamp body P_i for $i \in [1, 4]$ seen in Figure 6.2. The parts are initially at locations l_i for $i \in [1, 4]$, respectively, and are to be attached together using screws.

Several traditional BTs can represent this assembly scenario simply by varying the order of the part assembly as seen in Figure 6.3. Non-of them have a clear advantage over the others without considering the context or the world state in real time.

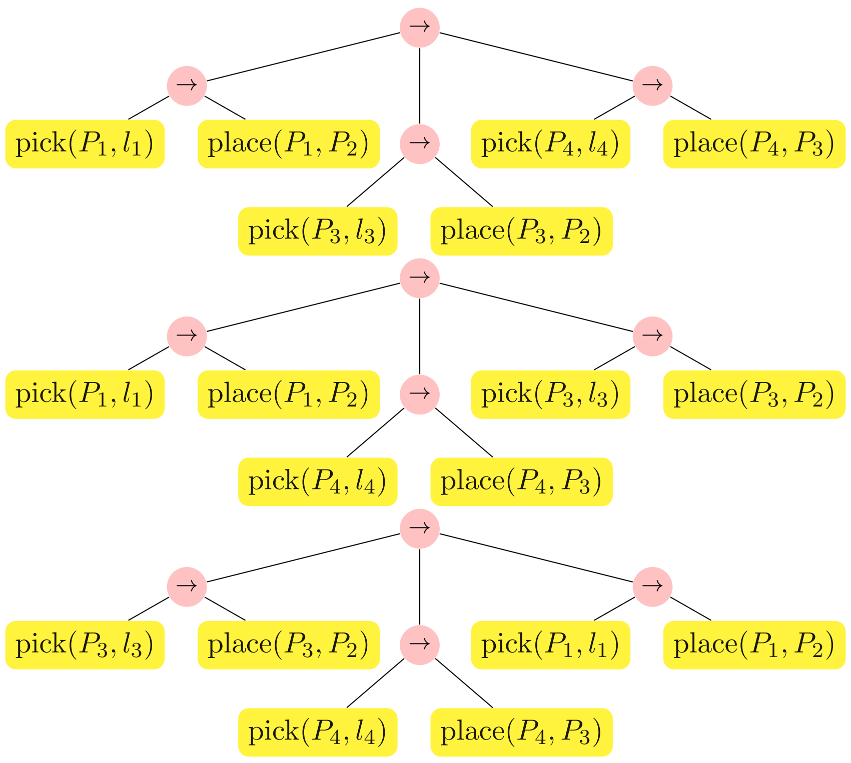


Figure 6.3: Different simple traditional BTs can be used to assemble the same four-part desk lamp.

They can be combined using control flow nodes as roots to compose a new tree that checks the world state before executing and then decides which tree to use. This can be seen in Figure 6.4, where condition nodes are added as prefix to the trees representing different assembly orders. Despite using only simple trees that use action nodes, we can see that the size of the resulting tree rapidly increases to become unmanageable, especially for a human designing the tree or a human supervising the robot and introspecting the tree at run-time. Such trees have some disadvantages for an SME trying to keep up with high and shifting demands which require fast prototyping and fast production:

- More human effort to design the tree
- Ad-hoc heuristic functions defined by the designer at design time
- The tree becomes less readable and more difficult to follow and maintain

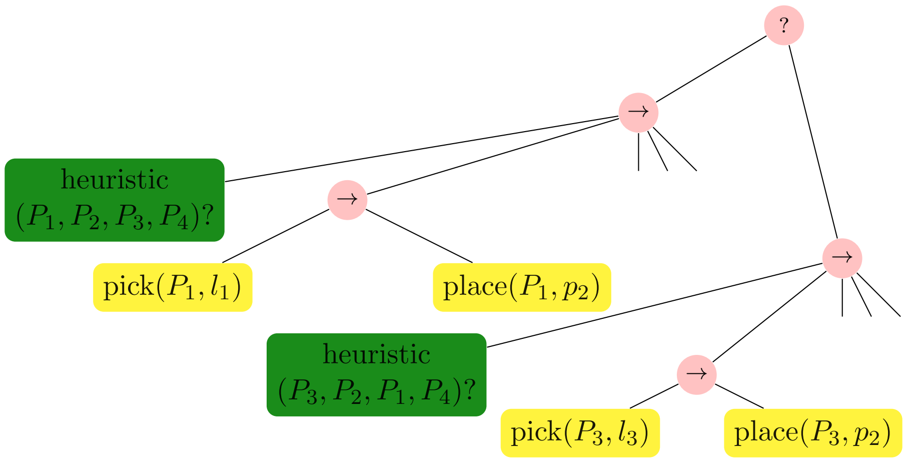


Figure 6.4: Using a traditional BT to assemble the lamp. The tree has to have a branch corresponding to each assembly ordering depending on which parts are closer to the robot. Only part of the tree is shown due to space limitation.

BT synthesis approaches can also be used to dynamically build the tree. However, they are created starting from a goal condition encapsulated in a condition node, which is grown into a full tree as seen in Figure 6.5. In order to optimise execution time of the tree, we need to incorporate execution time of the individual action nodes in the action selection criteria while growing the tree. Even then, such approach does not reorder the branches at runtime. This means it can grow a branch to avoid an obstacle or satisfy an action's precondition, but not to increase the throughput after the tree had been created.

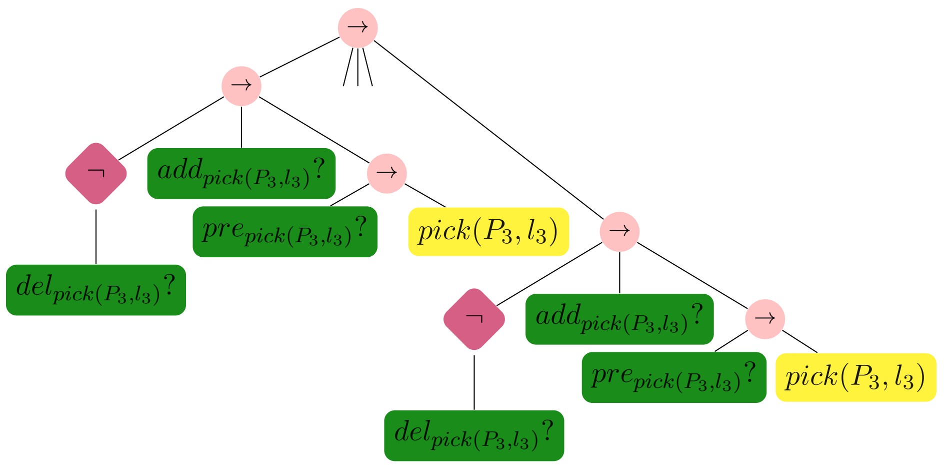


Figure 6.5: Synthesized tree using the approach of [Colledanchise et al., 2019] given the goal state assembled(P_1, P_2, P_3)?.

Figure 6.6 shows a tree that uses DSN to assemble the parts. The tree is much simpler than the other approaches and is more reactive to the environment allowing the robot to maintain high throughput even if parts locations are changed at run time. The robot is also able to incorporate the detours resulting from obstacle avoidance in its action reordering. This is done without affecting the size, modularity, or readability of the tree.

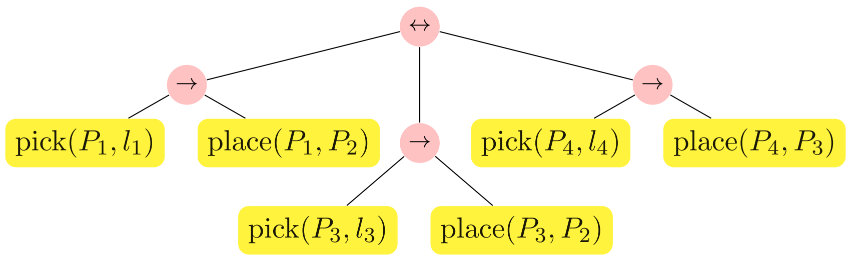


Figure 6.6: The tree created with MIP using DSN represented as a node with the symbol ↔.

6.3 Collaboration with BT-Driven Agents

Another form of HRC is the collaborative assembly, where a human and a robot work side by side on the same work piece to fulfill the same goal. This is considered the highest form of collaboration [Baier et al., 2022] as it involves heightened safety requirements due to the proximity of the human to the robot, especially when the collaboration involves handing over the work piece. In these scenarios, industrial requirements and expectations can be divided into three classes:

- **Trust:**

Human coworkers and supervisors should be able to trust the robot coworkers. In the context of robotics, trust and reliability are tightly coupled with robot intentions and action explainability.

- **Compliance:**

  Conforming to industrial safety standards including. These include not only stopping the robot on collisions, but may also include anticipation and mitigation of collisions.

- **Throughput:**

  High throughput and low idling time. Idling time is generally undesirable in industrial applications. It increases when a worker has to wait for another, which can happen when a robot is waiting for the human or the human is waiting for the robot. In HRC, we can reduce it by allowing the robot to work on other tasks instead of waiting for the human as well as choosing actions that enable the human to work on their tasks instead of waiting for the robot. This also applies in multi-robot systems.

To bridge the gap between the current state of BT research and these requirements, we introduce Human Action Node (HAN) as an extension to BTs with a framework to allocate tasks to the human and the robot in this scenario. The next sections detail this framework.

The proposed framework allows humans and BT-based robots to work side by side. This is achieved by including action models for the actions of the humans and the robot. The robot uses a rule-base to dispatch trees based on the actions currently performed by the human and by the robot. As seen in Figure 6.7, we use a tree dispatcher which can hold a dynamic list of trees, each describing a robot behavior. The trees can be paused if the robot is waiting for the human, in that case, we move it to the list of paused trees. During that time, the robot is able to pick another tree from the list of ready trees. Trees can move between the two lists either when a tree execution ends or when it reaches a human action. Pausing and resuming trees depend on a rule-base, which is updated by a perception module. When certain rules hold, the corresponding trees are marked either as paused or ready.

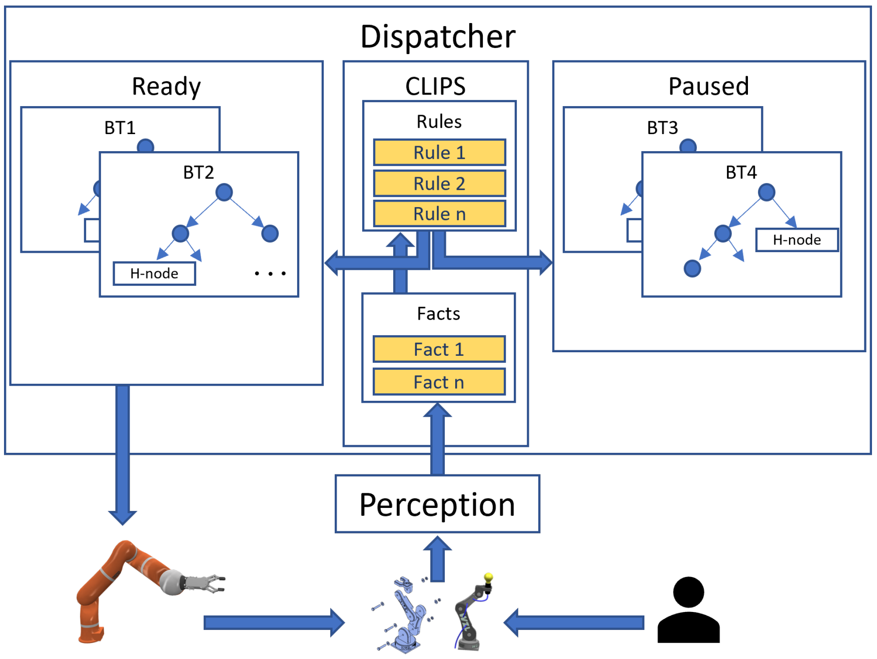


Figure 6.7: An overview of the proposed framework for using H-nodes with BTs.

6.3.1 Rule-Based Systems

C Language Integrated Production System (CLIPS) [Wygant, 1989] is a rule-based expert system introduced in the late 1980s. It has since been used for decision making in industrial scenarios and has recently been introduced as a teleoreactive behavior model for robots [Niemuller et al., 2018] in the Robo Cup Logistics League (RCLL). It has 3 main modules, a Knowledge Base (KB), a Rule Base (RB), and a reasoning engine. The KB usually holds all information that is true about the world and the world state, such as the locations of objects and the values of different object attributes. The RB models the world dynamics and how the robot should react to different states of the world. The reasoning engine matches states of the world to the different rules using the Rete matching algorithm [Forgy, 1988].

CLIPS rules are split into two parts, a header representing the antecedents and a body representing the consequent. The header is a collection of predicates used as conditions to the rule execution, and also help in grounding variables that match certain patterns. The body usually contains a procedure to be executed by the agent once a rule fires.

6.3.2 From PDDL Models to Rules

At startup time, our framework translates the PDDL definitions into CLIPS rules. Instead of using action preconditions as the rule header, which has been done in [Niemuller et al., 2018], we use the action's effects. This causes rules to fire when an action has been successfully executed rather than when an action is feasible.

The structure of the translated CLIPS rules is shown in Equation 6.8. These rules are added to CLIPS's RB. The left hand side of the arrow (rule header) is an aggregation of the actions effects. The right hand side marks the tree, t as active.

$$\bigwedge_{a \in \text{add}_h} a \wedge \neg \bigvee_{d \in \text{del}_h} d \Longrightarrow \text{activate}(t) \tag{6.8}$$

The main purpose is for rules to fire when the human coworker has finished their tasks. To this end, the conditions of translated rules contains the actions effects, rather than the traditionally used preconditions. This is done to allow the robot to continue its tasks when it recognizes that the human task is finished. For example, in the task lamp assembly, if the sub task of the human worker is to install a cable c and place a screw s, we assume that the human knows when such task is feasible and is capable of deciding whether or not they can start. For the robot, however we need to explicitly model the expected effects of this action, namely that the cable is installed in its given place. This is translated into the CLIPS rule $\textit{in-place}(c_1) \wedge \textit{in-place}(s) \implies \textit{activate}(\textit{Lamp-assembly})$

6.3.3 Human Action Nodes

A HAN is added to the tree an action node. When the parent node ticks a HAN, it returns a new status `PAUSED` and the tree is paused. At this point, the dispatcher can assign another tree to the robot, who starts executing it. Meanwhile, the perception modules keep updating CLIPS' KB. Whenever the post conditions of an action h are satisfied, the respective rule fires re-activating the tree. To handle the return status, we adapt the control flow nodes as seen in the next section.

6.3.4 Control Flow Nodes

In order to handle the new node type and the new return status, `PAUSED`, the tick functions of control flow nodes have to be modified. The modifications of a control flow node include initializing the node's children as well as the decision on whether to pass their returned status directly to the node's parent. These modifications are discussed in detail below.

Sequence Nodes

Sequence nodes allow the robot to pause its current tree while the human is performing a sub-task, either after handing over the work piece or leaving it in place. As seen in Algorithm 8, a modified sequence node works similar to a traditional one, returning its children's status to its parent except in case of SUCCESS, where it simply ticks the next child.

Algorithm 8: Modified sequence node tick function to handle the PAUSED return status of Human Action Node.

```
Function tick($s$):
    input: $s \leftarrow$ ticked sequence node
    $N \leftarrow |s.children|$
    if $s.status \notin \{$PAUSED, RUNNING$\}$ then
        $s.child\_index \leftarrow 1$
        reset($s$)                              ▷ initialize the children
    end
    for $i \in [s.child\_index, N]$ do
        state = tick($s.children[i]$)
        if $state \neq$ SUCCESS then
            $s$.state = state
            return $s$.state
        end
        $s$.child_index $\leftarrow$ $s$.child_index + 1
    end
    $s$.child_index $\leftarrow$ 1
    return SUCCESS
```

Fallback Nodes

Algorithm 9 shows how the tick function of fallback nodes with memory are modified to handle the new PAUSED status. The modification allows the node to return the state of its child as long as it has not failed. This passes the PAUSED status to the parent if one of the leaf nodes is paused. Using a fallback composition allows the robot to attempt performing a task. If the robot fails, the human can proceed to perform the task themselves after the tree had been paused (for this behavior, a notification system needs to be implemented to notify the human of such failure). We can switch the order to allow the human to attempt the execution first and the robot to be the fall back if the human fails (e.g., when the human is too busy with other tasks), which requires some failure criteria such as a timeout based on the duration of a durative action.

Algorithm 9: Modified fallback node tick function to handle the `PAUSED` return status of Human Action Node.

```
Function tick($f$):
    input: $f \leftarrow$ ticked fallback node
    if $f.status \notin \{$PAUSED, RUNNING$\}$ then
        $f.child\_index \leftarrow 1$
        reset($f$)                              ▷ initialize the children
    end
    for $i \in [f.child\_index, |f.children|]$ do
        $status \leftarrow$ tick($c$)
        if $status \neq$ FAILURE then
            $f.status \leftarrow status$
            return $status$
        end
        $f.child\_index \leftarrow f.child\_index + 1$
    end
    $f.child\_index \leftarrow 1$
    return FAILURE
```

Parallel Nodes

A parallel composition allows the human and the robot to work on the work piece at the same time. The tree can be paused if the robot finishes its sub tasks before the human. In the opposite case, the tree does not need to be paused because the robot will continue working on sub tasks defined in the tree, while the human can move on to other tasks. The modifications of the parallel node tick function are shown in Algorithm 10. The success and failure criteria are the same as the traditional parallel node, but ticking the node returns `PAUSED` only if all its remaining children are paused.

Algorithm 10: Modified parallel node tick function to handle the `PAUSED` return status of Human Action Node.

```
Function tick($p$):
    input: $p \leftarrow$ ticked parallel node
    if $p.paused$ then
        $p.paused$ = False
        for $c \in p.children$ do
            if $\neg c.paused$ then
                $children\_status[c] \leftarrow$ tick($c$)        ▷ only if not paused
            else
                $children\_status[c] \leftarrow$ PAUSED
            end
        end
    end
    $N \leftarrow |p.children|$
    $S \leftarrow |\{c \in p.children \mid children\_status[c] = $ SUCCESS $\}|$
    $F \leftarrow |\{c \in p.children \mid children\_status[c] = $ FAILURE $\}|$
    $R \leftarrow |\{c \in p.children \mid children\_status[c] = $ RUNNING $\}|$
    if $S \geq M$ then
        return SUCCESS
    end
    if $N - F < M$ then
        return FAILURE
    end
    if $R > 0$ then
        return RUNNING                 ▷ if at least one child is running
    end
    $p.paused \leftarrow True$              ▷ remaining children are paused
    return PAUSED                      ▷ this node is marked as paused too
```

6.3.5 Case Study

In order to demonstrate these nodes, we show a proof of concept application, where a cobot collaborates with a human worker to assemble two different products. In an industrial scenario, cobots are commonly equipped with power tools, such as screw drivers, or a gripper, suitable for grasping the objects involved in the task. Recent approaches target a high level of dexterity by allowing a robot to change its own end effector at runtime [Bachmann et al., 2023]. However, this requires a specific workspace that integrates the different tools and work pieces. Additionally, [Bachmann et al., 2023] is a fairly new approach that is not yet available in production plants, especially SMEs.

In our setup, we allow the robot, with the human coworker to work on the assembly of multiple products at the same time. The actions available in this domain are listed in Table 6.2 with their respective PDDL models.

The table shows the actions available for the robot and the modeled actions of the human worker. Since the robot is only equipped with an electric screw driver, it can only perform fastening actions [2], while the human worker performs the pick and place tasks to compensate for the robot's lack of grippers. To execute the fasten action, the screw $?s$ has to be $in-place$, and as an effect the predicate $fastened(?s)$ is added to the world state. The human is able to perform the $place-screws$ and $remove-lamp$ action at any time with no preconditions, only the effects that the three screws, s_1, s_2, and s_1, are $in-place$ or that they are no longer $in-place$.

As seen in Figure 6.8, in addition to the work pieces and the components, the robot workspace also shows a screen that can display the currently active BT with the returned value for each node, allowing the human to keep track of the robot's intention and actions at runtime. In this scenario, the products being assembled are a gearbox for an e-bike and a desk lamp used in Section 6.2.5.

Figure 6.9 shows a traditional BT describing the task of the robot, fastening the three screws sequentially. Given this BT, the robot is not informed of the human coworker's tasks and does not allow the robot to pause its tasks and take on a new tree. The robot can only switch tasks, when the tree execution ends. Additionally, the lack of knowledge about the human tasks, could endanger the coworker. Figure 6.10 shows alternate trees that use our proposed HAN to model the human worker's actions. This allows the robot to execute other trees while waiting for the human to place different parts as seen in Figure 6.10a. It also enables both the human and the robot to work on the same work piece using a tree with parallel nodes as seen in Figure 6.10b [3]

[2] Using the same logic, we can also include an unfastening action with its respective pre- and post-conditions, but this is omitted here for brevity.

[3] The execution can be seen in the video: https://youtu.be/nrJBUNkToNY?si=T-8B9vw36yh9B1kj.

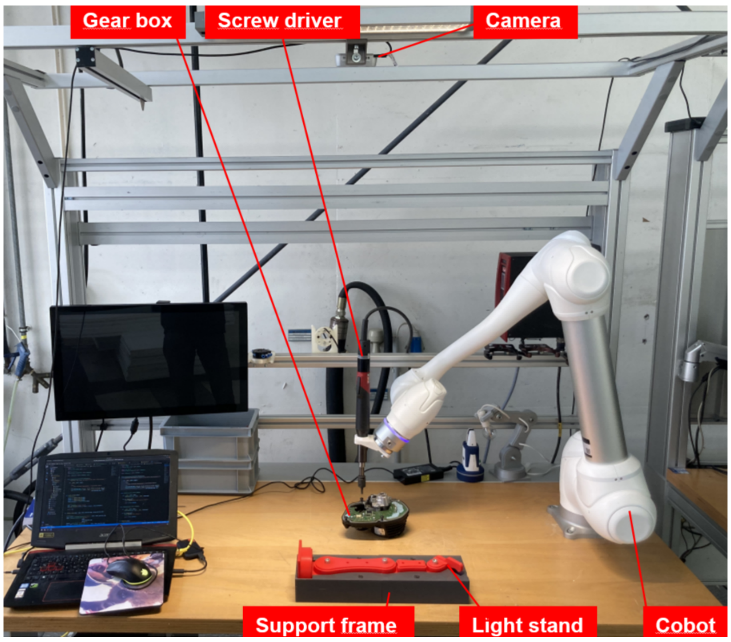


Figure 6.8: Use case for demonstrating the HRC assembly using BT. The demonstrator includes a robot arm equipped with a screw driver. The robot works side by side with a human worker sharing the workspace and work piece.

Table 6.2: Models of the available actions.

Action	pre	add	del
fasten(?s)	screw (?s) in-place(?s)	fastened(?s)	-

place-screws
(?s_1, ?s_2, ?s_3)
	-	in-place(?s_1)
		in-place(?s_2)
		in-place(?s_3)
			-

remove-lamp	-	-	in-place(?s_1)
			in-place(?s_2)
			in-place(?s_3)

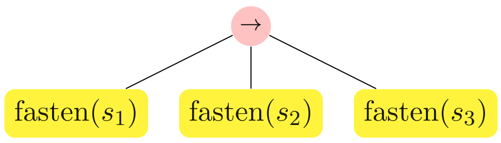
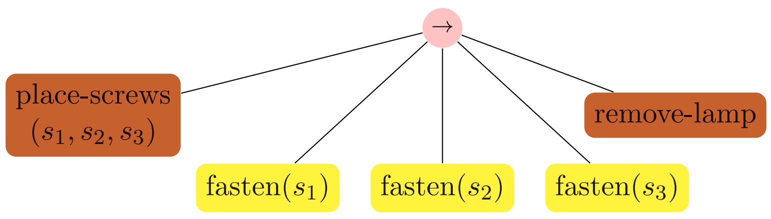
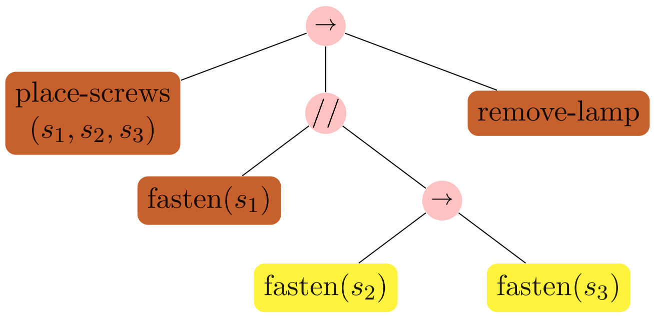


Figure 6.9: The traditional BT for the assembly of the gear set, similar to that for assembling the lamp.



(a) The human worker's task is to place the screws in their respective places while the robot's task is to fasten them in sequence afterwards.



(b) This tree shows parallel assembly, where the human worker and the robot can perform their respective fastening tasks on the same work-piece at the same time.

Figure 6.10: Different variations of the lamp assembly, where the human and the robot work together. HANs are highlighted in red.

6.4 Behavior Trees for Teleoperation

Industrial and domestic service domains have use case for robot teleoperation as seen in Chapter 5 as well as in previous work. Full, or partial, automation of these tasks involves a steep learning curve for the operator to directly control the robot arm, or to program the arm. To slightly flatten the learning curve, we need to guide the user's input assisting them in the teleoperation task. This type of assistance is use case specific, i.e., each use case requires reprogramming the behavior of the robot, which adds to the overhead of learning. To overcome this, we follow in the footsteps of some approaches that offer low-code solutions to robot programming using BTs. They don't inherently support teleoperation because they were created to control game Artificial Intelligence (AI) agents and are traditionally used to control fully autonomous agents. We extend BTs to include nodes that implement this guidance to assist the human operator. The modularity of the BT allows us to offer a library of atomic behaviors, e.g., line following or specific rotations that can be composed together to define new behaviors. The modularity can also be exploited to transfer behaviors across robot platforms.

In this work, we introduce PHAST BTs, which use a new node type, Shared Control Action Node (SCAN). This node allows BTs to handle shared autonomy control of a robot by modifying the user's input to follow a predefined activity. This section introduces a state of the art approach in activity representation which targets shared autonomy teleoperation introduced in [Quere et al., 2020]. After that, it goes over the general structure of PHAST BTs in addition to the details of SCAN and how they work. PHAST BTs are a structurally restricted class of BTs, we limit the structure of the tree to reduce the mental effort required to design the tree and introduce the SCAN which modifies the user input during operation. This type of input modification has shown great success in earlier approaches [Behery, 2016, Quere et al., 2020, Muelling et al., 2017], which lacked extensibility.

6.4.1 Shared Autonomy Activity Representations

Robot teleoperation is a wide spectrum of robot control paradigms. It covers direct human control, supervisory control, and different levels of shared autonomy, where human's direct control signal is modified to guide the robot's EE offering assistance to the human operator [Sheridan, 1992, Goodrich et al., 2013, Behery, 2016].

In this section, we focus on shared autonomy, where an agent and a human operator share control over a robot arm. This has been studied in many domains, including robot assisted surgery and in-orbit robot teleoperation. Several approaches are applied, including virtual fixtures [Marayong et al., 2003], which are commonly used in robotic surgery to restrict the EE motion to a specific direction or a specific path. Other geometric reasoning approaches assist users in Unmanned Aerial Vehicle (UAV) flight [Israelsen et al., 2014] and determining an operator's intention to guide the EE for grasp assistance [Muelling et al., 2017]. As the field matured and technology advanced, several other techniques considered allowing mainstream users, with little to no programming and robotics in the health care domain, to program the robots for offering task-specific assistance [Quere et al., 2020].

[Quere et al., 2020] introduced Shared Control Templates (SCTs), which represent an activity using Labelled Transition System (LTS) [Keller, 1976] [4]. They divide an activity into a set of phases, each represented as a state in a FSM [5]. Each state has a set of Input Mappings (IMs) and Active Constraints (ACs), which are functions that guide the EE's movement based on the user's input. Formally, a SCT is a LTS $L = \langle P, \Lambda, E \rangle$, where P is the set of activity phases. Each phase, $p \in P$, is represented as a tuple $\langle f_p, m_p \rangle$, containing an ordered list of ACs, f_p and an ordered list of IMs, m_p. An AC is a user defined virtual fixture guiding the user's input to constrain the EE within certain geometric bounds. On the other hand, an IM, maps the user's input from the low dimensional input space of a joystick or

[4] Originally introduced as Named Transition Systems

[5] [Quere et al., 2020] describes SCTs as a FSM, however they can be more accurately described as a LTS since SCTs do not have explicit start and end states. The transition function also relies on some metrics based on the robot's EE frame and not only on the input $i \in \Sigma$, where Σ is a finite non-empty set of input symbols. Therefore, these transition conditions are better modeled as a *label* in a LTS.

a Brain Computer Interface (BCI) to the high dimensional task -or robot configuration- space. A label $\lambda \in \Lambda$ could have different uses. They may be used to represent expected input, conditions that allow triggering the transition, or actions performed during the transition.

It is also worth mentioning that we make a distinction between the world state and the phases of an LTS representing an SCT. A phase of the SCT is equivalent to the current stage of execution of the activity, while the world state is defined by a finite number of predicates and function symbols describing the world. For example, a phase in pouring could be the stage of the pouring activity where the bottle is being tilted. During this stage, the world state could be described using a state space vector $\vec{s}$ containing the the predicate *capped*(*Bottle*) and the function *pose*(EE), which describe whether the bottle is capped and the pose of the end effector, respectively. In our work, we abstract away from the details and assume there's an underlying KB which is updated in real time by the robot's perception. SCTs defined more formally in Definitions 6.4.1 and 6.4.2, respectively.

Definition 6.4.1 (Shared Control Template)**.** An SCT is defined as the tuple $\langle P, \Lambda, E\rangle$, where P is a set of phases, Λ is a set of labels, and $E \subseteq P \times \Lambda \times P$ is a relation of labelled transitions between the phases. A transition $\langle p, \lambda, p'\rangle \in E$ is a transition from phase p to phase p', where $\lambda \in \Lambda$ is a condition to allow triggering the transition.

Definition 6.4.2 (Activity Phase)**.** An activity phase p is defined as a tuple $\langle f_p, m_p\rangle$, containing an ordered list of ACs f_p and an ordered list of IMs m_p. These contain user defined functions acting as virtual fixtures to guide the user's input. Each is modeled as a function $f \subset \mathbb{R}^6 \times |\vec{s}| \times \mathbb{R}^6$ and $m \subset \mathbb{R}^6 \times |\vec{s}| \times \mathbb{R}^6$ [6]. The world state during an activity phase is updated using these functions $s' = f(I, s)$ and $s' = m(I, s)$.

For example, the activity of opening a drawer is divided into 3 phases, approaching the handle, grasping it, and finally pulling it parallel to its axis. The first phase, starts at a given distance between the EE and the handle. In this phase, the EE movement is guided on a virtual line between the EE and the handle. In the second phase, the orientation of the gripper is adjusted to match one of the grasp poses of the drawer handle. It also starts at another distance threshold between the EE and the handle. The third phase, starts when the gripper is closed holding the handle, and guides the movement along the axis parallel to the horizon. As seen here, the transition between the phases mostly relies on the EE's pose [Quere et al., 2020]. Translations can be performed by using an AC that interpolates a line between the EE and a target location and projects the input commands on it. Rotations (and some translations) are performed by using an IM that limits the motion on a given axis of Cartesian space.

[6] For teleoperation use cases, we are concerned with only the 6-Degree of Freedom (DoF) EE pose and will assume without loss of generality that $\vec{s} \in \mathbb{R}^6$ unless mentioned otherwise.

Inspired by the work on SCT, we propose a BT extension that handles a user's input to guide the movement of the robot's EE to perform activities. We assume that both AC and IM belong to a set of predefined functions that can output a new EE pose, given the current one , i.e., $f \subset \mathbb{R}^6 \times \mathbb{R}^6$ and $m \subset \mathbb{R}^6 \times \mathbb{R}^6$, the input to these functions can be extended to include other information about the world state. For this extension, we restrict the structure of BTs, introduce new node types, and show that the given trees can bisimulate SCTs.

6.4.2 Tree Structure

The general structure of the PHAST BT is shown in Figure 6.11. Each subtree represents a phase of the activity's SCT. We limit the BTs to four levels with a fallback root node whose children are all sequence nodes.

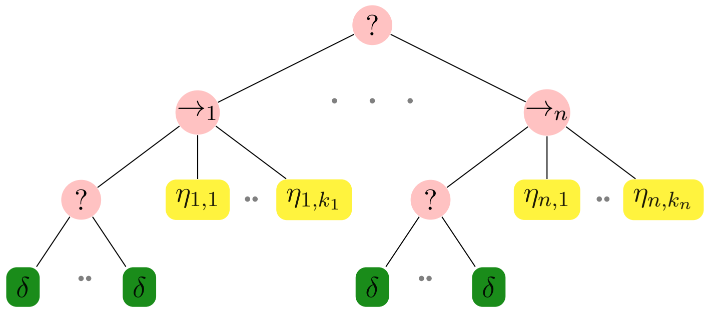


Figure 6.11: General structure of a PHAST BT. Limited to four levels and a fallback node for the root. The second layer contains only sequence nodes, each corresponds to an ADL phase. Each of which is a parent to a fallback node with δ-node children (green) representing the combination of previous states and geometric condition(s) followed by the SCANs (yellow).

Equation 6.9 shows the structure of an activity with n phases each containing multiple transition conditions and multiple SCANs. F, S, and C represent Fallback, Sequence, and Condition nodes, respectively, while $\eta_{i,j}$ represent SCANs. This can be represented using the tree shown in Figure 6.11. Given this structure, the translation into SCT becomes straightforward. We show that a tree $\mathcal{T}$ with n sequence nodes $s_n ... s_1$ can be translated into an SCT, which is a LTS $L = \langle P, \Lambda, E \rangle$. A phase $p_i \in P$ represents the i^{th} child of the root. Each p_i is an ordered list of auxiliary functions $f_{\eta_{i,1}}, \dots, f_{\eta_{i,k}}$ which are parameters of the SCANs in the i^{th} child. The condition nodes $c_{i,1}, \dots, c_{i,m}$ are grouped into an Edge $e_i \in E$ that goes from phase p_{i-1} to phase p_i and is represented by the tuple $\langle p_{i-1}, c_i, p_i \rangle$.

$$F(S_1(\delta_{1,1}, ...\delta_{1,m}, \eta_{1,1}, ...\eta_{1,k}) \; ... \; S_n(\delta_{n,1}, ...\delta_{n,m'}, \eta_{n,1}, ...\eta_{n,k'})) \tag{6.9}$$

6.4.3 Shared Control Action Nodes

Our proposed SCANs work similar to the IMs and the ACs used in the SCTs [Quere et al., 2020]. These nodes first calculate the next EE pose given the current one and the user's input. PHAST BTs provide SCANs to perform simple mappings and actively constrain the EE movement, respectively. We do not differentiate between IMs and ACs since their semantics are the same. The only difference is the restriction on IMs to map the input to a given world axis as opposed to the ACs which are allowed to further modify the input.

SCANs can read the EE pose from a world state, calculate the next pose using the respective predefined function, and send it to the low level controller. Equation 6.10 shows the tick function of the SCAN, the function f returns the next state s_{t+1}, e.g., EE pose, given the current one s_t and the user input I. This does not affect the modularity of the tree since they can still be used separately and independently. Placing these nodes in a sequence composition gives the same effect as using IM and AC in the SCT framework. Such a composition can be seen in Equation 6.11 where $m \in \mathbb{R}^6$ is the mapping parameter and $f : \mathbb{R}^6 \times \mathbb{R}^6 \to \mathbb{R}^6$ is the auxiliary function used as the AC.

$$s_{t+1} = f(I, s_t) \tag{6.10}$$

$$S(\eta(m), \eta(f)) \tag{6.11}$$

The return status of these nodes is always TRUE so they don't need to return `RUNNING` since they update the EE pose in small steps, which are inversely proportional to the tick frequency of the tree.

Each of the sequence nodes in a PHAST BT represents a phase of an activity. Next we show how the trees can switch between these phases.

6.4.4 δ-Branch

In order to compensate for the SCT's flexibility in combining phase transitions with labels, we use a combination of a decorator node and a condition node as seen in Figure 6.12. Using this results in an extra branch with a parameter p describing a phase, which is represented by a sequence node. The decorator node returns `SUCCESS` iff the last ticked node s in the second level of the tree is p. Its policy is $\delta(p) \hat{=} (s = p) \wedge \lambda$ where λ is the condition of the child condition node. As a short hand, we use $\delta_p(\lambda)$ to represent the policy and a single leaf node seen in Figure 6.11 with the symbol: δ.

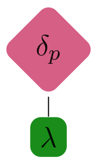


Figure 6.12: Unfolding a δ-node results in a decorator node with a condition child. The decorator checks whether the previous state was p before checking the condition λ.

Using the δ-branch reduces the maintainability of the trees when adding a new node, similar to the issues arising from adding a state to an FSMs. However, they are not necessary in practice and can be replaced by using conditions directly as seen later in Section 6.4.7. BTs have been successfully used to represent different tasks in many domains without the need to use such a mechanism [Iovino et al., 2020]. In this work, however, we employ these branches in order to simulate the edges present in a SCT, limiting the state transitions to predefined ones, to prove that PHAST BTs can simulate SCTs.

6.4.5 Relation To SCTs

In order to compare PHAST BTs to SCT [Quere et al., 2020], we provide two algorithms to translate each approach to the other. After that we show that the generated PHAST BT simulates the given SCT.

Algorithm 11: Translation of PHAST BT $\mathcal{T}$ to SCT.

```
Input: $\mathcal{T} \leftarrow$ PHAST BT node
1  $p_0 \leftarrow [\mathcal{I}]$
2  $P, \Lambda, E \leftarrow \langle \emptyset, \emptyset, \emptyset \rangle$
3  $\widetilde{\mathcal{T}} \leftarrow \text{reverse}(\mathcal{T})$
4  for $p_i \in \widetilde{\mathcal{T}}$ do
5      $\lambda \leftarrow \top$
6      $m_{p_i} \leftarrow \emptyset$
7      for $s \in children(p_i)$ do
8          if $type(s \hat{=} \delta_p(\lambda_s))$ is $\delta$ then
9              $\Lambda.\text{add}(\lambda_s)$
10             $E.\text{add}(\langle p, \lambda_s, p_i \rangle)$
11         end
12         if $type(s)$ is $\eta$ then
13             $m_{p_i}.\text{add}(f_s)$
14         end
15     end
16     $P.\text{add}(p_i)$
17 end
18 return $\langle P, \Lambda, E \rangle$
```

Algorithm 11 shows a translation of a PHAST BT rooted at $\mathcal{T}$ to SCT.

In Lines 1 and 2, we initialize the initial phase p_0 with $\mathcal{I}$ which is an identity function, i.e., when no conditions apply, we directly apply the user's input. In Line 3, we reverse the given Tree $\mathcal{T}$ to maintain the priorities of the children when transformed into activity phases. After that, the loop starting in Line 4 creates a phase (state) for each sequence node p_i in second level of $\mathcal{T}$. We iterate over the children of p_i adding a label λ and an edge $\langle p, \lambda_s, p_i \rangle$ for every δ-branch of the form $\delta_p(\lambda_s)$ and an IM for each SCAN with the corresponding auxiliary function f_s.

Algorithm 12: Translation of SCT $L = \langle P, \Lambda, E \rangle$ to PHAST BT

```
   Input: $L \leftarrow$ SCT $L = \langle P, \Lambda, E \rangle$
1  $\widetilde{\mathcal{T}} \leftarrow$ FALLBACK()
2  $\widetilde{\mathcal{T}}$.add-child($\mathcal{I}$)
3  for $p \in P$ do
4      $s \leftarrow$ SEQUENCE()
5      $f \leftarrow$ FALLBACK()
6      for $\langle p', \lambda, p \rangle \in E$ do
7          $f$.add-child($\delta_{p'}(\lambda)$)
8      end
9      $s$.add-child($f$)
10     for $m \in p$ do
11         $\mathcal{F}_m \leftarrow$ SCAN($f_m$)
12         $s$.add-child($\mathcal{F}_m$)
13     end
14     $\widetilde{\mathcal{T}}$.add-child($s$)
15 end
16 $\mathcal{T} \leftarrow reverse(\widetilde{\mathcal{T}})$
17 return $\mathcal{T}$
```

Algorithm 12 shows how we can translate an SCT $L = \langle P, \Lambda, E \rangle$ into an PHAST BT. The tree is created with a Fallback node as the root. For each phase $p \in P$, we collect the incoming edges $\langle p', \lambda, p \rangle$ and create a corresponding δ-branch $\delta_{p'}(\lambda)$ as seen in Line 7. After that we create a SCAN for each IM and AC in p. Next we create a sequence composition of those nodes and add it as a child of the root. Finally, Line 16 reverses the order of the root's children so they are ticked in order at execution time. We have now shown that given an SCT, we can construct a PHAST BT and vice versa. In the next section, we show that SCT and PHAST BT are equivalent in their expressive power.

6.4.6 Equivalence Through Bisimulation

In this section we prove that SCTs and PHAST BTs are able to represent the same activities. To this end, We show that for any SCT L, there exists a PHAST BT $\mathcal{T}$ such that $L \sim \mathcal{T}$ where $\sim$ is a bisimulation relation. That is, the world state, e.g., EE pose, calculated using L, s'_L is the same as the one calculated using $\mathcal{T}$, $s'_{\mathcal{T}}$.

A bisimulation relation is defined using a back-and-forth game [Ehrenfeucht, 1961], which formulates a condition for discernability between two systems. In this game two players P_1 and P_2 are provided two structures, G and H with the initial states $g_\circ$ and $h_\circ$ respectively. At each game step, P_1 chooses to move in either G or H, by moving along one of the arrows from state g to g' or h to h' in the respective structure. Player P_2 must make a move in the other structure. If some predicate holds in either one of g' or h' but not the other, then P_2 loses and the game ends, P_2 also loses if no move is available in the respective structure. If P_1 has no available moves, then P_1 loses. If G and H have finitely many states, then P_2 can have a winning strategy iff the same predicates, which hold at any given state h, also hold at a respective state g.

Such games are used to show equivalence of state transition systems, i.e., that they interact with the environment in the same way. For two given systems, A and B, if a binary relation exists between the states of A and the states of B, which can be used as a winning strategy by P_2, then this relation is a bisimulation and A and B exhibit the same behavior.

Definition 6.4.3 (Simulation Relation)**.** $L' = \langle S', \Lambda', T' \rangle$ is in a simulation relation with $L = \langle S, \Lambda, T \rangle$ ($L' \rightsquigarrow L$), iff:

1. for each state $s \in S$, there exists a state $s' \in S'$ such that all predicates that hold in s, hold in s' (i.e., s is indistinguishable from s') and
2. for every outgoing transition $t \in T$ from s_1 with label $\lambda_{1,2} \in \Lambda$ into state $s_2 \in S$, such that $t \doteq s_1 \xrightarrow{\lambda_{1,2}} s_2$, there is a transition $t' \in T'$, such that $t' \doteq s'_1 \xrightarrow{\lambda'_{1,2}} s'_2$, where s'_2 is indistinguishable from s_2.

Definition 6.4.4 (Bisimulation Relation)**.** Two state transition systems L and L' are in a bisimulation relation $\sim$, iff $L \rightsquigarrow L'$ and $L' \rightsquigarrow L$.

Theorem 1 (PHAST BTs Simulate SCTs)**.** *For any SCT $L = \langle P, \Lambda, E \rangle$, there exists at least one PHAST BT $\mathcal{T}$ such that $\mathcal{T} \rightsquigarrow L$.*

Proof. To show $\mathcal{T} \rightsquigarrow L$, we need to show that the world states resulting from $\mathcal{T}$'s interaction are indistinguishable from those resulting from L's interaction, i.e., $\mathcal{T}$ exhibits the same behavior as L. This can be shown using double induction on both the size of P and the number of ACs and IMs in each $p_i \in P$.

BC 0: we assume that L has one phase, $|P| = 1$ with $|p_1| = n$, i.e., the number of ACs and IMs together is n.

BC 1: When $n = 1$, the new world state s' changes by applying the single AC's auxiliary function f (or m for a single IM) to the current world state s and the user input $\sigma \in \mathbb{R}^6$. This applies for both L and $\mathcal{T}$ resulting from Algorithm 12 since $\mathcal{T}$ has only one leaf node wich applies f to s and σ. i.e., $s' = \mathcal{T}(\sigma, s) = L(\sigma, s) = f(\sigma, s)$.

IC 1: We show that when this holds for $|P| = 1$ and $|p_1| = n$, it holds for $|p_1| = n+1$ as well.

IH 1: Assume this holds when p_1 has k ACs. $p_1 = [\alpha_1, \ldots, \alpha_k]$ for some $k \in \mathbb{N}$. Where each α_i applies the auxiliary function f_i as shown in Equation 6.12. In the next induction step, we show that it would hold for $k+1$ ACs.

$$s'_k = \mathcal{T}(\sigma, s) = L(\sigma, s) = f_k(\sigma, f_{k-1}(\sigma, \ldots f_1(\sigma, s))) \tag{6.12}$$

IS 1: since L still has one phase, it would apply the auxiliary functions of each of the k AC in sequence shown in Equation 6.12. After that it applies the $k+1^{th}$ function, f_{k+1} so $s'_{L_{k+1}} = f_{k+1}(s'_k)$. Similarly, $\mathcal{T}$ has a single sequence node and $k+1$ leaf nodes, each is a SCAN η_i for $i \leq k+1$ corresponding to an α_i applying its respective f_i. The new world state $s'_{\mathcal{T}_{k+1}}$ is the result of applying the function f_{k+1} after the first k nodes. i.e., $s'_{k+1} = f_{k+1}(s'_k) = f_{k+1}(\sigma, f_k(\sigma, \ldots f_1(\sigma, s)))$

IC 0: We now show that if this holds when L has multiple phases, i.e., $|P| = z$ each with one or more ACs, i.e., $|p_i| \geq 1$ for all $p_i \in P$, then it also holds for $|P| = z+1$.

IH 0: we assume the above holds for some $|P| = z'$ where $|p_i| = k_i$ for all $p_i \in P$ and $i \leq z'$ and an arbitrary number of ACs k_i. i.e., $s'_L = s'_{\mathcal{T}}$. Each AC $\alpha_{p_i,j} \in p_i$ applies the auxiliary function $f_{p_i,j}$. As shown above, each phase calculates the next world state as seen in Equation 6.12. For multiple phases, this will be calculated as the piece-wise function shown in Equation 6.13.

$$s'_L = \begin{cases} f_{1,k_1}(\sigma, f_{1,k_1-1}(\sigma, \ldots f_{1,1}(\sigma, s)) \\ \quad \text{iff } s \vDash \lambda_{k_1} \wedge \exists_{p_x \in P} \langle p_x, \lambda_{k_1}, p_{k_1} \rangle \in E \\ \quad \vdots \\ f_{z',k_{z'}}(\sigma, f_{z',k_{z'}-1}(\sigma, \ldots f_{z',1}(\sigma, s)) \\ \quad \text{iff } s \vDash \lambda_{k_{z'}} \wedge \exists_{p_x \in P} \langle p_x, \lambda_{k_{z'}}, p_{k_{z'}} \rangle \in E \end{cases} \tag{6.13}$$

In this case, a PHAST BTs $\mathcal{T}$ generated using Algorithm 12 given L, has z' sequence nodes. Each node S_i has a δ-branch that checks the condition λ_i for all λ_i where $\langle p_x, \lambda_{k_i}, p_{k_i} \rangle \in E$. These are followed by the SCAN leaf nodes applying their respective auxiliary functions $f_1, \ldots, f_{k_i}$ in sequence. Therefore calculating the next state $s'_{\mathcal{T}}$ depends on which sequence node will get ticked, i.e., the first sequence node whose conditions hold as seen in Equation 6.14 where $des(\mathcal{T})$ is the set of descendants of $\mathcal{T}$. Due to the loop in Line 6 of Algorithm 12, $\mathcal{T}$ has a $\delta_{p'}(\lambda_{p_{z'}})$ branch iff p has an incoming edge with label $\lambda_{p_{z'}}$. i.e., $\forall_{p \in P} \exists_{p' \in P} \langle p', \lambda, p \rangle \leftrightarrow \delta_{p'}(\lambda_p) \in des(\mathcal{T})$. Therefore $s'_L = s'_{\mathcal{T}}$.

$$s'_{\mathcal{T}} = \begin{cases} f_{1,k_1}(\sigma, f_{1,k_1-1}(\sigma, \dots f_{1,1}(\sigma, s)) \\ \quad \text{iff } s \vDash \lambda_{p_1} \wedge \delta_{p'}(\lambda_{p_1}) \in des(\mathcal{T}) \\ \quad \vdots \\ f_{z',k_{z'}}(\sigma, f_{z',k_{z'}-1}(\sigma, \dots f_{z',1}(\sigma, s)) \\ \quad \text{iff } s \vDash \lambda_{p_{z'}} \wedge \delta_{p'}(\lambda_{p_{z'}}) \in des(\mathcal{T}) \end{cases} \tag{6.14}$$

IS 0: we show that if this holds for $|P| = z$ then it holds for $|P| = z+1$. With $z+1$ phases in L, the tree's second level has $z+1$ sequence nodes each with its respective k_i SCANs and conditions. L calculates $s'_{L^{z+1}}$ as $f_{z'+1,k_{z'+1}}(\sigma, \dots f_{z'+1,1}(\sigma, s))$ iff $s \vDash \lambda_{k_{z'+1}} \wedge \exists_{p_x \in P}\langle p_x, \lambda_{k_{z'+1}}, p_{k_{z'+1}}\rangle \in E$ otherwise, it is calculated as seen in Equation 6.13. The next state is calculated as $s'_{\mathcal{T}^{z+1}} = f_{z'+1,k}(\sigma, f_{z'+1,k-1}(\sigma, \dots f_{z'+1,1}(\sigma, s))$ iff $s \vDash \lambda_{p_{z'}+1} \wedge \delta_{p'}(\lambda_{p_{z'}+1}) \in des(\mathcal{T}^{z+1})$ otherwise, it is calculated as seen in Equation 6.14. Thus for $z+1$ phases in L, $s'_{L^{z+1}} = s'_{\mathcal{T}^{z+1}}$.

BC 2: Trivially, we can see that for a maximum of 1 AC per phase $p \in P$, $s'_L = s'_{\mathcal{T}}$, which is calculated as seen in Equation 6.15.

$$s'_L = s'_{\mathcal{T}} = \begin{cases} f_{1,1}(\sigma, s) \\ \quad \text{iff } s \vDash \lambda_{k_1} \wedge \exists_{p_x \in P}\langle p_x, \lambda_{k_1}, p_{k_1}\rangle \in E \\ \quad \vdots \\ f_{z,1}(\sigma, s) \\ \quad \text{iff } s \vDash \lambda_{k_z} \wedge \exists_{p_x \in P}\langle p_x, \lambda_{k_z}, p_{k_z}\rangle \in E \end{cases} \tag{6.15}$$

IH 2: We can assume that this holds for z phases and k ACs per phase where both the PHAST BT and L calculate the s' as seen in Equation 6.13.

IS 2: for $k+1$ ACs, the next state is calculated as seen in Equation 6.16.

$$s'_L = s'_{\mathcal{T}} = \begin{cases} f_{1,k_1+1}(\sigma, f_{1,k_1}(\sigma, \dots f_{1,1}(\sigma, s)) \\ \quad \text{iff } s \vDash \lambda_{p_1} \wedge \exists_{p_x \in P}\langle p_x, \lambda_{k_1}, p_{k_1}\rangle \in E \\ \quad \vdots \\ f_{z,k_z+1}(\sigma, f_{z,k_z}(\sigma, \dots f_{z,1}(\sigma, s)) \\ \quad \text{iff } s \vDash \lambda_{p_z} \wedge \exists_{p_x \in P}\langle p_x, \lambda_{k_z}, p_{k_z}\rangle \in E \end{cases} \tag{6.16}$$

□

Theorem 2 (SCTs Simulate PHAST BTs)**.** *For any PHAST BT $\mathcal{T}$, there exists at least one SCT L such that $L \rightsquigarrow \mathcal{T}$.*

Proof. The proof is analogous to that of Lemma 1 since Algorithm 12 constructs L by aggregating the auxiliary functions used in the SCANs adding them using the same order into the lists representing each phase. The algorithm also uses the δ-branches to create the labels on the phase transitions. It follows directly that in all the cases considered in the double induction $s'_L = s'_{\mathcal{T}}$. $\square$

From Theorems 1 and 2, we can see that for any SCTs L, there exists at least one PHAST BTs $\mathcal{T}$ such that $L \sim \mathcal{T}$.

6.4.7 Case Study

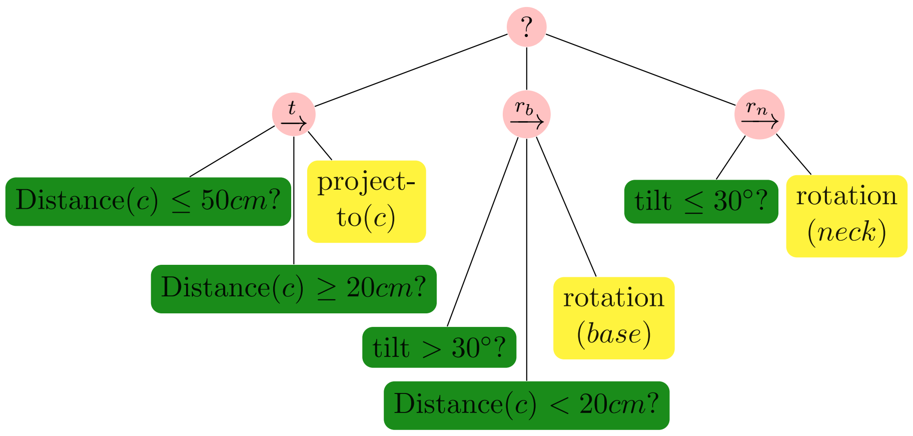


Figure 6.13: A PHAST BT representing the activity of pouring water from a bottle. The activity has 3 phases, translating the bottle t, rotating around the base b, and rotating around the neck n.

This section describes a use case for representing an Activity of Daily Living (ADL) using PHAST BTs. We describe the activity of pouring liquid from a bottle b to a cup c. This activity was chosen as an example as it was used as an example in previous work [Behery, 2016, Quere et al., 2020]. Here we also give examples of the input mapping functions that can be used in the SCANs. The activity is divided into 3 phases, translation t, base rotation r_b, and neck rotation r_n, as seen in Figure 6.13. Phase t is the translation to place the bottle at an adequate distance from the cup. In phases r_b, and r_n, we rotate the bottle around its base and neck, respectively. These are represented in Figure 6.13 as the sequence nodes $\xrightarrow{t}$, $\xrightarrow{r_b}$, and $\xrightarrow{r_n}$, respectively. The condition nodes measure the distance between the bottle and the cup as well as the bottle's tilt over the horizon.

In the first phase, the user input is projected on a straight line from the bottle to the cup, it starts when the bottle is 50 cm away from the cup till 20 cm and projects the input on the line between the bottle location l_b and the cup location l_c. When the user commands lead to a new location of the bottle (held in the gripper), l_b^u. We interpolate the line between the bottle location l_b and l_b^u as $\vec{l_b^u} = l_b - l_b^u$. Similarly, we interpolate a line between the locations of the bottle and cup, $\vec{L_{b,c}} = l_b - l_c$. After that, we project $\vec{l_b^u}$ on $\vec{L_{b,c}}$ using $project - to$ function that performs Equation 6.17

to calculate the new location of the bottle l'_b and uses it as the translation component in the new pose sent to the robot controller.

$$l'_b \; = \; l_b \; + \; (l_b^{\vec{u}} \; \cdot \; ||\vec{L_{b,c}}||) \; * \; ||\vec{L_{b,c}}|| \tag{6.17}$$

In the second and third phases, the rotation function seen in Equation 6.18 that rotates the given object around a given point on the bottle. We calculate the axis of rotation r as the cross product shown in Equation 6.18, where $\vec{L_1} = l_c - p_c$ is the line between the cup's location and it's pouring point p_c and $\vec{L_2} = l_b - p_c$ is the line between the bottle's location and p_c. We scale the user's input to get the rotation angle θ. This is done to tie the rotation speed and direction with the user's input allowing to speed up or slow down the operation by changing their input magnitude and to cancel it by giving commands in the opposite direction. The rotation matrix is shown in Equation 6.19, where c and s are $cos(\theta)$ and $sin(\theta)$ respectively.

$$\vec{r} = \begin{bmatrix} x \\ y \\ z \end{bmatrix} \; = \; L_1 \; \times \; L_2 \tag{6.18}$$

$$\begin{bmatrix} c + x^2(1-c) & xy(1-c) - zs & xz(1-c) + ys & 0 \\ xy(1-c) + zs & c + y^2(1-c) & yz(1-c) - zs & 0 \\ xz(1-c) - ys & yz(1-c) + xs & c + z^2(1-c) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{6.19}$$

The rotation function used in the SCANs in Figure 6.13 uses this rotation matrix for calculating the pose of the bottle in the next step.

Although these auxiliary functions seem rather complex to program, they can easily be encapsulated in predefined nodes as seen in several BT editors [Paxton et al., 2017, Gladiabots, 2019]. They offer composable predefined leaf nodes to create different behaviors using drag-and-drop interface.

6.5 Discussion

Traditional BT have shown great success in many robotics applications, from UAV flights, to domestic applications, to industrial tasks [Iovino et al., 2020, Colledanchise and Ögren, 2018]. Many approaches have also used LfD to learn BTs from the demonstrations of experts [Colledanchise et al., 2018, Iovino et al., 2021] or generated using classical planning [Martín et al., 2021] or a blend of learning and planning [Colledanchise et al., 2019] to name a few. However, some use cases still form a gap between BTs and alternatives such as FSMs. Other industrial use cases stood in the way of robotic automation all together because they relied on the human experience or because of a technological gap, i.e., the lack sensors or actuators that support full automation. Not to mention, that sometimes the lack of human expertise and funding prevents SMEs from the automation of their production plants, even despite the technological feasibility. Some tasks additionally endanger human workers due to the harsh or dangerous environments surrounding these tasks, e.g., nuclear decommissioning and in-orbit servicing.

A compromise for such use cases is hard coding tasks for the robot, HRC, or teleoperation. For hard coding, a human expert defines a trajectory for the robot arm, usually using the interface provided by the Original Equipment Manufacturer (OEM). This is done in use cases such as welding and cutting. HRC is used when a sub-task requires dexterity that a cheap robot arm cannot provide, e.g., installing cables during assembly, or when the task cannot be hard coded because of a highly dynamic environment, making the human expertise indispensable. Robots are teleoperated when the task is too dangerous for human workers to work alongside -or without- the robots but at the same time, difficult to hard code because it's executed in a dynamic environment or lacks models or sensors that can be used by the robot for decision making and motion planning. This is observed in use cases such as metal forming, robot surgery, and in-orbit servicing.

This chapter discussed three BT extensions that extend the scope of BTs usage, not only in robotics, but also in different applications.

The use of MIP [Ferguson et al., 1996] with BTs was demonstrated in [Rovida et al., 2017] but lacked the inherent readability and modularity of BTs because they transform the tree into an HTN at runtime making introspection more difficult. [Martín et al., 2021] creates plans using PDDL similar to Stanford Research Institute Problem Solver (STRIPS), then converts these into BTs for execution, this allows introspection and trouble shooting, but reduces the amount of control given to the human supervisors over the agent. In our work, DSNs allow a human supervisor to abstractly define the tasks, and the robot to ground them at runtime. This increases the flexibility offered to the supervisor by allowing the tree to be as detailed or as abstract as desired, while leaving room for runtime optimization.

Another extension proposed in this thesis allows BTs to integrate models of the human actions into the BT. This integration allows a BT to holistically model an assembly task, including the sub-tasks of the human workers, and consequently allows sharing a task model in a single artefact without loss of modularity. Using this kind of BTs combined with the CLIPS rule-based dispatcher, allows the robots to seamlessly switch tasks and start executing another BT instead of waiting for the human worker to do a task, increasing the throughput and the reducing idling time. This also increases the safety, since the task model is informed about the worker's tasks and can keep a distance, as well as trust, since the human is aware of the robot's current intent and action, thanks to the readability of BTs.

The third extension proposed here opens the door to a novel application of BTs, teleoperation tasks. Robot teleoperation approaches have used LTS, LfD, statistical methods, and hard-coded virtual fixtures. Despite their success, they are not modular, and sometimes not transparent enough to be trustworthy or reliable. BTs have demonstrated more usability, user friendliness, and modularity that allow a user with no programming experience to define a robot's actions. PHAST BTs enjoy these properties while preserving the representation power of LTS.

A BT can be even more dynamic and reactive by combining some of these extensions together. For example, we can combine the DSN with the HAN to build a tree without concretely assigning an order of the actions of the robot and the human. The tree would only indicate that the actions need to be performed sequentially as seen in Figure 6.14, giving more agency to both the human and the robot. At runtime, the robot can decide on the order of its actions depending on several run time aspects such as proximity from the human. The human can also perform the tasks in any order depending on their own preference without the need to explicitly notify the robot or modify its behavior. We can similarly combine DSNs with SCANs for a more dynamic execution of teleoperation actions that leave the human more autonomy to decide on the order of the action phases.

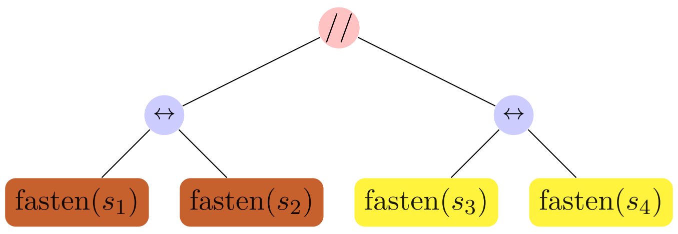


Figure 6.14: Combining assembly trees with dynamic sequence nodes from (highlighted in blue).

Although these extension require a PDDL model of each action node, they do not reduce the usability of BTs as a language. This is admissible in practice as it still reduces the mental workload on the supervisor compared to defining FSM to control the robot, due to the modularity of BTs. This modularity also gives more flexibility and control to the robot programmer compared to other approaches such as classical planning, LfD, and Machine Learning (ML) approaches, due to their transparency and readability.

Chapter 7

Conclusion and Outlook

To answer the call for the visions of Industry 4.0, RWTH Aachen University proposed an Internet of Production (IoP), which aims to connect all industrial entities across the globe. Similar to the Internet, the IoP needs standard structures, rules, and mechanisms to regulate the communication between these entities. Analogous to what the World Wide Web (WWW) did to the internet, the World Wide Lab (WWL) offers a collection of tools, communication standards, and methodologies that enable a network of this magnitude. This chapter concludes the thesis with a summary and an overview on future work. It first goes over the vision for the WWL, how it can be achieved in the long run and what can be done as first steps to move closer to that vision. After that, it discusses the proposed standard Digital Shadow (DS) representation for industrial robotics tasks and how they and the robots performing them can be integrated into the WWL. We also discuss tasks that rely on robotics but are difficult to automate as they involve human expertise. These tasks often involve a collaboration between the humans the robots, where some of the tasks require human agility and dexterity. Other times also involve a human worker that teleoperates the robot, when the environment is too dangerous or harsh for a human worker. Lastly, An outlook and recommendations for future work are presented.

7.1 Summary

The vision for the WWL is quite ambitious as the scope spans several scientific and engineering fields. To achieve this vision, the IoP brings together researchers across several institutes spanning these fields encouraging interdisciplinary collaborations. This thesis presented the vision of the WWL, which aims to connect industrial entities within and across company boarders. We further discuss use cases of industrial robotics in the IoP and their integration into the WWL.

The thesis additionally discussed a framework for shared autonomy teleoperation of robot arms as well as some proposed Behavior Tree (BT) extension that increase their usability and applicability in industrial robotics applications.

7.1.1 World Wide Lab

The WWL promises several advantages to the smart manufacturing landscape. It offers world-wide connectivity between not only production plants and labs, but also between the individual machines and components within them. The WWL requires standard representations for entities on different levels of production abstraction, from products, machines, and processes, to production plants, companies, and supply chains. However, this faces challenges from several perspectives. We summarize the technical aspects as follows:

- Standardized representations of DSs, that are flexible enough to represent the different levels of abstraction.
- Application Programming Interface (API) that allows the integration of different production partners into the WWL.
- Communication protocols, which allow storage, querying, and reuse of the information by different partners.
- Data-to-Knowledge pipelines that allow domain experts to extract knowledge from the available production data as well as integrate their domain knowledge and expertise with the data-driven solutions.

We additionally provide recommendations and a road map to reach the vision and exploit the full potential of the WWL. Initially, we need a Minimum Viable Product (MVP) that demonstrates some of the benefits on the small scale, starting from peer-to-peer networks that gradually grow and merge to increase the size of the network allowing the partners to reap more rewards. This is our proposed solution to the cold start problem facing the WWL and many recommender systems. In the latter case, the system allows users to rate some suggestions as a seed for the recommendations offered by the system. It then learns the patterns and the clusters related to the specific user. This is not possible the case of WWL because the connected parties are companies that need to see tangible results and advantages gained by joining the network.

7.1.2 Task Representations for Industrial Robots

This thesis focuses on some of the technical aspects in the industrial robotics domain. Namely, we focus on creating a standard DS to represent robotics tasks and activities in the WWL. The thesis proposes extending and standardizing BTs to represent robotics activities allowing them to cover more industrial use cases as well as to offer better Human-Robot Interaction (HRI) capabilities in terms of trust and reliability. We proposed several extensions to BTs to allow the robots to work collaboratively with humans.

We argue for the use of BTs as a standard DS in the WWL due to their modularity and reusability. They also show better scalability compared to other task representation approaches such as Finite State Machines (FSMs) and Hierarchical Task Networks (HTNs) and more control and usability compared other classical planning and learning approaches. BTs have additionally shown better usability for the end users. This has been demonstrated in both robotics and computer games, where players with no programming experience were able to program in-game agents.

Additionally in this thesis, we introduced several BT extensions to increase their applications in industrial scenarios. Using these extensions, BTs are able to handle use cases such as robot teleoperation and assembly based on Human-Robot Collaboration (HRC). BT-driven agents are also able to optimize their own BTs at run time. Additionally, the thesis proposed a framework to learn a robot operator's task models and transform them into BTs. These learned BTs can be composed into full task representations or be used for conformity checking.

7.1.3 Learning A BT-library of Human Action Models

This work proposed a framework for learning a library of BTs from the manual process execution by human workers. The framework first descritizes the teleoperator's velocity commands, which are logged during manual performance where the user commands are directly applied to the robot's end effector. After that we extract action patters from the user's discrete action sequences. The most common patterns are collected and compared using an extension Smith-Watermann [Smith et al., 1981], which is usually used for similar task in DNA sequencing. Additionally, we learn Maximally Specific Trees (MSTs) [Robertson and Watson, 2015] that represent these patterns and further add cross references between the trees for better readability and compression. We also learn Planning Domain Definition Language (PDDL) action models for these trees for higher safety and explainability. The presented framework is a human in the loop solution, which can either introduce shared autonomy assistance in teleoperation processes or verify the operator's direct control teleoperation in cases where shared autonomy is not desired by comparing the commands to the learned models.

7.1.4 BTs for Mixed Initiative Planning

Mixed Initiative Planning (MIP) traditionally allows a domain expert to collaboratively create plans with an autonomous agent [Ferguson et al., 1996]. This allows the human to abstractly define a plan, behavior, or task, which the robot can ground at run-time. Several approaches exist for defining and grounding such plans, including selection of target objects, ordering sub-tasks, as well as adding new sub-tasks at run-time. Using a Dynamic Sequence Node (DSN), we allow the behavior designer to create a BT describing the task and allow the robot to reorder some of the branches at runtime. DSNs scan their children and infer a PDDL action model for each one. The PDDL models are used to detect conflicts between the actions and branches. The children and their respective action models are then used to create a graph, which is used at run time to decide the order of execution of the children.

7.1.5 HRC Assembly

The second extension proposed in this thesis allows the BT to safely and efficiently handle HRC tasks. The tree executed by the robot contains both the robot's and the human coworker's tasks. We integrate these trees into the CLIPS rule based expert system [Wygant, 1989], which we use as a tree dispatcher. At startup time, the effects defined in the PDDL action models of the human tasks are translated into CLIPS rules. During execution, when the robot encounters a Human Action Node (HAN), the tree is paused, and the rule corresponding to that node is activated. In the mean time, the dispatcher can dispatch a new tree to the robot while the human works on their task. After the human had done their task and the post conditions of the actions are satisfied, the corresponding rule fires reactivating the tree. The advantages are three fold:

- Instead of idly waiting for during the execution of the human tasks, the robot is able to execute a new tree. This reduces the idling time and increases throughput.
- The trees representing the task contain both the robot's and the human's. This increases readability and maintainability, and facilitates debugging and reuse while maintaining modularity, reactivity, and safety of traditional BTs.
- Allowing the human to see the overall tree gives them an insight on the robot's intentions and next actions as well as the current state of execution. This increases the trust between the human and the robot, which is highly desirable in industrial scenarios.

7.1.6 BT for Robot Arm Teleoperation

Another use case that was not covered by traditional BTs is shared autonomy teleoperation. In this case, the BT should offer assistance to the robot's teleoperator by modifying the input signals such that they follow an abstractly defined trajectory to perform a task. Several approaches try to do this, but the challenge remains in allowing the users with no programming experience to define these tasks. Our work exploits the usability and readability of BTs using them as a no-code approach to define the activities. We extend BTs using Shared Control Action Nodes (SCANs) which perform the needed input modifications. In our work, we show that such BTs have the same representation power of the state of the art approaches, which rely on FSMs or similar structures for action representation. BTs have the added advantage of scalability and maintainability, allowing users to modify the tree or create a new one with ease.

7.1.7 BT Integration into the WWL

In our vision, these contributions allow BTs to cover more industrial use cases. They are also a step towards the use of BTs as a standard DS for representing robot tasks in the WWL. The modularity of these trees allow them to be used as a no-to-low code solution for programming robots by industry experts, who have no programming or robotics experience. The modularity further facilitates exchanging parts of the process know-how without having to reveal all the low level details or trade secrets. BTs are additionally flexible enough to allow the designer to choose a suitable level of granularity the conforms to their needs in terms of maintenance, usability, and reusability.

However, the work presented here leaves some open questions and uncovered some research gaps that still need to be bridged. The next section discusses the future steps in addressing the long term plans presented in this theses.

7.2 Future Work

For a standard DS that represents industrial robot actions and tasks, we need to empower BTs to cover more use cases. This means not only having the extensions that represent these use cases, but also having the interoperability between agents that use the extended trees. To this end, we need a BT library capable of unifying these extensions, allowing the behavior designer to choose between them and combine their functionalities at design time. To combine these functionalities, BTs need a unified theory backing them, which allows validation and verification. This is a step towards ensured safety and guaranteed compliance to industrial safety standards.

To further integrate BTs into the WWL, we need to investigate graph-based approaches for efficient storage, search, and retrieval of the BTs. Complemented by the modularity of BTs and their ability to encapsulate lower level logic (e.g., perception and control), this allows different parties to share know-how and software artefacts within and across the boundaries of a company or production plant while withholding their unique selling points to protect their economical edge.

Integrating this into a proof of concept implementation of the WWL provides an end-to-end communication spanning production lines and value chains across several different dimensions of production. It also creates Data-to-Knowledge pipelines allowing human workers, supervisors, and managers to extract actionable knowledge from processes and exchange information and knowledge feeding them back to the production process. After building the infrastructure of the WWL, it needs a proof of concept implementation that acts as a demonstrator and a starting point for a bigger global network of industrial partners.

Integrating the demonstrator into an academic community such as the Cluster of Excellence (CoE) allows exploiting the accompanying infrastructure to collect data. Such a demonstrator needs to allow a seamless integration of new partners along with their processes, materials, and production machines. We need to allow them to use the network and verify its technical and financial feasibility and usability to make an informed decision on its acceptance. Adopting the idea of the WWL at the earlier stages allows them to easily connect to other partners as the network grows. However, the WWL faces the cold start problem, which is a common issue in social platforms and recommender systems. This problem stems from the dependency cycle between growing the network and showcasing the network's potential and benefits. This dependency cycle should be incrementally broken by starting from a set of peer-to-peer networks that are later interconnected to others to gradually grow the network to a global scale.

Bibliography

[Abadi et al., 2015] Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., et al. (2015). Tensorflow: Large-scale machine learning on heterogeneous systems, software available from tensorflow.org. *https://www.tensorflow.org*. 73

[Almeida et al., 2020] Almeida, J. G., Silva, J., Batista, T., and Cavalcante, E. (2020). A linked data-based service for integrating heterogeneous data sources in smart cities. In *ICEIS (1)*, pages 205–212. 31

[Ames et al., 2019] Ames, A. D., Coogan, S., Egerstedt, M., Notomista, G., Sreenath, K., and Tabuada, P. (2019). Control barrier functions: Theory and applications. In *2019 18th European control conference (ECC)*, pages 3420–3431. IEEE. 47

[Anandan, 2019] Anandan, T. M. (2019). Ros-industrial for real-world solutions. https://www.automate.org/industry-insights/ros-industrial-for-real-world-solutions. Accessed: 2022-11-15. 10

[Ångström, 2022] Ångström, D. (2022). Genetic algorithms for optimizing behavior trees in air combat. Master's thesis, Linköping University. https://elib.dlr.de/105891/. 85

[Bach et al., 2016] Bach, M. P., Čeljo, A., and Zoroja, J. (2016). Technology acceptance model for business intelligence systems: Preliminary research. *Procedia Computer Science*, 100:995–1001. 36

[Bachmann et al., 2023] Bachmann, T., Eiberger, O., Eiband, T., Lay, F., Angsuratanawech, P., Rodriguez, I., Lehner, P., Stulp, F., and Nottensteiner, K. (2023). Task-specific reconfiguration of variableworkstations using automated planning of workcell layouts. In *ISR Europe 2023; 56th International Symposium on Robotics*, pages 250–257. VDE. 101

[Baier et al., 2022] Baier, R., Dammers, H., Mertens, A., Behery, M., Gossen, D., Nouduri, S., Pelzer, L., Shahidi, A., Trinh, M., Brecher, C., et al. (2022). A framework for the classification of human-robot interactions within the internet of production. In *International Conference on Human-Computer Interaction*, pages 427–454. Springer. 1, 5, 22, 81, 83, 95

[Bambach and Seuren, 2015] Bambach, M. and Seuren, S. (2015). On instabilities of force and grain size predictions in the simulation of multipass hot rolling processes. *Journal of Materials Processing Technology*, 216:95–113. 10, 38

[Bauernhansl et al., 2018] Bauernhansl, T., Hartleif, S., and Felix, T. (2018). The digital shadow of production–a concept for the effective and efficient information supply in dynamic industrial environments. *Procedia CIRP*, 72:69–74. 1, 2, 3, 38, 41, 42, 45, 46, 47

[Becerra et al., 2021] Becerra, M. A., Tobón, C., Castro-Ospina, A. E., and Peluffo-Ordóñez, D. H. (2021). Information quality assessment for data fusion systems. *Data*, 6. 45

[Becker et al., 2021] Becker, F., Bibow, P., Dalibor, M., Gannouni, A., Hahn, V., Hopmann, C., Jarke, M., Koren, I., Kröger, M., Lipp, J., Maibaum, J., Michael, J., Rumpe, B., Sapel, P., Schäfer, N., Schmitz, G. J., Schuh, G., and Wortmann, A. (2021). A conceptual model for digital shadows in industry and its application. In Ghose, A., Horkoff, J., Silva Souza, V. E., Parsons, J., and Evermann, J., editors, *Conceptual Modeling*, pages 271–281, Cham. Springer International Publishing. 35, 37, 38

[Behery, 2016] Behery, M. (2016). A knowledge-based activity representation for shared autonomy teleoperation of robotic arms. Master's thesis. https://elib.dlr.de/105891/. 105, 114

[Behery et al., 2023a] Behery, M., Brauner, P., Kluge-Wilkes, A., Baier, R., Mertens, A., Schmitt, R. H., Ziefle, M., and Lakemeyer, G. (2023a). Digital shadows for robotic assembly in the world wide lab. In *Proceedings of the 56TH CIRP International Conference on Manufacturing (CMS)*. 2, 6, 7, 27, 146

[Behery et al., 2023b] Behery, M., Brauner, P., Zhou, H. A., Uysal, M. S., Samsonov, V., Bellgardt, M., Brillowski, F., Brockhoff, T., Ghahfarokhi, A. F., Gleim, L., Gorissen, L. M., Grochowski, M., Henn, T., Iacomini, E., Kaster, T., Koren, I., Liebenberg, M., Reinsch, L., Tirpitz, L., Trinh, M., Posada-Moreno, A. F., Liehner, L., Schemmer, T., Vervier, L., Völker, M., Walderich, P., Zhang, S., Brecher, C., Schmitt, R., Decker, S., Gries, T., Häfner, C. L., Herty, M., Jarke, M., Kowalewski, S., Kuhlen, T. W., Schleifenbaum, J. H., Trimpe, S., van der Aalst and-Martina Ziefle, W. M., and Lakemeyer, G. (2023b). Actionable AI for the future of production. In *Internet of Production Fundamentals, Applications and Proceedings*. Springer. 1, 4, 7, 22, 27, 146

[Behery et al., 2023c] Behery, M., Deutsch, J., Trinh, M., Koetter, D., Brecher, C., and Lakemeyer, G. (2023c). Human robot collaborative assembly using behavior trees and dynamic tree dispatching. In *ISR Europe 2023; 56th International Symposium on Robotics*, pages 258–263. VDE. 6, 7, 83, 146, 148

[Behery et al., 2023d] Behery, M., Glawe, F., Koren, I., Ziefle, M., Lakemeyer, G., and Brauner, P. (2023d). Vision paper: Leveraging industrial big data – past, present, and future of the world wide lab. In *2023 IEEE International Conference on Big Data (BigData)*, pages 1308–1313, Los Alamitos, CA, USA. IEEE Computer Society. 1, 6, 7, 22, 27, 146

[Behery and Lakemeyer, 2022] Behery, M. and Lakemeyer, G. (2022). Motion Descriptors for Intention Recognition in Robot Teleoperation Tasks. In *Workshop on Prediction and Anticipation Reasoning in Human Robot Interaction*, Philadelphia. 5, 145

[Behery and Lakemeyer, 2023] Behery, M. and Lakemeyer, G. (2023). Digital shadows of safety for human robot collaboration in the world-wide lab. In *AAAI Spring Symposium on "HRI in Academia and Industry: Bridging the Gap"*. 2, 6, 7, 27, 146

[Behery et al., 2023e] Behery, M., Trinh, M., Brecher, C., and Lakemeyer, G. (2023e). Assistive robot teleoperation using behavior trees. In *Proceedings of Variable Autonomy for human-robot Teaming (VAT) workshop @ ACM/IEEE HRI 2023*. 5, 7, 83, 146

[Behery et al., 2023f] Behery, M., Trinh, M., Brecher, C., and Lakemeyer, G. (2023f). Self-optimizing agents using mixed initiative behavior trees. In *2023 IEEE/ACM 18th Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*, pages 97–103. IEEE. 5, 7, 83, 146

[Behery et al., 2021] Behery, M., Trinh, M., and Gerhard, L. (2021). Human action nodes for behavior trees. In *Proceedings of the Workshop Robotics for People (R4P): Perspectives on Interaction, Learning and Safety*, pages 13–14. 7, 83, 145

[Behery et al., 2020] Behery, M., Tschesche, M., Rudolph, F., Hirt, G., and Lakemeyer, G. (2020). Action discretization for robot arm teleoperation in open-die forging. In *Proceedings of the SMC 2020 conference*, pages 2100–2105. 5, 7, 22, 53, 145, 147

[Bello et al., 2016] Bello, I., Pham, H., Le, Q. V., Norouzi, M., and Bengio, S. (2016). Neural combinatorial optimization with reinforcement learning. *arXiv preprint arXiv:1611.09940*. 89

[Berger and Armstrong, 2022] Berger, S. and Armstrong, B. (2022). The Puzzle of the Missing Robots. *MIT Case Studies in Social and Ethical Responsibilities of Computing*, (Winter 2022). https://mit-serc.pubpub.org/pub/puzzle-of-missing-robots. 9, 51, 85

[Bergs et al., 2021] Bergs, T., Gierlings, S., Auerbach, T., Klink, A., Schraknepper, D., and Augspurger, T. (2021). The concept of digital twin and digital shadow in manufacturing. *Procedia CIRP*, 101:81–84. 9, 38

[Berners-Lee, 1989] Berners-Lee, T. J. (1989). Information management: A proposal. Technical report. 27

[Berners-Lee and Cailliau, 1990] Berners-Lee, T. J. and Cailliau, R. (1990). Worldwideweb: Proposal for a hypertext project. 27

[Berti et al., 2023] Berti, A., Koren, I., Adams, J. N., Park, G., Knopp, B., Graves, N., Rafiei, M., Liß, L., Unterberg, L. T. G., Zhang, Y., et al. (2023). Ocel (object-centric event log) 2.0 specification. 35

[Biber et al., 2024] Biber, A., Sharma, R., and Reisgen, U. (2024). Robotic welding system for adaptive process control in gas metal arc welding. *Welding in the World*, pages 1–10. 24

[Birkenkampf et al., 2014] Birkenkampf, P., Leidner, D., and Borst, C. (2014). A knowledge-driven shared autonomy human-robot interface for tablet computers. In *14th IEEE-RAS International Conference on Humanoid Robots (Humanoids)*, pages 152–159. IEEE. 84

[Bishop, 2006] Bishop, C. M. (2006). *Pattern recognition and machine learning*. springer. 61

[Bodenbenner et al., 2023] Bodenbenner, M., Pennekamp, J., Montavon, B., Wehrle, K., and Schmitt, R. H. (2023). Fair sensor ecosystem: Long-term (re-)usability of fair sensor data through contextualization. In *2023 IEEE 21st International Conference on Industrial Informatics (INDIN)*, pages 1–8. 32

[Braud et al., 2021] Braud, A., Fromentoux, G., Radier, B., and Le Grand, O. (2021). The Road to European Digital Sovereignty with Gaia-X and IDSA. *IEEE Network*, 35(2):4–5. 31

[Brauner et al., 2022] Brauner, P., Dalibor, M., Jarke, M., Kunze, I., Koren, I., Lakemeyer, G., Liebenberg, M., Michael, J., Pennekamp, J., Quix, C., Rumpe, B., van der Aalst, W., Wehrle, K., Wortmann, A., and Ziefle, M. (2022). A computer science perspective on digital transformation in production. *ACM Trans. Internet Things*, 3(2). 3, 21, 29, 31, 35, 37

[Brecher et al., 2023] Brecher, C., Padberg, M., Jarke, M., van der Aalst, W., and Schuh, G. (2023). The internet of production: Interdisciplinary visions and concepts for the production of tomorrow. In *Internet of Production: Fundamentals, Applications and Proceedings*, pages 1–12. Springer, Cham. 35

[Cai et al., 2021] Cai, Z., Li, M., Huang, W., and Yang, W. (2021). Bt expansion: A sound and complete algorithm for behavior planning of intelligent robots with behavior trees. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 6058–6065. 43, 46, 66, 84, 85

[Canny, 1986] Canny, J. (1986). A computational approach to edge detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-8(6):679–698. 55, 56

[Čertickỳ et al., 2018] Čertickỳ, M., Churchill, D., Kim, K.-J., Čertickỳ, M., and Kelly, R. (2018). Starcraft ai competitions, bots, and tournament manager software. *IEEE Transactions on Games*, 11(3):227–237. 71

[Colledanchise et al., 2019] Colledanchise, M., Almeida, D., and Ögren, P. (2019). Towards blended reactive planning and acting using behavior trees. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 8839–8845. IEEE. 19, 44, 45, 46, 66, 85, 92, 95, 115

[Colledanchise et al., 2016] Colledanchise, M., Marzinotto, A., Dimarogonas, D. V., and Oegren, P. (2016). The advantages of using behavior trees in mult-robot systems. In *Proceedings of ISR 2016: 47st International Symposium on Robotics*, pages 1–8. VDE. 18

[Colledanchise and Natale, 2018] Colledanchise, M. and Natale, L. (2018). Improving the parallel execution of behavior trees. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 7103–7110. IEEE. 18

[Colledanchise et al., 2018] Colledanchise, M., Parasuraman, R., and Ögren, P. (2018). Learning of behavior trees for autonomous agents. *IEEE Transactions on Games*, 11(2):183–189. 19, 45, 46, 85, 115

[Colledanchise and Ögren, 2018] Colledanchise, M. and Ögren, P. (2018). *Behavior Trees in Robotics and AI*. CRC Press. 2, 3, 14, 17, 41, 44, 46, 115

[Csiszar et al., 2017] Csiszar, A., Hoppe, M., Khader, S. A., and Verl, A. (2017). Behavior trees for task-level programming of industrial robots. In *Tagungsband des 2. Kongresses Montage Handhabung Industrieroboter*, pages 175–186. Springer. 18

[Curry, 2020] Curry, E. (2020). *Real-time linked dataspaces: Enabling data ecosystems for intelligent systems*. Springer Nature. 31

[Daabis, 2024] Daabis, R. (2024). Multi-agent reinforcment learning for pick-and-place tasks in the robocup logistics league. Master thesis, RWTH Aachen University. 5, 148

[Dammers et al., 2023] Dammers, H., Behery, M., Lakemeyer, G., and Gries, T. (2023). Supervised learning for robotic draping tasks in composite preforming. *Proceedings of the International Conference on Composite Materials (ICCM)*. 5, 23, 146, 147

[Dammers et al., 2022] Dammers, H., Vervier, L., Mittelviefhaus, L., Brauner, P., Ziefle, M., and Gries, T. (2022). Usability of human-robot interaction within textile production: Insights into the acceptance of different collaboration types. In *International Conference on Usability and User Experience*, volume 39, pages 213–223. AHFE. 2, 81, 84, 85

[Deutsch, 2023] Deutsch, J. (2023). Human action nodes for behavior trees. Bachelor thesis, RWTH Aachen University. 148

[Dey and Child, 2013] Dey, R. and Child, C. (2013). Ql-bt: Enhancing behaviour tree design and implementation with q-learning. In *2013 IEEE Conference on Computational Inteligence in Games (CIG)*, pages 1–8. IEEE. 19

[Ding and Goshtasby, 2001] Ding, L. and Goshtasby, A. (2001). On the canny edge detector. *Pattern recognition*, 34(3):721–725. 48

[Ehrenfeucht, 1961] Ehrenfeucht, A. (1961). An application of games to the completeness problem for formalized theories. *Fund. Math*, 49(129-141):13. 111

[El-Ariss et al., 2021] El-Ariss, W., Daher, N., and Elhajj, I. H. (2021). To resume or not to resume: A behavior tree extension. In *2021 American Control Conference (ACC)*, pages 770–776. IEEE. 18

[Emara, 2022] Emara, M. (2022). Dynamics moldeling of industrial robots using transformer networks. Master thesis, RWTH Aachen University. 147

[Ferguson et al., 1996] Ferguson, G., Allen, J. F., Miller, B. W., et al. (1996). Trains-95: Towards a mixed-initiative planning assistant. In *AIPS*, pages 70–77. 84, 116, 121

[Ferguson, 1973] Ferguson, T. S. (1973). A bayesian analysis of some non-parametric problems. *The annals of statistics*, pages 209–230. 60

[Fikes and Nilsson, 1971] Fikes, R. E. and Nilsson, N. J. (1971). Strips: A new approach to the application of theorem proving to problem solving. *Artificial intelligence*, 2(3-4):189–208. 67

[Forgy, 1988] Forgy, C. L. (1988). Rete: A fast algorithm for the many pattern/many object pattern match problem. In *Readings in Artificial Intelligence and Databases*, pages 547–559. Elsevier. 97

[French et al., 2019] French, K., Wu, S., Pan, T., Zhou, Z., and Jenkins, O. C. (2019). Learning behavior trees from demonstration. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 7791–7797. 19, 85

[Fu et al., 2016] Fu, Y., Qin, L., and Yin, Q. (2016). A reinforcement learning behavior tree framework for game ai. In *Proceedings of the 2016 International Conference on Economics, Social Science, Arts, Education and Management Engineering*, pages 573–579. Atlantis Press. 85

[Futterer, 2022] Futterer, M. (2022). Knowledge graphs as a dynamic semantic interface to the internet of production. Bachelor thesis, RWTH Aachen University. 148

[Gannouni, 2019] Gannouni, A. (2019). Neural combinatorial optimization for production planning. Master thesis, RWTH Aachen University. 147

[Gannouni et al., 2020] Gannouni, A., Samsonov, V., Behery, M., Meisen, T., and Lakemeyer, G. (2020). Neural combinatorial optimization for production scheduling with sequence-dependent setup waste. In *2020 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, pages 2640–2647. IEEE. 6, 145, 147

[Garmendia et al., 2022] Garmendia, A. I., Ceberio, J., and Mendiburu, A. (2022). Neural combinatorial optimization: a new player in the field. *arXiv preprint arXiv:2205.01356*. 89

[Ghahfarokhi et al., 2021] Ghahfarokhi, A. F., Park, G., Berti, A., and van der Aalst, W. M. (2021). OCEL: A standard for object-centric event logs. In *European Conference on Advances in Databases and Information Systems*, pages 169–175. Springer. 35

[Gladiabots, 2019] Gladiabots (2019). Gladiabots: AI Combat Arena. online. 42, 115

[Godbole and Sarawagi, 2004] Godbole, S. and Sarawagi, S. (2004). Discriminative methods for multi-labeled classification. In *Pacific-Asia conference on knowledge discovery and data mining*, pages 22–30. Springer. 62

[Goodrich et al., 2013] Goodrich, M. A., Crandall, J. W., and Barakova, E. (2013). Teleoperation and beyond for assistive humanoid robots. *Reviews of Human factors and ergonomics*, 9(1):175–226. 83, 105

[Görür and Rasmussen, 2010] Görür, D. and Rasmussen, C. (2010). Dirichlet process gaussian mixture models: Choice of the base distribution. *Journal of Computer Science and Technology*, 25(4):653–664. 60

[Hägele et al., 2016] Hägele, M., Nilsson, K., Pires, J. N., and Bischoff, R. (2016). Industrial robotics. *Springer handbook of robotics*, pages 1385–1422. 16

[Hart et al., 2014] Hart, S., Dinh, P., and Hambuchen, K. (2014). Affordance templates for shared robot control. In *Artificial Intelligence and Human-Robot Interaction, AAAI Fall Symposium Series, Arlington, VA. USA*. 84

[Hiscott et al., 2020] Hiscott, J., Alexandridi, M., Muscolini, M., Tassone, E., Palermo, E., Soultsioti, M., and Zevini, A. (2020). The global impact of the coronavirus pandemic. *Cytokine & growth factor reviews*, 53:1–9. 22

[Hopmann et al., 2023] Hopmann, C., Hirt, G., Schmitz, M., and Bailly, D. (2023). *Internet of Production: Challenges, Potentials, and Benefits for Production Processes due to Novel Methods in Digitalization*, pages 1–11. Springer, Cham. 21, 24, 29

[Horton et al., 2001] Horton, R. P., Buck, T., Waterson, P. E., and Clegg, C. W. (2001). Explaining intranet use with the technology acceptance model. *Journal of information technology*, 16:237–249. 36

[Howey et al., 2004] Howey, R., Long, D., and Fox, M. (2004). Val: Automatic plan validation, continuous effects and mixed initiative planning using pddl. In *16th IEEE International Conference on Tools with Artificial Intelligence*, pages 294–301. IEEE. 84

[Hu, 2013] Hu, S. J. (2013). Evolving paradigms of manufacturing: From mass production to mass customization and personalization. *Procedia Cirp*, 7:3–8. 22

[Industrial Value Chain Initiative, 2019] Industrial Value Chain Initiative (2019). What is ivi? – industrial value chain initiative. https://iv-i.org/wp/en/about-us/whatsivi/. accessed on 2023-09-01. 21

[Ioffe and Szegedy, 2015] Ioffe, S. and Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*. 61

[Iovino et al., 2020] Iovino, M., Scukins, E., Styrud, J., Ögren, P., and Smith, C. (2020). A survey of behavior trees in robotics and ai. 3, 9, 17, 18, 41, 42, 47, 109, 115

[Iovino et al., 2021] Iovino, M., Styrud, J., Falco, P., and Smith, C. (2021). Learning behavior trees with genetic programming in unpredictable environments. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4591–4597. IEEE. 19, 45, 46, 85, 115

[Isla, 2005] Isla, D. (2005). Gdc 2005 proceeding: Handling complexity in the halo 2 ai. 12

[ISO/TS 15066:2016, 2016] ISO/TS 15066:2016 (2016). Robots and robotic devices — Collaborative robots. Standard, International Organization for Standardization, Geneva, CH. 2

[Israelsen et al., 2014] Israelsen, J., Beall, M., Bareiss, D., Stuart, D., Keeney, E., and van den Berg, J. (2014). Automatic collision avoidance for manually tele-operated unmanned aerial vehicles. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6638–6643. IEEE. 105

[James et al., 2015] James, J., Weng, Y., Hart, S., Beeson, P., and Burridge, R. (2015). Prophetic goal-space planning for human-in-the-loop mobile manipulation. In *2015 IEEE-RAS 15th International Conference on Humanoid Robots (Humanoids)*, pages 1185–1192. IEEE. 84

[Jones et al., 2018] Jones, S., Studley, M., Hauert, S., and Winfield, A. (2018). Evolving behaviour trees for swarm robotics. In *Distributed Autonomous Robotic Systems*, pages 487–501. Springer. 46

[Juszczak et al., 2002] Juszczak, P., Tax, D., and Duin, R. P. (2002). Feature scaling in support vector data description. In *Proc. asci*, pages 95–102. Citeseer. 61

[Kagermann, 2015] Kagermann, H. (2015). Change through digitization – value creation in the age of industry 4.0. In *Management of Permanent Change*, pages 23–45. Springer Gabler, Wiesbaden, DE. 30

[Kah et al., 2015] Kah, P., Shrestha, M., Hiltunen, E., and Martikainen, J. (2015). Robotic arc welding sensors and programming in industrial applications. *International Journal of Mechanical and Materials Engineering*, 10:1–16. 24

[Kaster et al., 2023] Kaster, T., Rissom, J.-H., Gorissen, L., Walderich, P., Schneider, J.-N., and Hinke, C. (2023). Approach toward the application of mobile robots in laser materials processing. *Journal of Laser Applications*, 35(4). 24

[Keller, 1976] Keller, R. M. (1976). Formal verification of parallel programs. *Communications of the ACM*, 19(7):371–384. 105

[Kerber et al., 2018] Kerber, E., Heimig, T., Stumm, S., Oster, L., Brell-Cokcan, S., and Reisgen, U. (2018). Towards robotic fabrication in joining of steel. In *ISARC. Proceedings of the International Symposium on Automation and Robotics in Construction*, volume 35, pages 1–9. IAARC Publications. 24

[Khan et al., 2017] Khan, M., Wu, X., Xu, X., and Dou, W. (2017). Big data challenges and opportunities in the hype of industry 4.0. In *2017 IEEE International Conference on Communications (ICC)*, pages 1–6. IEEE. 32

[Kluge-Wilkes et al., 2023] Kluge-Wilkes, A., Baier, R., Kunze, I., Müller, A., Shahidi, A., Wolfschläger, D., Brecher, C., Corves, B., Hüsing, M., Nitsch, V., et al. (2023). Modular control and services to operate lineless mobile assembly systems. In *Internet of Production: Fundamentals, Applications and Proceedings*, pages 1–26. Springer. 22

[Kluge-Wilkes et al., 2022] Kluge-Wilkes, A., Gunaseelan, B., and Schmitt, R. H. (2022). Ontology-based task allocation for heterogeneous resources in line-less mobile assembly systems. In *MHI Colloquium*, pages 53–63. Springer International Publishing Cham. 32, 43

[Kordi, 2009] Kordi, M. T. (2009). *Entwicklung von Roboter-Endeffektoren zur automatisierten Herstellung textiler Preforms für Faserverbund-bauteile*. Produktentwicklung. Shaker, Aachen. 23

[Kötter, 2022] Kötter, D. (2022). Computer vision for safety in collaborative assembly. Master thesis, RWTH Aachen University. 147

[Krishnan et al., 2017] Krishnan, S., Garg, A., Patil, S., Lea, C., Hager, G., Abbeel, P., and Goldberg, K. (2017). Transition state clustering: Unsupervised surgical trajectory segmentation for robot learning. *The International journal of robotics research*, 36(13-14):1595–1618. 55, 59, 60

[Krömker, 2020] Krömker, L. (2020). Learning a behavior tree library from observations. Master thesis, RWTH Aachen University. 5, 7, 53, 64, 65, 69, 147

[Ladj et al., 2021] Ladj, A., Wang, Z., Meski, O., Belkadi, F., Ritou, M., and Da Cunha, C. (2021). A knowledge-based digital shadow for machining industry in a digital twin perspective. *Journal of Manufacturing Systems*, 58:168–179. 38

[Lee et al., 2017] Lee, I., Kim, D., Kang, S., and Lee, S. (2017). Ensemble deep learning for skeleton-based action recognition using temporal sliding lstm networks. In *IEEE International Conference on Computer Vision (ICCV)*, pages 1012–1020. 55, 59, 61

[Lee and Liu, 2003] Lee, W. S. and Liu, B. (2003). Learning with positive and unlabeled examples using weighted logistic regression. In *ICML*, volume 3, pages 448–455. 69

[Li et al., 2010] Li, W., Zhang, Z., and Liu, Z. (2010). Action recognition based on a bag of 3d points. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition-Workshops*, pages 9–14. 63

[Liebenberg, 2021] Liebenberg, M. (2021). *Autonomous Agents for the World Wide Lab: Artificial Intelligence in the Manufacturing Industry.* PhD thesis, RWTH Aachen University. 1, 3, 10, 29, 35, 38, 41

[Liebenberg and Jarke, 2020] Liebenberg, M. and Jarke, M. (2020). Information systems engineering with digital shadows: concept and case studies. In *International Conference on Advanced Information Systems Engineering*, pages 70–84. Springer. 23, 29, 35, 37, 38, 42

[Lin and Bercher, 2021] Lin, S. and Bercher, P. (2021). Change the world-how hard can that be? on the computational complexity of fixing planning models. In *IJCAI*, pages 4152–4159. 84

[Liu et al., 2023] Liu, Z., Wang, J., Wu, J., and Xing, A. (2023). Research and expectation on industrial welding robots. In *Proceedings of the 2023 International Conference on Mechatronics and Smart Systems.* EWA Publishing. 24

[Mann et al., 2020] Mann, S., Pennekamp, J., Brockhoff, T., Farhang, A., Pourbafrani, M., Oster, L., Uysal, M. S., Sharma, R., Reisgen, U., Wehrle, K., et al. (2020). Connected, digitalized welding production—secure, ubiquitous utilization of data across process layers. *Advanced Joining Processes*, pages 101–118. 35

[Marayong et al., 2003] Marayong, P., Li, M., Okamura, A. M., and Hager, G. D. (2003). Spatial motion constraints: Theory and demonstrations for robot guidance using virtual fixtures. In *2003 IEEE International Conference on Robotics and Automation (Cat. No. 03CH37422)*, volume 2, pages 1954–1959. IEEE. 105

[Martín et al., 2021] Martín, F., Ginés, J., Rodríguez, F. J., and Matellán, V. (2021). Plansys2: A planning system framework for ros2. In *IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2021, Prague, Czech Republic, September 27 - October 1, 2021.* IEEE. 19, 115, 116

[Mayr et al., 2021] Mayr, M., Chatzilygeroudis, K., Ahmad, F., Nardi, L., and Krueger, V. (2021). Learning of parameters in behavior trees for movement skills. In *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 7572–7579. IEEE. 19

[McDermott et al., 1998] McDermott, D., Ghallab, M., Howe, A., Knoblock, C., Ram, A., Veloso, M., Weld, D., and Wilkins, D. (1998). Pddl-the planning domain definition language. 66, 67, 86

[Merrill, 2019] Merrill, B. (2019). Building utility decisions into your existing behavior tree. In *Game AI Pro 360*, pages 81–90. CRC Press. 45, 86, 91

[Mertens et al., 2022] Mertens, A., Brauner, P., Baier, R., Brillowski, F., and et. al. (2022). Modelling human factors in cyber physical production systems by the integration of human digital shadows. In Michael, J., Pfeiffer, Jérôme, and Wortmann, A., editors, *Modellierung 2022 Satellite Events*, pages 147–149, Bonn. Gesellschaft für Informatik e.V. 35

[Mertens et al., 2021] Mertens, A., Pütz, S., Brauner, P., Brillowski, F., et al. (2021). Human digital shadow: Data-based modeling of users and usage in the internet of production. In *14th Int. Conf. on Human System Interaction (HSI)*, pages 1–8. IEEE. 32, 33, 35

[Methnani et al., 2021] Methnani, L., Aler Tubella, A., Dignum, V., and Theodorou, A. (2021). Let me take over: Variable autonomy for meaningful human control. *Frontiers in Artificial Intelligence*, page 133. 84

[Meyes et al., 2018] Meyes, R., Tercan, H., Thiele, T., Krämer, A., Heinisch, J., Liebenberg, M., Hirt, G., Hopmann, C., Lakemeyer, G., Meisen, T., et al. (2018). Interdisciplinary data driven production process analysis for the internet of production. *Procedia Manufacturing*, 26:1065–1076. 10

[Mining Safety, 2017] Mining Safety (2017). Advantages and disadvantages of robotics in welding. accessed on 06.06.2024. 24

[Mitcham, 2005] Mitcham, C. (2005). *Management.* Encyclopedia of science, technology, and ethics: Macmillan. 33

[Molnar, 2017] Molnar, M. F. (2017). The US advanced manufacturing initiative. *https://www.nist.gov/system/files/documents/2017/04/28/Molnar_091211.pdf*. accessed on 01.03.2024. 28

[Mower, 2005] Mower, J. P. (2005). Prep-mt: predictive rna editor for plant mitochondrial genes. *BMC bioinformatics*, 6:1–15. 75

[Muelling et al., 2017] Muelling, K., Venkatraman, A., Valois, J.-S., Downey, J. E., Weiss, J., Javdani, S., Hebert, M., Schwartz, A. B., Collinger, J. L., and Bagnell, J. A. (2017). Autonomy infused teleoperation with application to brain computer interface controlled manipulation. *Autonomous Robots*, 41(6):1401–1422. 105

[Myers and Miller, 1988] Myers, E. W. and Miller, W. (1988). Optimal alignments in linear space. *Bioinformatics*, 4(1):11–17. 65

[Myers et al., 2003] Myers, K. L., Jarvis, P., Tyson, M., and Wolverton, M. (2003). A mixed-initiative framework for robust plan sketching. In *ICAPS*, pages 256–266. 84

[Naumova, 2021] Naumova, B. (2021). Dynamic sequence nodes for increased behavior tree robustness. Bachelor thesis, RWTH Aachen University. 147

[Nguyen and LUU, 2020] Nguyen, X. T. and LUU, Q. K. (2020). Factors affecting adoption of industry 4.0 by small-and medium-sized enterprises: A case in ho chi minh city, vietnam. *The Journal of Asian Finance, Economics and Business*, 7(6):255–264. 85

[Niemuller et al., 2018] Niemuller, T., Hofmann, T., and Lakemeyer, G. (2018). Clips-based execution for pddl planners. In *Proceedings of the 28th International Conference on Automated Planning and Scheduling (ICAPS), WS on Integrated Planning, Acting and Execution.* 97, 98

[Otto and Jarke, 2019] Otto, B. and Jarke, M. (2019). Designing a multi-sided data platform - findings from the International Data Spaces case. *Electronic Markets*, 29(4):561–580. 31

[Otto and Zunke, 2015] Otto, M. and Zunke, R. (2015). Einsatzmöglichkeiten von Mensch-Roboter-Kooperationen und sensitiven Automatisierungslösungen: Zukunft der Arbeit–die neuen Roboter kommen. *http://www.blog-zukunft-der-arbeit.de/wp-content/uploads/2015/03/03_2015-11-25_IGMetall_Robotik-Fachtagung_OttoZunke.pdf*. accessed on 11.09.2024. 83

[Özkahraman and Ögren, 2020] Özkahraman, Ö. and Ögren, P. (2020). Combining control barrier functions and behavior trees for multi-agent underwater coverage missions. In *2020 59th IEEE Conference on Decision and Control (CDC)*, pages 5275–5282. IEEE. 47

[Pantelić et al., 2016] Pantelić, O., Pajić, A., and Nikolic, A. (2016). Analysis of available cloud computing models to support cloud adoption decision process in an enterprise. In *2016 6th International Conference on Computers Communications and Control (ICCCC)*, pages 135–139. IEEE. 37

[Paredes-Gualtor et al., 2017] Paredes-Gualtor, J., Moscoso-Zea, O., Saa, P., Sandoval, F., and Rodas, P. (2017). Unified cloud computing adoption framework. In *2017 International Conference on Information Systems and Computer Science (INCISCOS)*, pages 247–252. 36

[Paxton et al., 2017] Paxton, C., Hundt, A., Jonathan, F., Guerin, K., and Hager, G. D. (2017). Costar: Instructing collaborative robots with behavior trees and vision. In *2017 IEEE international conference on robotics and automation (ICRA)*, pages 564–571. IEEE. 115

[Pedregosa et al., 2011] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., et al. (2011). Scikit-learn: Machine learning in python. *the Journal of machine Learning research*, 12:2825–2830. 73

[Pennekamp et al., 2019] Pennekamp, J., Glebke, R., Henze, M., Meisen, T., Quix, C., Hai, R., Gleim, L., Niemietz, P., Rudack, M., Knape, S., Epple, A., Trauth, D., Vroomen, U., Bergs, T., Brecher, C., Bührig-Polaczek, A., Jarke, M., and Wehrle, K. (2019). Towards an infrastructure enabling the internet of production. In *2019 IEEE International Conference on Industrial Cyber Physical Systems (ICPS)*, pages 31–37. 30

[Perzylo et al., 2019] Perzylo, A., Rickert, M., Kahl, B., Somani, N., Lehmann, C., Kuss, A., Profanter, S., Beck, A. B., Haage, M., Hansen, M. R., et al. (2019). Smerobotics: Smart robots for flexible manufacturing. *IEEE Robotics & Automation Magazine*, 26(1):78–90. 9, 51, 85

[Piller and Nitsch, 2022] Piller, F. T. and Nitsch, V. (2022). *How Digital Shadows, New Forms of Human-Machine Collaboration, and Data-Driven Business Models Are Driving the Future of Industry 4.0: A Delphi Study*, chapter 1, pages 1–31. Springer International Publishing, Cham. 38

[Powell et al., 2022] Powell, D., Magnanini, M. C., Colledani, M., and Myklebust, O. (2022). Advancing zero defect manufacturing: A state-of-the-art perspective and future research directions. *Computers in Industry*, 136:103596. 29

[Prashanth et al., 2017] Prashanth, S., Subbaya, K., Nithin, K., and Sachhidananda, S. (2017). Fiber reinforced composites-a review. *J. Mater. Sci. Eng*, 6(03):2–6. 23

[Pütz et al., 2022] Pütz, S., Rick, V., Mertens, A., and Nitsch, V. (2022). Using IoT devices for sensor-based monitoring of employees' mental workload: Investigating managers' expectations and concerns. *Applied ergonomics*, 102:103739. 33, 35

[Qi and Tao, 2018] Qi, Q. and Tao, F. (2018). Digital twin and big data towards smart manufacturing and industry 4.0: 360 degree comparison. *Ieee Access*, 6:3585–3593. 9

[Quere et al., 2020] Quere, G., Hagengruber, A., Iskandar, M., Bustamante, S., Leidner, D., Stulp, F., and Vogel, J. (2020). Shared control templates for assistive robotics. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1956–1962. 105, 106, 108, 109, 114

[Quigley et al., 2009] Quigley, M., Conley, K., Gerkey, B., Faust, J., Foote, T., Leibs, J., Wheeler, R., Ng, A. Y., et al. (2009). Ros: an open-source robot operating system. In *ICRA workshop on open source software*, volume 3, page 5. Kobe, Japan. 10

[Raymond, 1978] Raymond, R. (1978). On closed world databases'. In *Logic and Data Bases*, pages 56–76. Plenum Press. 15

[Recker, 2014] Recker, D. (2014). *Entwicklung von schnellen Prozessmodellen und Optimierungsmoeglichkeiten fuer das Freiformschmieden.* PhD thesis, RWTH Aachen University, Aachen, Germany. Zsfassung in dt. und engl. Sprache; Zugl.: Aachen, Techn. Hochsch., Diss., 2014. 23

[Reynolds et al., 2009] Reynolds, D. A. et al. (2009). Gaussian mixture models. *Encyclopedia of biometrics*, 741(659-663). 48

[Robertson and Watson, 2015] Robertson, G. and Watson, I. (2015). Building behavior trees from observations in real-time strategy games. In *2015 International symposium on innovations in intelligent systems and applications (INISTA)*, pages 1–7. IEEE. 65, 66, 121

[Rohrer, 2022] Rohrer, T. (2022). Predictive object-centric process monitoring. Bachelor thesis, RWTH Aachen University. 6, 147

[Rovida et al., 2017] Rovida, F., Grossmann, B., and Krüger, V. (2017). Extended behavior trees for quick definition of flexible robotic tasks. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 6793–6800. IEEE. 18, 85, 116

[Rovida et al., 2018] Rovida, F., Wuthier, D., Grossmann, B., Fumagalli, M., and Krüger, V. (2018). Motion generators combined with behavior trees: A novel approach to skill modelling. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5964–5971. IEEE. 18

[Schmaus et al., 2020] Schmaus, P., Leidner, D., Krüger, T., Bayer, R., Pleintinger, B., Schiele, A., and Lii, N. Y. (2020). Knowledge driven orbit-to-ground teleoperation of a robot coworker. *IEEE Robotics and Automation Letters*, 5(1):143–150. 84

[Schmitz et al., 2021] Schmitz, M., Wiartalla, J., Gelfgren, M., Mann, S., Corves, B., and Hüsing, M. (2021). A robot-centered path-planning algorithm for multidirectional additive manufacturing for waam processes and pure object manipulation. *Applied Sciences*, 11(13):5759. 24

[Schneider et al., 2024] Schneider, J.-N., Gorißen, L., Kaster, T., Walderich, P., and Hinke, C. (2024). LSTM-based inverse dynamics learning for franka emika robot. In *2024 International Conference on Control, Automation and Diagnosis (ICCAD)*, pages 1–6. 23

[Schuh et al., 2021] Schuh, G., Gützlaff, A., Schmidhuber, M., and Maibaum, J. (2021). Development of digital shadows for production control. *ESSN: 2701-6277*. 38

[Seuren et al., 2012] Seuren, S., Willkomm, J., Bücker, H., Bambach, M., and Hirt, G. (2012). Sensitivity analysis of a force and microstructure model for plate rolling. In *Metal Forming 2012, proceedings of* *the 14th International Conference on Metal forming, September 16-19, 2012, Krakow, Poland*, pages 91–94. Wiley-VCH. 10, 38

[Seydibeyoglu et al., 2017] Seydibeyoglu, M. O., Mohanty, A. K., and Misra, M. (2017). *Fiber technology for fiber-reinforced composites*. Woodhead Publishing. 23

[Shahroudy et al., 2016] Shahroudy, A., Liu, J., Ng, T.-T., and Wang, G. (2016). Ntu rgb+d: A large scale dataset for 3d human activity analysis. In *IEEE Conference on Computer Vision and Pattern Recognition*. 63

[Sheridan, 1992] Sheridan, T. B. (1992). *Telerobotics, automation, and human supervisory control*. MIT press. 105

[Shirinzadeh et al., 2007] Shirinzadeh, B., Cassidy, G., Oetomo, D., Alici, G., and Ang Jr, M. H. (2007). Trajectory generation for open-contoured structures in robotic fibre placement. *Robotics and Computer-Integrated Manufacturing*, 23(4):380–394. 23

[Shirinzadeh et al., 2000] Shirinzadeh, B., Wei Foong, C., and Hui Tan, B. (2000). Robotic fibre placement process planning and control. *Assembly Automation*, 20(4):313–320. 23

[Shivakumar and Cohen, 2017] Shivakumar, S. and Cohen, G. (2017). *Securing Advanced Manufacturing in the United States: The Role of Manufacturing USA: Proceedings of a Workshop*. National Academies Press. 28

[Silva et al., 2003] Silva, J. N. C. P. d., Loureiro, A., Godinho, T., Ferreira, P., Fernando, B., and Morgado, J. (2003). Welding robots. 24

[Silva, 2015] Silva, P. (2015). Davis' technology acceptance model (TAM) (1989). *Information seeking behavior and technology adoption: Theories and trends*, pages 205–219. 36

[Smith et al., 1981] Smith, T. F., Waterman, M. S., et al. (1981). Identification of common molecular subsequences. *Journal of molecular biology*, 147(1):195–197. 64, 121

[Styrud et al., 2021] Styrud, J., Iovino, M., Norrlöf, M., Björkman, M., and Smith, C. (2021). Combining planning and learning of behavior trees for robotic assembly. *arXiv preprint arXiv:2103.09036*. 85

[Trinh et al., 2022a] Trinh, M., Behery, M., Emara, M., Lakemeyer, G., Storms, S., and Brecher, C. (2022a). Dynamics modeling of industrial robots using transformer networks. In *2022 Sixth IEEE International Conference on Robotic Computing (IRC)*, pages 164–171. IEEE. 5, 22, 145, 147

[Trinh et al., 2023a] Trinh, M., Dammers, H., Behery, M., Baier, R., Henn, T., Gossen, D., Corves, B., Kowalewski, S., Nitsch, V., Lakemeyer, G., Gries, T., and Brecher, C. (2023a). Safety of human-robot collaboration within the internet of production. *Proceedings of HCII 2023*. 6, 22, 146

[Trinh et al., 2024] Trinh, M., Faggian, G., Bottin, M., Rosati, G., Petrovic, O., and Brecher, C. (2024). Velocity- and load-dependent joint friction identification for a 6-dof industrial robot. In *2024 IEEE 18th International Conference on Advanced Motion Control (AMC)*, pages 1–6. 24

[Trinh et al., 2023b] Trinh, M., Kötter, D., Chu, A., Behery, M., Lakemeyer, G., Petrovic, O., and Brecher, C. (2023b). Safe and flexible planning of collaborative assembly processes using behavior trees and computer vision. *Intelligent Human Systems Integration (IHSI): Vol. 69, 2023*. 1, 2, 83

[Trinh et al., 2023c] Trinh, M., Kötter, D., Chu, A. C., Behery, M. B., Lakemeyer, G., Petrovic, O., and Brecher, C. (2023c). Safe and flexible planning of collaborative assembly processes using behavior trees and computer vision. *Intelligent Human Systems Integration (IHSI): Vol. 69, 2023*. 6, 27, 81, 146, 147

[Trinh et al., 2022b] Trinh, M., Schwiedernoch, R., Gründel, L., Storms, S., and Brecher, C. (2022b). Friction modeling for structured learning of robot dynamics. In *Congress of the German Academic Association for Production Technology*, pages 396–406. Springer. 24

[Trinh et al., 2023d] Trinh, M., Yadav, R., Schwiedernoch, R., Gründel, L., Petrovic, O., and Brecher, C. (2023d). Modeling of temperature-dependent joint friction in industrial robots using neural networks. In *2023 Seventh IEEE International Conference on Robotic Computing (IRC)*, pages 206–213. 24

[Tschesche, 2019] Tschesche, M. (2019). Extracting action sequences from open-die forging observations. 147

[Universal Robotics, 2022] Universal Robotics (2022). Robotic welding. accessed on 06.06.2024. 24

[Van Gemmeren, 2023] Van Gemmeren, M. (2023). Motion prediction in fibre-reinforced polymer preforming. Bachelor thesis, RWTH Aachen University. 147

[Vogel-Heuser et al., 2012] Vogel-Heuser, B., Bayrak, G., and Frank, U. (2012). *Forschungsfragen in" Produktionsautomatisierung der Zukunft": Diskussionspapier für die acatech Projektgruppe" ProCPS-Production CPS";[im Rahmen der Erarbeitung der Forschungsagenda Cyber-Physical Systems (agendaCPS)]*. acatech, Dt. Akad. der Technikwiss. 28

[Walderich et al., 2024] Walderich, P., Gorissen, L., Kaster, T., and Hinke, C. (2024). Sensor fusion for position estimation in robot-based laser material processing. *Journal of Laser Micro/Nanoengineering*, 19(1). 24

[Wang et al., 2020] Wang, T., Shi, D., and Yi, W. (2020). Extending behavior trees with market-based task allocation in dynamic environments. In *Proceedings of the 2020 4th International Symposium on Computer Science and Intelligent Control*, pages 1–8. 45, 46

[Waqas, 2023] Waqas, M. (2023). Mcfat rl: Multi-agent collaboration for assembly tasks using reinforcement learning. Master thesis, RWTH Aachen University. 148

[Watkins, 1989] Watkins, C. J. C. H. (1989). Learning from delayed rewards. 19

[White House Office Of The Press Secretary, 2011] White House Office Of The Press Secretary (2011). President obama launches advanced manufacturing partnership. *https://obamawhitehouse.archives.gov/the-press-office/2011/06/24/president-obama-launches-advanced-manufacturing-partnership*. accessed on 01.03.2024. 28

[Wilkinson et al., 2016] Wilkinson, M. D., Dumontier, M., Aalbersberg, I. J., Appleton, G., et al. (2016). The fair guiding principles for scientific data management and stewardship. *Scientific data*, 3(1):1–9. 32

[Wolf, 2023] Wolf, S. (2023). Learning from demonstrations for robot-based preforming for the production of fiber-reinforced polymers. Bachelor thesis, RWTH Aachen University. 5, 147

[Wolfgarten et al., 2019] Wolfgarten, M., Rosenstock, D., Rudolph, F., and Hirt, G. (2019). New approach for the optimization of pass-schedules in open-die forging. *International Journal of Material Forming*, 12(6):973–983. 23

[World Economic Forum and McKinsey & Company, 2021] World Economic Forum and McKinsey & Company (2021). Global lighthouse network: Reimagining operations for growth. White paper, World Economic Forum. 21

[Wu et al., 2015] Wu, C., Zhang, J., Savarese, S., and Saxena, A. (2015). Watch-n-patch: Unsupervised understanding of actions and relations. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4362–4370. 60

[Wu et al., 2017] Wu, H., Zhang, J., and Huang, K. (2017). Msc: A dataset for macro-management in starcraft ii. *arXiv preprint arXiv:1710.03131*. 71, 72

[Wübbeke et al., 2016] Wübbeke, J., Meissner, M., Zenglein, M. J., Ives, J., and Conrad, B. (2016). Made in China 2025. *Mercator Institute for China Studies. Papers on China*, 2(74):4. 28, 29

[Wygant, 1989] Wygant, R. M. (1989). CLIPS a powerful development and delivery expert system tool. *Computers & industrial engineering*, 17(1-4):546–549. 97, 122

[Xu et al., 2021] Xu, X., Lu, Y., Vogel-Heuser, B., and Wang, L. (2021). Industry 4.0 and industry 5.0—inception, conception and perception. *Journal of Manufacturing Systems*, 61:530–535. 28, 29

[Zhang and Zhou, 2007] Zhang, M.-L. and Zhou, Z.-H. (2007). ML-KNN: A lazy learning approach to multi-label learning. *Pattern Recognition*, 40(7):2038–2048. 62

[Zhang et al., 2017] Zhang, Q., Sun, L., Jiao, P., and Yin, Q. (2017). Combining behavior trees with maxq learning to facilitate cgfs behavior modeling. In *2017 4th International Conference on Systems and Informatics (ICSAI)*, pages 525–531. IEEE. 19

[Zhang et al., 2016] Zhang, X., Xie, W., Hoa, S. V., and Zeng, R. (2016). Design and analysis of collaborative automated fiber placement machine. *International Journal of Advanced Robotics and Automation*, 1(1):1–14. 23

[Zhao and Han, 2022] Zhao, B. and Han, X. (2022). Design of an improved process mining algorithm for manufacturing companies with industrial robots. In *China Intelligent Robotics Annual Conference*, pages 425–437. Springer. 81

[Zienkiewicz et al., 2005] Zienkiewicz, O. C., Taylor, R. L., and Zhu, J. Z. (2005). *The finite element method: its basis and fundamentals*. Elsevier. 10, 38

Statement of Originality

The work contributed in this thesis has been published and discussed over the past few years in several conferences, symposia, workshops, journals, and magazines. Several B.Sc. and M.Sc. theses, which I supervised throughout my time at the Knowledge-Based Systems Group (KBSG), contributed to the publications resulting from my work, the students were included in authoring the respective publications. Some of the ideas presented here have also been tuned over the course of several fruitful discussions with many colleagues at the KBSG and Internet of Production (IoP) as well as my supervisor, Gerhard Lakemeyer.

These publications and the work they involve were funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Germany's Excellence Strategy – EXC-2023 Internet of Production – 390621612. Additionally, this work is supported by the EU ICT-48 2020 project TAILOR (No. 952215).

Below is the list of publications that contributed to this thesis and my respective contributions to each.

1. [Behery et al., 2020]:

   I supervised the conceptualization, research, and development efforts and ran some of the experiments. I was also the main author of the paper.

2. [Gannouni et al., 2020]:

   I co-supervised the development and experimentation and was one of the three main authors of the paper.

3. [Behery et al., 2021]:

   I posed the problem as well as the proposed solution and was the main author of the paper.

4. [Behery and Lakemeyer, 2022]:

   I posed the problem, developed the theory, and programmed the solution to run the experiments.

5. [Trinh et al., 2022a]:

   I co-supervised the development and was one of the main authors of the paper.

6. [Trinh et al., 2023a]:

   I was one of the main contributors to the joint efforts that lead to this paper.

7. [Trinh et al., 2023c]:

   I contributed to the conceptualization and co-supervised the development and was one of the main authors of the paper.

8. [Dammers et al., 2023]:

   I co-supervised the development and experimentation effort. My work included the supervision of the technical and Artificial Intelligence (AI) oriented sides such as model selection and application.

9. [Behery and Lakemeyer, 2023]:

   I contributed the idea and was the main author of the paper.

10. [Behery et al., 2023d]:

    I was the main author of the paper. I co-organized the workshop that helped in some of the results reported in the paper. I also defined the road map discussed.

11. [Behery et al., 2023a]:

    I defined the road map presented in the paper. My contributions included, but were not limited to, the literature review and the conceptualization of the proposal in the paper.

12. [Behery et al., 2023b]:

    The book chapter reports on the collective efforts of the work stream I lead within the Cluster of Excellence (CoE): the IoP. I gave teh structure, and together with Philipp Brauner, I lead the writing effort.

13. [Behery et al., 2023c]:

    I proposed the problem and solution concepts. I also supervised the development and the creation of the proof of concept. I was also the main author of the paper.

14. [Behery et al., 2023e]:

    I defined the problem and proposed the solution and developed the necessary theory to build a proof of concept in the future.

15. [Behery et al., 2023f]:

    I defined the problem and its application in an industrial context and did the literature review and was the main author of the paper.

Below is the list of theses supervised during the time I worked at the KBSG. Some of these theses have contributed to the publications above while others were not published at the time of their completion. The results of the rest are either planned for publication at the time of writing, or are reported in this thesis.

1. [Gannouni, 2019]:

   I co-supervised the thesis from the computer science side. This thesis lead to the publication [Gannouni et al., 2020]

2. [Tschesche, 2019]:

   I supervised and guided the student and proposed some of the solutions involved. This thesis lead to the publication [Behery et al., 2020]

3. [Krömker, 2020]:

   I supervised and guided the student and proposed some of the solutions involved.

4. [Naumova, 2021]:

   I supervised and guided the student and proposed the solution.

5. [Rohrer, 2022]:

   I co-supervised the thesis from the applied AI side to guide the student through the model selection, training, and evaluation.

6. [Kötter, 2022]:

   I co-supervised the student and guided him through the applied AI side as well the writing. This thesis lead to the publication of [Trinh et al., 2023c]

7. [Emara, 2022]:

   I co-supervised the student and guided him in some of the technical aspects as well the writing. This thesis lead to the publication of [Trinh et al., 2022a]

8. [Wolf, 2023]:

   I co-supervised from the computer science and applied AI side. I helped the student in selecting the simulation framework and model selection.

9. [Van Gemmeren, 2023]:

   I co-supervised the thesis from the the computer science and applied AI side. I helped with the problem definition and solution as well as development of the concepts for the data collection. This lead to the publication of [Dammers et al., 2023].

10. [Deutsch, 2023]:

    I co-supervised the thesis from an applied robotics and AI perspective. I helped selecting the control framework used. This lead to the publication of [Behery et al., 2023c].

11. [Futterer, 2022]:

    I co-supervised the thesis from the side of applied robotics and applied computer science.

12. [Waqas, 2023]:

    I co-supervised the thesis from the computer science side.

13. [Daabis, 2024]:

    I supervised the student and guided her through the selection of the simulation and training frameworks and best practices for programming and training the models.