

**REPRESENTATIONS, TOOLS AND INTERFACES FOR IMPROVING EXPERT
DESIGN OF COLLABORATIVE HUMAN-ROBOT INTERACTIONS**

by

Andrew J. Schoen

A dissertation submitted in partial fulfillment of
the requirements for the degree of

Doctor of Philosophy

(Computer Sciences)

at the

UNIVERSITY OF WISCONSIN–MADISON

2023

Date of final oral examination: 12/19/23

The dissertation is approved by the following members of the Final Oral Committee:

- Bilge Mutlu, Professor, Computer Science, University of Wisconsin-Madison
- Aws Albarghouthi, Professor, Computer Science, University of Wisconsin-Madison
- Robert Radwin, Professor, Engineering, University of Wisconsin-Madison
- Allison Sauppé, Associate Professor, Computer Science, University of Wisconsin-La Crosse

© Copyright by Andrew J. Schoen 2023
All Rights Reserved

For Luna.

ACKNOWLEDGMENTS

If I were to list the individuals who helped me to get me where I am today, these acknowledgements may exceed the length of the following document. In lieu of an exhaustive list, I would like to highlight the following individuals who have made a significant impact on my life and career.

A colossal thanks goes to my advisor, Dr. Bilge Mutlu. It took some time for me to find my research path, but you had the patience and trust that I could figure it out when I didn't. You accepted me into this research community, and helped me grow into the researcher I am today. I am forever grateful for your guidance and support. Thanks also to Dr. Aws Albarghouthi, whose patience and constructive feedback showed me invaluable ways of problem solving. Another thanks goes to Dr. Allison Sauppé, whose work in many ways laid the trail for my own, and who brought thoughtful perspectives in our discussions. Finally, I would like to thank Dr. Robert Radwin, whose helpful and at time contrasting perspectives made our work better.

I would like to thank someone that encouraged me to start making software in research, Dr. David Perlman. I learned the basics of programming from you, and in so doing gave form to a passion I never realized was there all along. Dr. Stacey Schaefer similarly deserves recognition for letting me explore this passion in a neuroscience lab setting. I hope something I made still has some use. Another thanks goes to Dr. Nagesh Adluru, who had faith that I could be a computer scientist. Finally, I would like to thank Carolyn Zahn-Waxler and Morris Waxler, whose care, empathy, and thoughtfulness made me a better person and scholar.

My colleagues, mentees, and friends made an immeasurable impact on my Ph.D. journey. This work has been a collaborative effort, and I would not be here without individuals like Nathan White, Curt Henrichs, Daniel Rakita, Anna Konstant, Dakota Sullivan, and Amanda Siebert-Evenstone, who served as key collaborators on these projects. I would also like to thank the many undergraduate students who worked with me on this research, especially Ze Dong Zhang and Mathias Strohkirch. I hope I was able to teach you something useful, and that you enjoyed

the experience. Finally, a special thanks goes out to all my peers and friends that while not directly involved in this research were nevertheless a source of feedback, support, and encouragement. A special call-out to David Porfirio, who has been a true friend and colleague since the beginning of my Ph.D. journey, as well as Pragathi Praveena, whose advice and perspective is always valued.

Thanks to my sister, Alissa, who was always someone I could look up to. Gratitude also goes to Bryan, Adelyn, and Everly, whose addition to the family have brought us all so much joy. Finally, I would like to thank my parents, Amy and Mark, who have been the embodiment of unconditional support and love, as well as an unwavering source of encouragement as I pursued my education and career.

AUTHOR'S STATEMENT

The research within this dissertation was funded through National Science Foundation (NSF) awards 1426824, 1651129, 1822872 and 1925043.

The research within this dissertation represents the author's own work. However, the author frequently collaborated with other individuals — Nathan White, Curt Henrichs, Anna Konstant, Daniel Rakita, Dakota Sullivan, Ze Dong Zhang, Mathias Strohkirch, Amanda Siebert-Evenstone, Robert Radwin, David Shaffer, and Bilge Mutlu — who made substantial contributions. The contributions of these individuals are noted where relevant in the dissertation body. Especially substantial contributions with other researchers are elaborated upon here.

- *Curt Henrichs* has contributed significantly to the implementation and design of the *Authr* system, particularly with regards to the backend motion planning and server. He also contributed significantly to the original design of the *CoFrame* system, described in Chapter 4.
- *Nathan White* has contributed significantly to the design and implementation of the *CoFrame* system, described in Chapter 4, as well as the reinforcement learning implementation of *Allocobot*, described in Chapter 7.

Various components of this dissertation have already been published by the author in the following works: Schoen et al. (2020), Schoen et al. (2022), Schoen et al. (2023), and Schoen and Mutlu (2024).

CONTENTS

Contents iv

List of Figures vii

Abstract xiii

1 Introduction 1

- 1.1 *Motivation* 1
- 1.2 *Thesis Statement* 3
- 1.3 *Methodology* 3
- 1.4 *Users and Stakeholders* 4
- 1.5 *Contributions* 9
- 1.6 *Dissertation Overview* 10

2 Background 13

- 2.1 *Collaborative Robot Deployment* 13
- 2.2 *Program Requirements of Collaborative Robotics* 15
- 2.3 *Proximal Development and Scaffolding* 18
- 2.4 *Representations of Collaborative Interactions* 19

3 Authr 22

- 3.1 *Background* 24
- 3.2 *Technical Approach* 26
- 3.3 *Evaluation* 41
- 3.4 *Discussion* 58
- 3.5 *Chapter Summary* 61

4 CoFrame 63

- 4.1 *Background* 65
- 4.2 *Expert Model* 69

4.3	<i>System Design & Implementation</i>	72
4.4	<i>Case Studies</i>	82
4.5	<i>Chapter Summary</i>	84
5	Lively	87
5.1	<i>Background</i>	91
5.2	<i>System Design & Implementation</i>	95
5.3	<i>Case Studies</i>	106
5.4	<i>Chapter Summary</i>	110
6	OpenVP	112
6.1	<i>System Design & Implementation</i>	115
6.2	<i>Source Code and Usage</i>	122
6.3	<i>Chapter Summary</i>	122
7	Allocobot	124
7.1	<i>Background</i>	127
7.2	<i>System Design & Implementation</i>	129
7.3	<i>Future Work</i>	145
7.4	<i>Chapter Summary</i>	147
8	General Discussion	148
	References	158

LIST OF FIGURES

1.1	A table of the target user profiles and stakeholders considered in this dissertation. Images for target users and stakeholders are generated with AI.	11
1.2	A graphical representation of the systems presented in this dissertation, as well as their main approach in addressing the skills gap challenge for specifying collaborative robot behavior. This skills gap is represented as a chasm by which the designer/programmer must cross. Some systems reduce this gap by supporting their current models of work and knowledge, allowing for proximal advancement. Other systems aim to reduce this gap by providing tools that support the creation of more complex programs, thereby lowering the barrier for entry.	12
3.1	A description of the <i>Therbligs</i> implemented in <i>Authr</i> , including parameters, pre-conditions, and post-conditions.	32
3.2	For the technical evaluation, we constructed three manufacturing tasks: Kitting, Assembly, and Repair. For Kitting, <i>top</i> , a toy (cylinder) and a battery pack (cube) were moved to each container. In Assembly, <i>middle</i> , screws (grey cylinders) are placed in each of the four corners of a PCB and rotated, while two cables (pink cubes) are placed in the center. Finally Repair, <i>bottom</i> , features two faulty components (red cubes) being removed and replaced with new parts (green cubes).	35
3.3	We evaluated the <i>Authr</i> Allocation and Parallelization algorithms (blue) versus a <i>MO-DAE</i> planner (grey) with three different <i>Plans</i> (<i>Kitting</i> , <i>Assembly</i> , and <i>Repair</i>) on 4 different metrics (<i>Compute Time</i> , <i>Overall Plan Score</i> , <i>Overall Plan Time</i> , and <i>Overall Plan Cost</i>). Lower scores for all metrics are desirable.	37

3.4	The three modes in <i>Authr</i> . In <i>setup</i> , users first configure the workspace; <i>Destinations</i> are able to be added, deleted, and modified, and each <i>Agent</i> and <i>Thing</i> gets assigned an initial location in the scene. Moving into planning in the <i>Plan</i> Tab, <i>Tasks</i> are represented as containers for <i>Therbligs</i> and are ordered from left to right. Within each <i>Task</i> , <i>Therbligs</i> are ordered from top to bottom. <i>Therbligs</i> and <i>Tasks</i> are also configured. In <i>simulate</i> , after designing an interaction, users can simulate the actions of human and robot <i>Agents</i>	38
3.5	Participants viewed a video of an actor performing a simple kitting task and used <i>Authr</i> to translate it to a human-robot task.	43
3.6	USE and SUS scores from Evaluation 1.	46
3.7	For Evaluation 2, we constructed 2 comparable tasks, Cluster Sort and Ordered Sort. For Cluster Sort, top, participants organized blocks into clusters by type, and in Ordered Sort, bottom, participants organized blocks into a grid.	47
3.8	Codes generated using system states and nCoder.	51
3.9	Resulting Overall <i>Plan</i> Cost, Time, and Scores for Automatic and Manual procedures. Cost refers to the objective corresponding to effort or wear (depending on <i>Agent</i>), and Score refers to the overall score, based on the weighted Time and Cost. Lower scores for all measures are more desirable.	52
3.10	A comparison of the activity networks for Automatic Allocation (red) versus Manual (blue) conditions. Each network is displayed as both a network graph and box, indicating the mean and confidence intervals of the networks within the projected space.	57
4.1	In this chapter, we describe a system called <i>CoFrame</i> that integrates a set of <i>Expert Frames</i> in collaborative robotics, focusing on <i>Safety Concerns</i> , <i>Program Quality</i> , <i>Robot Performance</i> , and <i>Business Objectives</i> , to train operators in using, programming, and troubleshooting cobot applications.	64

- 4.2 The mapping of the themes from the *Expert Model* (Siebert-Evenstone et al., 2021) into each of the four *Expert Frames*: *Safety Concerns* (pink), *Program Quality* (blue), *Robot Performance* (yellow), and *Business Objectives* (green). Figure adapted from Siebert-Evenstone et al. (Siebert-Evenstone et al., 2021). 70
- 4.3 The four *Expert Frames* of *CoFrame*, and the relationships between them. As operators address concerns in each frame, they unlock other considerations. For example, only after addressing whether a Location or Waypoint is reachable (*Robot Performance*), do they address issues with the pose of the end effector. 73
- 4.4 The layout of the *CoFrame* interface. Operators can use the Program Editor tile (G) to construct their program, and can visualize the results in the Simulator tile (B). The Expert Frames tile (A) allows them to swap between different *Expert Frames* and view issues in each frame. When not viewing issues, the Contextual Information tile (C) shows relevant frame-related information, and when viewing issues also provides detailed information about the issue and suggestions for changes. Within the Program Editor (G) operators can drag blocks from the Block Drawer (D) into the Program Canvas (E). The Program Canvas contains the program (F) along with implemented skills. 74

- 4.5 Three case studies showing the process of evaluating feedback from the system and informing adjustments to the operator’s program. The gradient background of the figure denotes the switching between *Expert Frames* by the operator, from *Safety Concerns* (pink), *Program Quality* (blue), *Robot Performance* (orange), and *Business Objectives* (green). In Case Study 1, the operator begins by addressing a missing trajectory block (A), followed by filling in its parameters (B). The operator then addresses reachability concerns (C). They finish by addressing issues with robot collision (D). In Case Study 2, the operator begins by addressing joint speed issues and visualizes the speed (E). They transition to solving pinch point issues (F). They finish by addressing issues with the robot’s space usage (H). In Case Study 3, the operator begins by solving issues with uninitialized machine logic (I), then addressing problems with thing movement (J). They return to addressing a machine logic for a non-stopped machine (K). The operator finishes by viewing the robot’s cycle time (L). 81
- 5.1 We present *Lively* for real-time motion generation that balances task and communicative goals while maintaining feasibility. We provide three levels of interfaces to address varying use cases. The *Design Level* enables programming robots using a state-based approach. The *Develop Level* is configurable and portable, usable in applications such as ROS-based control and web-based simulation. The *Extend Level* supports the addition of new characteristics and goal specifications for greater customizability and extendability. 88
- 5.2 An early version of *LivelyStudio* that received feedback from animators and roboticists, which led to a redesigned 3D environment, more explicit state-based design process (states as graph nodes), and bundling of behavior attributes with specific goals and weights. 97

- 5.3 The layout of the *LivelyStudio* interface. From left to right, a *Simulator* window shows the robot in the currently selected state; the *Block Picker* allows dragging structural blocks like *States* or *Behavior Properties* like *Position Bounding*; the *State Editor* canvas that allows for states to be dragged around and modified. At the top-right, a menu that reveals a *Transition Widget*, which lists transitions from the current state, and a settings button that reveals a full URDF editor. 98
- 5.4 *LivelyStudio*'s set of *Behavior Properties* that match *Objective Functions* within *Lively*. Note, Velocity Minimization, Acceleration, and Jerk Minimization come in both joint-based and robot root variants, and while usable separately, are included within the *Smoothness* macro property. . 100
- 5.5 Solve times for the UR3e, Panda, and Pepper robots, with randomized locations of environmental colliders. Of note, speed is largely unaffected by shape count. 102
- 6.1 An example flow-based programming system designed with *OpenVP*, illustrating a simple logic about how a robot should behave if a patron enters a store. 112
- 6.2 Overview of *OpenVP*'s *Environment* layout, highlighting the four main sections: the *Drawer Selector*, where the active drawer can be set, the *Block Drawer*, where blocks in the current set can be selected from, the *Tab Selector*, where individual tabs can be added, removed, hidden, and edited, and finally the *Program Canvas*, where the program is visualized and edited. Full customization of the theme is possible, as shown in the light/dark modes. 113
- 6.3 Overview of block customization via their associated *TypeSpec* data. For brevity, some variants are not included, notably non-block *FieldInfo* structs, (e.g. *NumberFieldInfo*, *StringFieldInfo*, etc.). Also not shown is the *Extra* and *ConnectSpec* fields, discussed elsewhere. 117

6.4	An example of a Documentation section generated for an example <i>Function</i> block. The documentation automatically curates how that block is used in other blocks, and what blocks it uses. Additionally, the <i>Description</i> tab will render the textual markdown description from the <i>TypeSpec</i>	119
6.5	A small example flow-based program, illustrating the ability to draw connections between canvas-based nodes. Connectivity is configured within <i>BlockSpec</i> structs.	120
7.1	A mapping of the current components within <i>Allocobot</i> 's representation. Note, <i>Carry</i> , <i>Move</i> , <i>Travel</i> , and <i>Reach Primitives</i> are used internally within the algorithm, but are not specified explicitly, and therefore not shown. Overarching types are depicted as cards, where the solid header indicates the type. Any properties general across all types are listed first. Directly under outlined sub-type names are properties specific to that type or types. Rating is a simple Low/Medium/High categorical value.	132
7.2	A graphic showing the flow of the <i>Allocobot</i> process, highlighting the questions specific to each phase. The first phase is specifications, and are inputs that the stakeholder or integrator might provide to detail the job. The second and third phases reflect the two decision types (meta-parameters and simulation). The fourth phase represents the types of questions that can be answered after the process.	136
7.3	A description of the primitives utilized in the algorithm and the mapping onto different ergonomic models. Primitives in gray are used internally within the algorithm.	139

ABSTRACT

Collaborative robots (cobots) are a relatively new class of robots meant to be true collaborators with their human coworkers. This is in comparison to standard industrial robots, which due to safety considerations, must be sequestered away in cages from humans to avoid injuring them. Collaborative robots' promise is in their ability to assist human workers in tasks, thereby making the workers' jobs more efficient, enjoyable, and comfortable. However, despite this promise and a great deal of technical capability, cobots are not being utilized to their full potential. In cases where they don't get placed in storage, they frequently work independently away from any human coworkers, much like standard robots but without the protective cages around them.

This is partly due to a discrepancy between the requirements of true collaborative interaction design and the skill sets of the individuals who may be tasked with programming or designing their behaviors; the design of truly collaborative robot interactions is a fundamentally different challenge than the automation of conventional robots, incorporating aspects like resource dependencies, task allocation, motion design, human ergonomics, and more. However, many automation experts are familiar with traditional automation, but not interaction design. In contrast, individuals like motion designers and animators may provide useful guidance on behaviors like gesture and motion, but lack other skills that allow their applicable skills to translate easily to this domain.

The goal of this dissertation is to explore how to better support the effective adoption of cobots by developers. Put another way, how can we make it easier for developers accustomed to conventional robot programming - or otherwise relevant domains - to design behaviors and programs for true collaborative robot interactions that are beneficial, safe, and effective?

I propose that the answer to this question lies to some extent in the development of new tools and systems, which combined with the right program and behavior representations, as well as relevant feedback, optimization, verification, and synthesis techniques. These tools and systems can help developers by growing and

translating their domain knowledge into the interactive robot domain, while also improving and restructuring the programs they produce to better suit collaborative work. I consider multiple aspects of cobot programming, such as allocation, ergonomics, and motion. Finally, I conceptualize, design, and implement systems and tools meant to support these developer in each of these areas with the goal of empowering them to utilize collaborative robots more effectively.

1 INTRODUCTION

1.1 Motivation

Robots possess the potential to be effective collaborative partners in a variety of applications, and while they appear to have the capability to significantly improve the quality of human work (Pearce et al., 2018; Liu et al., 2022) and while they have been used successfully in certain situations (Alvarez-de-los Mozos and Renteria, 2017; Sauppé and Mutlu, 2015), a number of hurdles currently exist that prevent them from being used in both an effective and widespread manner, leading to a sizeable gap between the potential utility that collaborative robots (cobots) can theoretically provide, and which they demonstrate in research-based applications, and the types of usage seen in real-life scenarios (El Zaatari et al., 2019; Michaelis et al., 2020). Specifically, applications involving purportedly collaborative robots are often not collaborative at all, and instead involve the robot performing a task while the human is either not present or is performing a different task (Michaelis et al., 2020). This means that the current space of robotic usage is typically a binary one: either the robot is performing a task, or the human is performing a task, but rarely are they performing a task together. In other words, if the robot is capable of performing the task in isolation, this is typically handled as strict automation, and if the robot is not capable of this, it is relegated to a human-only task. Such a dichotomy prevents collaborative robots from providing a number of possible benefits, such as reduced ergonomic strain, increased productivity, or a wider range of suitable workers.

So why is this the case? Why are collaborative robots not being used in a collaborative manner? The answer, according to Michaelis et al. (2020) is that the engineers and programmers for these robots are not equipped with the knowledge to design these collaborative interactions. This is not to say that they are not capable of designing these interactions, but rather that they are not equipped with the knowledge and support to do so. The complex nature of the interaction that needs to be specified is not one that strict automation experts are accustomed to handling,

and the tools that are available to them are not designed to support them in doing so, being also designed with strict automation in mind.

As a term, "collaborative robotics" refers to interactions between humans and machines in service of achieving tasks in a shared space (Vicentini, 2021). Despite the utilitarian description, collaborative robotics very much sits at the intersection of task-focused application development and social robotics, the latter being another umbrella term encompassing interactions between humans and robots designed to interact in a social manner with the goal of achieving a variety of goals, including entertainment and the improvement of quality of life (Breazeal et al., 2016).

Truly dynamic collaborative interactions will likely depend on some degree of signalling and communication for both humanoid (Riek et al., 2010) and non-humanoid robots (Cha et al., 2016). Much like social interactions, they will involve a back-and-forth between agents, where signalling and coordination strategies can be deployed effectively to improve the result of the process (Mutlu et al., 2013; Huang et al., 2015; Andrist et al., 2018). Indeed, collaborative robots in manufacturing contexts have been shown to have a social impact (Saupé and Mutlu, 2015). Thus, it is important to take a wide perspective on the creation of these interactions, including but expanding on low-level details such as grasp planning (Marturi et al., 2019) to include more social aspects like gesture (Saupé and Mutlu, 2015), and high-level sub-task organization (Pearce et al., 2018). Furthermore, these types of interactions are likely to become more common in both the workplace (Galín and Meshcheryakov, 2019) and home (Wilson et al., 2019), and will require thoughtful design at all levels.

Therefore, unlike standard robot programming, collaborative design involves a back-and-forth between one or more agents, and involves differing capabilities, timings, sensory awareness, and communication styles between agents. While domain experts may have the knowledge of how to implement some of these different factors in a context-specific manner, this may not translate into an ability to coalesce this information into an effective human-robot collaborative program. What is needed going forward are two mutually dependent developments: tools that support domain expert in the design of truly collaborative interactions, and

representations of tasks, behaviors, and activities that are well-matched for the mental models that these experts may use, while also being amenable to transformation and reformulation by the aforementioned tools that utilize them.

1.2 Thesis Statement

My thesis is as follows: **Tools and systems which support domain experts during the programming process through the use of task and program representations, transformation, and relevant feedback can support the design of collaborative robot behaviors.** This dissertation aims to provide partial support to this thesis through a mixture of empirically validated systems and research-motivated systems. The precise representations and methods used herein are varied, but all place some degree of focus on considerations such as motion and space, program logic and interaction flow, and the valuation of tradeoffs during the design process. The approaches to these designs follow one of two paths: either the design focuses on the creation of a tool which, though the use of these aforementioned custom representations and methods provide a lower barrier to entry for the existing level of expertise of robot programmers, or the design focuses on the application of similar concepts in a design that supports learning and improvement of robot programmers' skills.

1.3 Methodology

To explore this thesis, I utilize a variety of approaches, generally including an initial exploration of the problem space, followed by the design and implementation of a system, and finally an evaluation of the system, in either a summative or formative manner. Evaluations may also include an exploration of case studies.

The first stage of this process is to explore the problem space. In some cases, this involves consulting existing literature, previous work, or performing on-site visits, and generally involves identifying a specific challenge faced by companies considering collaborative robots and brainstorming techniques that could be applied.

During this open-ended initial phase, these different challenges may be considered with a variety of possible solutions, factoring in metrics like suitability, feasibility, and utility.

The second stage involves the design and implementation of a candidate solution. This may involve the creation of an algorithm, specification (*e.g.*, a Domain-Specific Language), an interface or system, or most likely, a combination of all three. This stage is generally iterative, and may involve a number of prototypes, possibly involving the use of formative evaluations which guide development.

Finally, these systems may be evaluated using a mixture of approaches. In some cases, the algorithms designed can be evaluated using concrete metrics, such as algorithmic performance or output quality. In other cases, the systems are evaluated using a more qualitative approach, such as a user study or case study. In either case, the goal is to evaluate the system in a manner that is appropriate for the system itself, and to provide insight into the strengths and weaknesses of the system, as well as places with potential for future work.

As a final note, this dissertation utilizes the term “representation”, which in this context we define as both the set of elements that are used to describe a program, as well as the way in which these elements can be combined. This is inherently connected to the the concept of a user’s mental model, but is not necessarily the same. The extent to which the representation facilitates an effective mental model, or one conducive to the task at hand, is a key consideration in the designs we consider. Additionally, it is highly related to the concept of a Domain-Specific Language (DSL), but with at times relaxed definitions.

1.4 Users and Stakeholders

An important consideration in this space of collaborative robot programming is the specific target users we are focused on, since whatever tools, representations, and algorithms are used need to be designed for these individuals. In this dissertation, we define our overarching target user profile as a domain expert. A domain expert is someone who is highly knowledgeable in a certain field or subject matter. Further-

more, we refine this set of individuals to ones that are focused on the development of behaviors relevant to the design of collaborative robot behaviors. We can assume that the users of such systems are skilled, and motivated to engage in the programming experience in a professional capacity. What we don't assume, however, is that these are necessarily experts in collaborative robotics. In fact, given the current lack of expertise in this specific arena, we assume that these users are in fact not experts in collaborative robotics, but instead experts in fields for which their expertise provides unique value to the collaborative robotics space. Additionally, we are not targeting individuals like workers who interact with robots programmed by these experts as formal users, instead considering them stakeholders in such systems. In the section below, I outline a collection of user profiles that we consider in the following chapters and discuss the motivation for their inclusion. As a point of contrast, I also discuss various stakeholders - like workers - and discuss how these may overlap and differ from the set of target users. These profiles and stakeholders are summarized in Figure 1.1. As a final note, I do not explicitly refer to our target users as "end-users," like the ones described by Lieberman et al. (2006), due to the complex relationships between various stakeholders and users in this space, and the fact that the users we consider are not necessarily the end-users of the systems they create.

User Profile: Automation Expert

Automation experts are conventionally trained in domains such as industrial or mechanical engineering. They may have either an associate's degree or a bachelor's degree. Sufficiently large companies may hire such individuals internally to assist in automation, while smaller companies may contract this work out to "integrator" companies with automation experts on staff. Automation experts provide the initial implementation and maintenance of automation solutions within a company. This may involve converting current human-centric processes into automated ones, or modifying existing automation solutions for performance or quality improvement, or as needs or products change. In terms of technical skill, these individuals may

be familiar with rule-based automation systems, like ladder logic in Programmable Logic Controllers (PLCs), as well with more general-purpose programming languages like Python or C++. In the area of collaborative robotics, these experts may be attempting to increase automation in a task that is difficult to fully automate, likely due to some skill or knowledge needed, and which the worker provides. They therefore have the immediate goal of restructuring the process, such that the worker is still able to provide that skill, while the robot manages the rest. While simple low-interaction implementations are more easily accomplished, higher-level interactions involve more complex rules than something like ladder logic provides, and the interactivity and concurrency becomes more challenging in a strictly procedural or imperative form. Furthermore, these experts may not be familiar with the components of the interaction like safety, signaling, synchronization, and communication that were not critical in traditional automation. In this dissertation, we explore systems focused on this profile of user in Chapters 3, 4, and 7.

User Profile: Ergonomics Specialist

Ergonomics specialists are typically trained in domains such as human factors, and are generally trained in the analysis and improvement of workflows and processes in manufacturing and industrial settings, so as to improve the health and safety of workers. Sufficiently large companies may hire such individuals to continually monitor and improve the ergonomics of their processes, adapting to changes in products and assembly lines. They are familiar with varying methods of modeling human work, and analyzing workflows within these models for metrics like ergonomic strain, suggesting changes when necessary. If collaborative robots are introduced, such individuals may be asked to provide feedback about whether a given solution is within acceptable ergonomic and safety limits.

While these specialists have a highly technical background in the modeling and analysis of human work, and generally have deep knowledge about the nature of the work being done, they are generally not familiar with the specifics of robot programming, and therefore lack the technical knowledge to directly contribute to

the programming process. Methods for which they can leverage this ergonomic and task knowledge in assisting roboticists and automation experts is therefore of interest. In this dissertation, we explore systems focused on this profile of user in Chapters 3 and 7.

User Profile: Motion Designer

Unlike the previous two user profiles, this type of user is not necessarily tied to the field of manufacturing or robotics. Instead, these types of users may be trained in design and art, such as animation, game character design, or even dance, working in places like the entertainment or game industry. While education levels vary, these individuals bring with them a deep practical knowledge and artistic perspective on how motion can be used to convey information, intention, and more. Technical expertise may also vary, but many in animation and game design are familiar with tools like Maya (Autodesk, 2023), Blender (Blender Foundation, 2023), and/or game engines like Unreal (Epic Games, 2023). Animation systems generally employ a timeline and keyframe-based approach, whereby the positions and configurations of joints on character skeletons are specified at points in time, and the system interpolates between these points to create a smooth animation. Game engines can also be used, and generally provide more infrastructure to handle swapping between different animations, as well as interaction between these animations and physics-based environments. Such individuals may be asked to contribute to the design of certain robotics applications, and will be essential going forward to make collaborative robotics more acceptable and natural for their human partners.

However, despite their applicable knowledge and experience, there don't currently exist many avenues by which these individuals can contribute to the improvement of collaborative robot interactions. Additionally, while their knowledge of movement and motion is relevant, there exist certain differences and constraints in the way that robots function, versus the way characters may be animated. For example, in a digital medium, animation may employ a variety of techniques such as squash and stretch which can violate rules of rigid robotics, depending on the

specifics of the implementation. Furthermore, even game animations can take certain liberties with physics when transitioning between different animations, which may not be possible in a physical robot. In robotics, these motions may be more tightly coupled with real-time sensing and perception, which may be unfamiliar to animators. Therefore, methods by which their valuable insight can be incorporated, while also respecting the particulars of collaborative robotics, are needed.

In this dissertation, we explore a system focused on this profile of user in Chapter 5.

Stakeholders

In the interest of comparison, it is useful to consider how the above set of users may differ from the types of individuals who may benefit or find of interest the systems presented in this dissertation. In Figure 1.1 I present a set of three such stakeholders (business leads, workers, and roboticists), which I will briefly summarize here.

The first stakeholder we consider is the business lead, who may be a manager, director, or other individual who is responsible for the business decisions of a company. They may be trained or have experience in business and analytics, and in this context, may have interest in utilizing collaborative robots in order to improve process efficiency, product quality, reduce costs, or improve worker safety. However, these individuals need clear metrics and evidence that collaborative robots will provide these benefits in order to justify the investment. We consider a system in Chapter 7 which may be of interest to these stakeholders.

Workers are a natural stakeholder in the introduction of collaborative robots, as they are the ones who will be working alongside them. While they may not have the technical knowledge to contribute to the programming process, they may have valuable insight into the nature of the work, and what portions they would most like improved. After the introduction of a collaborative robot, these are the individuals who would benefit from an effective application, and suffer from a poor one. On the other hand, they may have concerns about the introduction of robots, and how it may affect their job security. Given the importance of this stakeholder,

and the multifaceted nature of their involvement, Chapters 3, 4, 5, and 7 all may be of interest to these stakeholders.

The final stakeholder we consider is a roboticist. This is a broad term, and not exclusive from the automation experts in the set of user profiles. Depending on the system and their role, they may also serve as stakeholders. This may occur in cases where the target user comes from a design background, and the result of their work is a specification that needs to be implemented. Therefore, this role serves as a reminder that in various circumstances, the identities of stakeholders and users can be more complex and nuanced. Roboticists can come from backgrounds such as computer science and engineering, and working in either industry or academic settings. Their jobs may involve the technical implementation of robotic systems, including the design of sub-systems, as well as the integration of these subsystems within architectures such as ROS (Robot Operating System) (Quigley et al., 2009). As such, clear descriptions of what the intended systems may be, how those descriptions translate to implementation, as well as the ability to organize these systems in coherent ways, are necessary. Chapters 5, 6, and 7 may be of interest to these stakeholders.

1.5 Contributions




The contributions of this dissertation include the design of algorithms and program representations that operate upon them for the specification of collaborative robotic programming, as well as the implementation of systems that utilize these representations and algorithms to support the design of collaborative robotic programs by a variety of domain experts. The overarching goal in supporting this design process is to address the current gap between the capabilities that collaborative robots have, and the ability of domain experts (e.g. automation experts) to utilize them effectively.

In some cases, this involves designing representations, like DSLs, that are well-suited for the domain experts' current mental models in order to support knowledge gain. In other cases, this involves considering how to make more complex represen-

tations more accessible to these experts at their current levels through feedback and automated processing. Usually, elements of both are present. Addressing these approaches are systems like *CoFrame*, which uses a variety of education-focused methods to improve the the types of collaborative programs that current engineers can produce. *Lively*, another tool created for non-robotics experts, focuses on a method of motion specification that prioritizes a workflow adjacent to the current workflows of animators, while incorporating specific improvements meant to support dynamic, collaborative activities. *Authr* is a system which by using specific action primitives, supports the creation of more complex collaborative programs, while maintaining simplified program representations. Finally, *Allocobot* is an in-progress system which improves on *Authr's* approach by considering a greater range of action primitives, and by incorporating a more robust transformative process.

1.6 Dissertation Overview

This dissertation is organized as follows: Chapter 2 provides a background on collaborative robotics, including a discussion of current collaborative robot deployment, program design in this space, proximal development, and representations for collaborative interactions. Chapters 3-6 include the systems and tools that were designed as part of the work within this dissertation by myself and collaborators. Chapter 7 introduces ongoing work, *Allocobot*, that will build on many of these prior works. Finally, Chapter 8 concludes the dissertation with a discussion of the contributions of this work, as well as a discussion of future work.

User Profiles				
	Description	Goals	Challenges	Projects
 Automation Expert	Trained in domains such as industrial or mechanical engineering with an associate's degree or a bachelor's degree. May be direct employees or contracted.	Translate human-only processes into collaborative interactions, balancing contributions of humans and robots, and managing tradeoffs.	Higher-level interactions require more flexible logic, more subsystems, and management of concurrency and synchronization, as well as domain knowledge outside their own.	<div style="background-color: #f08080; border-radius: 10px; padding: 2px 10px; display: inline-block;">Authr</div> <div style="background-color: #f0c080; border-radius: 10px; padding: 2px 10px; display: inline-block;">CoFrame</div> <div style="background-color: #c080f0; border-radius: 10px; padding: 2px 10px; display: inline-block;">Allocobot</div>
 Ergonomics Specialist	Trained in domains such as human factors. Familiar with the analysis of workflows for their physical impact on human workers.	May serve as a consultant that addresses areas for improvement in human processes, possibly through the introduction of collaborative robots.	Lack the technical know-how to practically implement robotic behaviors, infrastructure not present to allow their input in the programming process.	<div style="background-color: #f08080; border-radius: 10px; padding: 2px 10px; display: inline-block;">Authr</div> <div style="background-color: #c080f0; border-radius: 10px; padding: 2px 10px; display: inline-block;">Allocobot</div>
 Motion Designer	Trained in domains such as animation, and game character design. Familiar with tools for specifying keyframe-based behavior, and know about motion legibility and use.	Improve the quality of interaction, as well as reception of robot by workers, through motion design.	Experience with real-time systems, and how they may execute on physical robots is limited. May not have formal programming experience.	<div style="background-color: #80c0f0; border-radius: 10px; padding: 2px 10px; display: inline-block;">Lively</div>




Stakeholder Profiles				
	Description	Goals	Challenges	Projects
 Business Lead	Trained in business and analytics, and focused on improvements to product value, process efficiency, and worker satisfaction and safety.	Making decisions and business plans about whether and how to introduce collaborative robots in their companies. Seek clear metrics illustrating value.	Tradeoffs for product production and quality, worker recruitment, retention, and liability are complex and mutually dependent. Cobot value may not be clear until late in development.	<div style="background-color: #c080f0; border-radius: 10px; padding: 2px 10px; display: inline-block;">Allocobot</div>
 Worker	May be highly skilled in particular processes and workflows, but not necessarily through formal education.	Vested interest in maintaining occupation, but may desire improvements to workflows that benefit them.	Poor implementations of interactions may be unsafe or ineffective. Traditional automation may displace jobs.	<div style="background-color: #f08080; border-radius: 10px; padding: 2px 10px; display: inline-block;">Authr</div> <div style="background-color: #f0c080; border-radius: 10px; padding: 2px 10px; display: inline-block;">CoFrame</div> <div style="background-color: #80c0f0; border-radius: 10px; padding: 2px 10px; display: inline-block;">Lively</div> <div style="background-color: #c080f0; border-radius: 10px; padding: 2px 10px; display: inline-block;">Allocobot</div>
 Robotician	May have backgrounds in computer science and/or engineering, focused on robotics. May be working in an academic or industry setting.	Tasked with the technical challenge of implementing behaviors within robotics technologies such as ROS (Robot Operating System).	Systems are constructed as amalgamations of subsystems, which may need synthesis and organization for effective use. Clear descriptions of intended systems are also needed.	<div style="background-color: #80c0f0; border-radius: 10px; padding: 2px 10px; display: inline-block;">Lively</div> <div style="background-color: #80f0c0; border-radius: 10px; padding: 2px 10px; display: inline-block;">OpenVP</div> <div style="background-color: #c080f0; border-radius: 10px; padding: 2px 10px; display: inline-block;">Allocobot</div>

Figure 1.1: A table of the target user profiles and stakeholders considered in this dissertation. Images for target users and stakeholders are generated with AI.

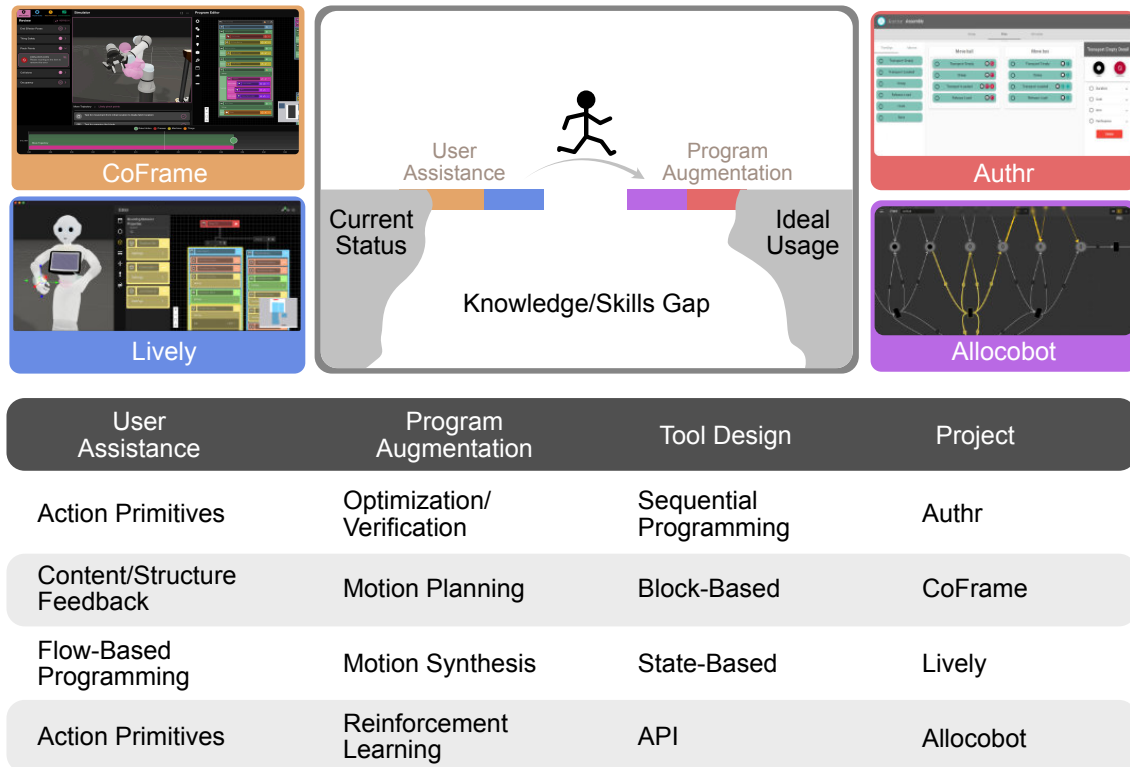


Figure 1.2: A graphical representation of the systems presented in this dissertation, as well as their main approach in addressing the skills gap challenge for specifying collaborative robot behavior. This skills gap is represented as a chasm by which the designer/programmer must cross. Some systems reduce this gap by supporting their current models of work and knowledge, allowing for proximal advancement. Other systems aim to reduce this gap by providing tools that support the creation of more complex programs, thereby lowering the barrier for entry.

2 BACKGROUND

2.1 Collaborative Robot Deployment

Collaborative robots, commonly called “cobots”, are a class of robots focused on providing assistance to human workers in a shared workspace as full coworkers (Fanuc, 2023; Kuka, 2023; Universal Robots, 2023). Features such as sensing, compliance, a smaller form-factor, and safety mechanisms that allow them to work in close proximity to humans usually separate these types of robots from their non-collaborative peers, and are intended to work alongside humans, rather than in isolation (Javaid et al., 2022). The premise therefore is to provide assistance to humans in a way that is safe and comfortable. According to Grand View Research (2023), they are also growing in popularity, with the cobot market expected to expand at a compound annual rate of 29.9% in the U.S. and 32.0% between 2023 and 2030 globally. They reflect a push in industry referred to as “Industry 4.0,” characterized by an increased usage of technology and automation, as well as features like artificial intelligence (AI) and the Internet of Things (IoT), and of course, robotics (Ortiz, 2020; Le et al., 2020).

Physically, most archetypical cobots are single-arm manipulators, intended to be mounted onto either a mobile base or a fixed structure. Such robots include devices such as Universal Robots’ UR-Series (Universal Robots, 2023), KUKA’s LBR-Series (Kuka, 2023), and FANUC’s CRX-Series (Fanuc, 2023). These robots are typically designed to be used in a wide variety of applications, and are often used in manufacturing (especially in small and midsize enterprises), logistics, and healthcare (Universal Robots, 2023; Kuka, 2023; Fanuc, 2023). However, interest in more humanoid robots is also growing, such as Agility’s “Digit” robot (Agility Robotics, 2023) and Sanctuary AI’s “Phoenix” robot (Sanctuary AI, 2023), where the goal is to produce robots that are more capable of performing human-like tasks in more human-like manners. Other solutions blur the line between humanoid or animal-like robots and collaborative robots, such as Rethink’s “Sawyer”, which features a screen with facial features (Rethink Robotics, 2023), and Boston Dynamics’

“Spot” robot, which features a quadrupedal design and manipulator arm (Boston Dynamics, 2023). While these and the humanoid robots represent a class of robots less clearly aligned with the term “cobot”, they nevertheless feature many of the same characteristics and design goals.

Cobots are primarily meant to be deployed in places like Small-Medium Enterprises (SMEs) and manufacturing settings (Schnell, 2021; Ortiz, 2020; Le et al., 2020). For example, current cobot usage includes tasks such as palletization (assembling groups of items from other groups), pick-and-place (retrieving objects and delivering them to a certain area), and applications of glues, adhesives, and coatings (Kakade et al., 2023).

However, their deployment lags behind their potential (Michaelis et al., 2020; Berger and Armstrong, 2022). To this point, Berger and Armstrong (2022) compare the capabilities of cobots to their non-collaborative peers. These non-collaborative robots are generally inflexible, focused on single, predictable tasks. They are generally regarded as considered unsafe, and therefore must be sequestered behind safety cages. That being said, it is relatively straightforward for integrators to program certain tasks, after which only routine maintenance and minor updates may be needed. In contrast, cobots are designed to be safe around people, and are sold as flexible solutions. Despite this, as Michaelis et al. (2020) notes, cobots are often used in the same way as their non-collaborative peers. Berger and Armstrong (2022) and Michaelis et al. (2020) attribute this to the fact that the selling point of cobots – their flexibility and collaborative potential – makes their usage more complex. Reprogramming in diverse product mix environments requires not just an initial programming, but likely routine reprogramming and adjustments. Furthermore, as Berger and Armstrong (2022) note, many companies are reluctant to welcome integrators into their companies on such a regular basis, on account of fears that this may lead to a loss of proprietary information. In such cases, this leaves the task of programming cobots to inside talent, which do not typically have the level of expertise to design and implement complex cobot programs. To understand why this is the case, we must also understand why cobot programming is not as straightforward as their non-collaborative counterparts.

2.2 Program Requirements of Collaborative Robotics

Collaborative robots present unique challenges to programming and behavior specification. These challenges are rooted in the fact that cobots are designed to work alongside humans, and therefore must be able to adapt to the dynamic and unpredictable nature of human behavior. This is in contrast to non-collaborative robots, which are typically designed to work in isolation, and therefore can be programmed to perform a single task in a predictable environment.

To begin, all collaborations are not the same. A variety of different metrics have been used to categorize collaborative work, such as the scheme defined by Phillips et al. (2016). This scheme defines three different levels of collaborative interdependence, specifically *Pooled Interdependence*, *Sequential Interdependence*, and *Reciprocal Interdependence*. In *Pooled Interdependence*, each individual works independently with minimal direct interaction between them. The example given in their paper is the work of a robotic vacuum, which independently performs its task with only minimal intervention (e.g. in cases where the robot gets stuck). These types of interdependence are present in manufacturing contexts as well, where one worker may perform some aspect of work while a colleague works nearby. In *Sequential Interdependence*, each individual works on a task in sequence, and the output of one individual can serve as the input for the next. While disruptions to one individual's subprocess may have downstream effects, the pre-defined structure of the task avoids many ambiguities about coordination strategies. Examples of sequential interdependence include cases like assembly lines. Finally, in *Reciprocal Interdependence*, each individual contributes to a task simultaneously, usually with an assignment of specific aspects of the collaborative work, and a high level of cooperative behaviors and signalling. Tasks such as this could include collaborative assembly, where one worker holds an object while the other performs some action on it. Another method of categorizing collaborative robot interactions was created by Christiernin (2017), which instead grouped these interactions into four increasing levels of interaction ranging from *Level 0*, featuring no interaction, to *Level 3*, featuring a high level of interaction and use of joint activities. Regardless of the

specific method of classification, these distinctions are important, both because they differ in the complexity of their design, but also because they differ in how they impact the individuals performing them. For example, Zhao et al. (2020) found that higher levels of interdependence were associated with less physiologically-measured stress for the human worker, and improved perceptions that they were working with another collaborator.

If greater collaboration and interdependence have such benefit, then why not have more? The issue is that doing so requires both a deep understanding of the task to be performed, as well as the ability to create behaviors for the cobot that adequately communicate intent, attention, and understanding. For example, the use of gaze and joint attention has been shown to improve task performance and perceptions (Huang and Thomaz, 2011; Andrist et al., 2017). At a practical level, however, the implementation of such a system requires apparatus to track the attention of the human worker, detect the focus of the attention, and then allow the robot to reason about how that attention relates to its knowledge of the process being completed.

Similarly, the relation between the physical motions that the cobot makes and their temporal and spatial relation to the human worker is also important. For example, robots that are too slow to respond to human actions such as handovers negatively impact both their perceptions of the robot and the task efficiency. Ideally, such systems are able to detect and adapt to these timings (Huang et al., 2015; Huang and Mutlu, 2016). This also extends to the framing that the actions take, where user-centric approaches can promote actions better suited for the specific human worker (Wang et al., 2020).

This isn't the only manner in which a cobot may need to adapt in real time, however. Safety is a major concern in collaborative robotics (Siebert-Evenstone et al., 2021). Despite their safety features, it is still possible to program unsafe actions, or incorrectly handle unexpected human behaviors. The difficulty in managing this unpredictability is in part due to the complex nature of what defines an unsafe action, since the same behavior executed at a different speed, or in a different location, while carrying a different object, may be designated as safe or unsafe.

Compounding this issue is that robotic motions, especially ones that involve robots with sufficiently high degrees of freedom, are the product of multiple joint states, which can be difficult to reason about. If the human worker is poorly modeled, and their current positions poorly estimated, this makes the aforementioned challenges even more difficult. The challenge in resolving these ambiguities has made it a major hurdle to the successful integration of cobots into the workplace (Tian et al., 2023).

Coordination and collaboration is not simply timing and motion. While gaze and attention are important, they represent only a portion of the types of social behaviors that contribute to the overall experience. Whether intended or not, cobots are generally received by their collaborators as social entities (Sauppé and Mutlu, 2015), and their acceptance by these collaborators is in large part dependent on how well they make use of these social cues. For example, gesture cues and lifelike motion have been shown to improve the perceptions of cobots (Elprama et al., 2016; Cuijpers and Knops, 2015; Terzioğlu et al., 2020). It is why Fischer (2019) argues that collaborative robots should embrace this social and emotional aspect of collaboration.

The logic of the collaborative program requires consideration as well. Whereas traditional robotics could in large part utilize non-branching logic methods given their cyclical and repeated behavior, many sequential and most reciprocal collaborative tasks require architectures more amenable to the handling of control flow and events, like the ones mentioned above. This has led to the interest in representations which support this type of framing. These vary from imperative approaches (Huang et al., 2016; Huang and Cakmak, 2017) to trigger-based approaches (Senft et al., 2021b) to flow-based (Alexandrova et al., 2015). Regardless of structure, it is important to note that these programs are not the syntax alone, but also the physical representation of the program, given that the behaviors specified in these programs are meant to be executed by an embodied physical agent working in a physical space. As such, Ajaykumar and Huang (2020) emphasize the importance of visualizing and communicating this information to the program specifier.

2.3 Proximal Development and Scaffolding

As seen, collaborative robotics represents a diverse set of challenges, given the complexity of the tasks, and all the different aspects of the implementation that must be considered. A contributing cause of their relatively limited deployment thus far (considering their potential) is that the individuals tasked with condensing all the necessary factors into programs are trained in standard robot programming. They lack the background and perspective to consider these interdependencies and interactions because they were irrelevant to the domain they were trained in. Alternatively, individuals who may provide these perspectives might not have the technical expertise or access to the tools needed to contribute to the programming process.

So how then do we bridge this gap? The domain of education suggests that this can be accomplished in part by scaffolding (Berk and Winsler, 1995; Gonulal and Loewen, 2018). This idea, predominantly discussed in the concept of childhood education, comes from the theory of the Zone of Proximal Development by Lev Vygotsky. Specifically, it suggests that when an individual is learning something new, there exists a gap between what they currently know, and what they are trying to learn. The challenge is that at their current level of understanding, the individual may not be able to fully grasp the full scope of the new concept all at once. Instead, intermediate steps within this zone of proximal development are needed that allow the individual to gradually build up their understanding of the concept.

This can be applied to the context in which individuals learn to program. Young (1981) suggests that programmers develop a mental model of the program while doing development work, and Mayer (1981) suggests facilitating approaches by which the current knowledge of the programmer can be connected to the novel information they have yet to learn. All this motivates the idea that part of the solution to bridging this gap is to create methods by which the programmer or contributor can be introduced to these concepts, while still allowing them to leverage their existing domain expertise.

This is only part of the solution, however. Even with scaffolding and highly

effective representations, the problem remains difficult. Therefore, any solution also needs to consider how feedback and program augmentation can be used to supplement the capabilities of the programmers. Porfirio and colleagues have done this in the context of social robotics, creating systems which improved the quality of robot programs through verification and support (Porfirio et al., 2018, 2020). To fully realize the potential of collaborative robotics, solutions must consider how to frame these problems in a way that is accessible to the programmer, and how to provide feedback and support that allows them to create programs that improve the quality of the resulting programs.

2.4 Representations of Collaborative Interactions

To understand what types of representations are used, and how they may need to adapt for collaborative robots, let us first consider many of the methods by which cobots are currently being programmed.

In terms of current usage, the most common method of programming is through the use of imperative or sequential programming approaches (Michaelis et al., 2020), frequently through the use of systems like Universal Robots' Polyscope software (Universal Robots, 2023), which feature a hierarchically-organized sequence of behaviors. While these systems can be extended with custom scripting, the core assumption of sequentiality remains, much like in conventional robotics.

These behaviors can be either manually scripted, or taught through methods like Programming by Demonstration (PbD) and Kinesthetic Teaching, where the robot is guided physically by the programmer through a series of actions which demonstrate the desired behavior (Billard et al., 2008; Akgun et al., 2012; Skoglund et al., 2007). While having the benefit of leveraging the physical embodiment of the robot within the space, and being generally easy to use, these methods can be limited in their generalizability, since in the simplest cases, the only feature recorded is the sequence of keyframes, thereby lacking higher-level understanding of the intended task. Additionally, during a demonstration there may be aspects of the demonstration which are intended, while others may be unintended or

superficial. Some methods have been developed to address these limitations by specifying multiple demonstrations and prioritizing certain demonstrations over others Sakr et al. (2022) or incorporating a labelling process (Ravichandar et al., 2020). While the implementation details differ, these methods attempt to use these demonstrations and labels to infer higher-level representations of the task, independent of specific demonstrations. In the end, however, the result of these methods is a specific skill, possibly parameterizable, which can be executed from within a larger context.

A high-level approach to programming cobots involves planning. In this approach, the programmer specifies a set of valid actions, which include rules about when their usage is valid, and what the expected outcomes of performing them are. A planner algorithm takes this specification, along with the current state of the world and the goal state, generating the set of actions needed to arrive at the intended goal state. A common representation for these planning specifications is the Planning Domain Description Language (PDDL), for which a variety of versions exist with varying flexibility and capability of expression (Fox and Long, 2003). It has also been extended to the Hierarchical Domain Definition Language, which allows for the hierarchical specification of skills (Höller et al., 2020). To be clear, PDDL and HDDL refer to the representation of the specification, not the algorithm used to generate the plan. A variety of algorithms exist for this purpose, each with differing capabilities and performance characteristics. If executed once, the output of these solvers is a static plan that could be executed from start to finish. Since the plans themselves don't adapt to changes in the environment that occur during the planning or execution process, a common approach is to use replanning algorithms, by which the solver will periodically recompute a plan, usually when a discrepancy is detected between the expected state and the observed world state.

This type of planning is not meant to be confused with motion and trajectory planning, which similarly involves the specification of a current and goal state, but instead focuses on the lower-level challenge of finding the sequence of joint or motor commands that move the robot between them. A variety of algorithms to perform this function exist, such as OMPL (Şucan et al., 2012) and TrajOpt

(Schulman et al., 2013). Like standard planners, replanning is frequently combined with these methods, usually to handle safety and incorporate collision avoidance (Palleschi et al., 2021).

There exist a variety of different state-based or flow-based programming approaches for specifying higher-level behaviors. These include behavior trees (*e.g.*, CoSTAR (Paxton et al., 2017)), and flow diagrams (Fogli et al., 2022). With these approaches, tradeoffs exist between high expressivity and ease of use, as shown with CoSTAR, which allows for the specification of complex behaviors, but requires a high level of expertise to use.

Finally, given the success of Large Language Models in program generation, interest has grown in using these methods to generate robot and cobot programs. In some cases, these methods can be used as alternatives within the approaches outlined above, such as in PDDL planning (Silver et al., 2022), or more standalone vision-language-action systems (Brohan et al., 2023). While these methods are intriguing and promising in many ways, certain concerns currently remain, such as the forgetfulness with which these systems can generate programs (Chen and Huang, 2023).

In short, the methods by which cobots are programmed are diverse, and tend to focus on varying levels of the task specification. For example, some approaches may focus on the high-level structure of the task or the overall plan, while others focus on the low-level details of the motion. This impacts the work within this dissertation. At times, this means borrowing from higher-level approaches described here, or ones which are more analogous to the sequential structures of traditional robotics. Through strategic application of analysis, synthesis, and optimization methods to these representations within specialized tools and systems, we look to improve the effectiveness of cobot programmers.

3 AUTHR

The work presented in this chapter concerns the rationale, design, and implementation of *Authr*, a system for authoring human-robot collaborative plans from specifications of agent-agnostic ones. Prior to this work, there had been research considering the process of allocating work between humans and robots as optimizations meant to minimize cycle time and ergonomic strain (Pearce et al., 2018), but little that considered how such systems would be made accessible and usable to the individuals (e.g. engineers) who could use them.

This is an overlooked, but essential consideration. As discussed, there is currently a disconnect between the theoretical capabilities, or promise, of cobots, compared to their current usage (Michaelis et al., 2020). Despite their capability and demonstrated utility in certain instances, their usage is comparatively low (Berger and Armstrong, 2022), and the reason is that the key benefit of cobots (as opposed to automation) – collaboration – is made difficult by the lack of straightforward ways to reprogram, reconfigure, and integrate into existing systems. Contributing to this challenge is the lack of skills and knowledge of the individuals who would be responsible for such integration (Michaelis et al., 2020; Berger and Armstrong, 2022).

Therefore, the challenge of adding a collaborative robot, and understanding how it should be utilized, is an essential stage in facilitating this adoption. Moreover, any such solution needs to be accessible to the individuals who would be responsible for the integration. *Authr* attempts to address this challenge by providing a system that allows users to specify tasks in a way that is familiar to them, and then automatically generates a collaborative plan that can be executed by a robot.

The work in this chapter addresses four key technical challenges involved in human-robot teaming: (1) *representation*: representing work for both human interpretation and robot execution; (2) *task-skill matching*: creating human-robot plans that match task elements with worker skills while achieving task goals; (3) *robot programming*: implementing task elements for collaborative robots in a way that

supports exploration of task plans across robot platforms; and (4) *authoring pipeline*: facilitating intuitive and effective translation of manual work into human-robot plans¹. Building on methods and tools from ergonomics, robotics, and human-computer interaction, we address these challenges by (a) formalizing a task- and action-level representation that is human-interpretable and robot-executable, (b) utilizing a multi-agent allocation algorithm that generates plans that match worker skills to task elements within task constraints, (c) developing a software stack which converts plans into robot-executable actions built on an extendable Robot Operating System (ROS) (Quigley et al., 2009) infrastructure, and (d) designing an intuitive software environment that enables users to effectively create human-robot plans.

In the remainder of the chapter, we discuss these technical challenges in more detail, describe our solutions for each challenge, present the system design and implementation of *Authr*, describe two user studies that evaluated different facets of human-robot teaming using *Authr*, and discuss our findings and their implications for the design of tools which support the authoring of human-robot collaborative plans. The contributions of this work include:

- A novel *workflow* to translate manual human tasks to human-robot tasks;
- Novel *representations* and formalizations for modeling, planning, simulation, and implementation;
- The design of an *authoring environment* that supports users in following this approach;
- An *open-source implementation* of the environment for public use and further development;²

¹The research discussed in this chapter is derived from published work by myself and Curt Henrichs, Mathias Strohkirch, and Dr. Bilge Mutlu. All authors contributed significantly to the conceptualization, design, implementation, evaluation, analysis, and/or the writing of the original manuscript.

²<https://github.com/Wisc-HCI/authr>

- *Empirical evaluations* of the approach and the authoring environment through a series of user studies.

Target Users

In this work, the design of *Authr* was meant to target individuals like automation experts and ergonomics specialists. Given the potential overlap between these two fields, or the likelihood that individuals in each group may collaborate towards automation goals, a key consideration was to design a system which was intelligible to both users. For this reason, representations like *Therbligs* were chosen, as they are commonly used in ergonomics and human factors research. Process representations like Hierarchical Task Analysis (HTA) (Stanton, 2006) were also utilized, as they may also appear in both fields. In line with our understanding of both groups, we sought to design a system that through these familiar starting points and beneficial feedback and assistance would allow users to create and iteratively explore the design of collaborative robot plans.

Research Questions

Authr represents the first and most rigorously empirical exploration of customized authoring tools for human-robot collaboration this dissertation. Importantly, we consider whether certain representations are both effective and understandable by the target users, the effects that providing assistance like automated agent allocation have on the overall quality of a human-robot plan, and what the impact of such capability - or lack thereof - has on the user experience.

3.1 Background

A great deal of prior work has focused on the development of visual programming environments (VPEs) to enable easy programming of tasks. A primary example is the student-oriented *Scratch* interface, which uses a block design to indicate

conventional programming constructs (Maloney et al., 2010). This approach has inspired a number of VPEs such as *Hammer*, a robotics-focused, android-based programming tool allowing novice users to design programs for robot arm movement and tool use (Mateo et al., 2014), and *Code3*, a drag-and-drop system built for the PR2 robot (Huang, 2017), among others. Another influential VPE, *LEGO Mindstorms NXT Programming Environment*, focused on education and robotics (Kim and Jeon, 2007; Klassner and Anderson, 2003). Flow designs have also been investigated: *Roboflow* embeds pre- and post-conditions into flow structures, focusing on a low-level specification of behaviors for robotics (Alexandrova et al., 2015); and *ROSCO* (ROS Commander), a tool created for the PR2 Robot, uses hierarchical finite state machines and low-level building blocks to specify spatially situated actions (Nguyen et al., 2013). While all these interfaces contributed substantially in a variety of ways, they generally focused on specifying *robot behavior*, as opposed to *human-robot collaboration*.

The space of human-robot collaboration specification is still quite new. The ROBO-PARTNER project has helped by articulating the needs and requirements of such systems, namely user-friendly interfaces, planners that allow the creation of efficient human-robot collaboration task plans, robot instruction libraries that allow for easy generation and modification of robot programs, and continual attention to safety concerns (Michalos et al., 2014, 2015). In an attempt to begin addressing these requirements, the *CoSTAR* system was developed, which integrates perception and reasoning into behavior trees (Paxton et al., 2017, 2018). While the interface was successful in allowing users to specify complex programs, users had difficulties understanding the types and intentions of the robots' actions. *RAZER* was designed for task-level programming to allow shop-floor operators to leverage lower-level actions developed by experts, and it was later extended to support programming by demonstration (Steinmetz et al., 2018, 2019). They compared their solution with systems such as *CoSTAR* and *Scratch*, finding *RAZER* to be easier to understand by non-experts. Graphical representation of the workspace to assist users in creating task graphs have also been explored Riedelbauch and Henrich (2018).

In an effort to improve the efficiency of human-robot plans, research into multi-

agent task planning has been explored with works such as *Tercio* Gombolay et al. (2018) and *multi-abstraction search approach (MASA)* (Zhang and Shah, 2016). *Tercio* takes inspiration from real-time processor scheduling for multi-robot hierarchical problems. The objective is to assign tasks to agents and schedule tasks with the goal of minimizing change in agent assignment and minimize number of spatial interfaces between tasks assigned to different robots. *MASA* uses a multi-level optimization approach with three phases: finding an initial solution for agent placement, hill-climbing to minimize maximum make-span, and finally refinement to the solution. While both approaches work well for optimizing agent allocation, there is a relatively high planning time. Another consideration is prioritization of various goals such as maximal efficiency and minimal strain, as shown by Pearce et al. (2018). They found that tasks that benefited most from the goal of minimizing time and ergonomic strain were ones which enabled parallel work, were repetitive, and utilized robot-performable actions.

Researchers have started to address how to leverage agent allocation in human-robot-collaborative authoring environments with systems such as *Sharedo* (Kato et al., 2014) and *WeBuild* (Fraser et al., 2017). *Sharedo* functions as a structured to-do list for daily tasks where multiple agents (human, robots, virtual-assistants) coordinate based on their capabilities. *WeBuild* provides allocation of tasks for multiple humans with varying capabilities in order to offload group coordination. Our work, drawing from the related literature, addresses the challenges of authoring human-robot collaboration within the manufacturing context.

3.2 Technical Approach

Translating manual tasks into human-robot task plans involves a number of technical challenges. We discuss these challenges in this section and detail our solutions in the next section.

1. Representing tasks for humans and robots

Translating tasks that are currently performed manually by human workers into human-robot plans requires representing them in a way that is both interpretable by a human, so that they can be trained on the task and their performance can be assessed, and executable by a robot. Tasks describing manual work in manufacturing settings are generally represented as written natural-language lists of mid-level descriptions of task actions. Although this representation is human-interpretable, implementing tasks into robots based on these descriptions is challenging (Paxton et al., 2018). Furthermore, users without the necessary experience in developing collaborative applications may generate implementations which are not generalizable across robot platforms and are ill-suited for task-level analysis of plan efficiency or safety. Therefore, we need a representation that enables the user to capture task elements from natural-language descriptions or from qualitative observations of the task and to specify task elements for humans and robots to perform.

2. Matching task elements with worker skills

Answering the question of which aspects of the task that robots and humans should perform is critical to realizing the promise of human-robot teaming for improved productivity and worker safety. This requires effectively matching human and robot skills to elements of the task, considering the cost of the human or the robot performing the elements. Furthermore, while a simple matching can determine whether a specific task element can be performed by a human or a robot, it does not help the user determine whether it should be performed by a human or a robot given specific task expectations and requirements, such as speed (a robot that can perform a task element may be too slow) and ergonomic safety (a task element that a human worker can perform much more efficiently may be ergonomically unsafe for the human worker). Hence, there is a need to match task elements to the skills and capabilities of human workers while considering outcomes such as efficiency and safety at a task level.

3. Supporting exploration across robot platforms

When engineers in industry are considering converting a manual process into one involving a collaborative robot, either as automation performed by the robot or collaboration between the robot and a human operator, they are faced with the decision of using manufacturer-provided software environments (*e.g.*, Universal Robots Polyscope³), utilizing third-party tools (*e.g.*, Artiminds⁴), or developing a custom software solution built on top of low-level APIs. Compounding the problem of making an informed choice is a potential lack of experience in developing collaborative human-robot teaming applications (Michaelis et al., 2020). Therefore, we need to provide a tool that enables users to quickly and easily evaluate their tasks for multiple robot platforms before purchasing a particular robot. For example, an engineer interested in understanding whether a Universal Robots UR5 robot or a Franka Emika Panda robot would better fit into a given task would likely have to implement the same task for both robots using different programming tools or setups, as highlighted by the creators of the CoSTAR robot programming environment (Guerin et al., 2015). Furthermore, if the engineer is interested in seeing alternative task plans in action to further refine them, the user must program each plan individually. Users should be provided visual or demonstration-based robot programming tools in order to easily program robots and integrated planning tools to easily handle skill-based task allocation. Thus we need to enable the user to quickly and easily develop, deploy, view, and modify task plans for end-to-end exploration across multiple collaborative robot platforms.

4. Developing an intuitive and effective authoring pipeline

A final technical challenge is to enable users to rapidly and iteratively capture task models for manual work, explore human-robot task plans, and deploy the created plans on robot platforms for assessment, refinement, and training. Although users might have prior experience with robot programming tools, such as the

³<https://www.universal-robots.com/>

⁴<https://www.artiminds.com/>

demonstration or visual-programming tools that are commonly used to program collaborative robots, we must create intuitive software tools that users can quickly learn and use in order to effectively facilitate the complex process of human-robot teaming.

1. Creating a shared representation for human-robot work

The goal of our representation is to facilitate the translation of natural-language task descriptions to a formal representation that remains interpretable to human collaborators, yet robots can understand and perform without having to update their underlying implementation. The tasks being translated are generally in the form of written natural-language lists composed of task specific actions or tasks that can be observed by an engineer as they are being performed by an operator. Although examples of translating task-specific actions into robot action primitives exist in various domains (*e.g.*, cooking (Bollini et al., 2013) and route-navigation (Bugmann et al., 2004)), these solutions tend to be highly contextual or robot-specific. One promising solution is *Therbligs*, as proposed by Gilbreth and Gilbreth (1924), which address the issue of defining operational action primitives for human work. Researchers have since applied *Therbligs* to modeling or specifying robot behavior in various contexts (Lin and Chiang, 2015; Jun et al., 2012; Pearce et al., 2018; Akrouf et al., 2013).

Our representation builds on *Therbligs* and is further inspired by the work of Pearce et al. (2018) where they modeled human-robot tasks using Hierarchical Task Analysis (HTA) with lowest-level sub-tasks allocated between humans and robots. In work analysis literature, HTA decomposes tasks into nested sub-tasks until sufficient detail is achieved to perform work actions (Stanton, 2006). Our representation adopts this approach with two important changes. First, our approach only considers three levels in HTA, operationalized as the *Plan*, *Task*, and *Therblig*, where *Therbligs* function as sub-tasks. Second, *Therbligs* have both high- and low-level parameters. High-level parameters include *Agents*, *Things*, *Destinations*, while low-level parameters include numerical values, such as gripper effort, time, and

cost.

Agents, Things, and Destinations—In our representation, *Agents*, *Things*, and *Destinations* are used to fully specify the high-level behaviors of *Therbligs*. Consider the action of a robot placing an item, such as a mug, in a container for shipping. In this case, we can think of the transport action as the *Therblig*. High-level parameters, such as *Agents*, *Things*, and *Destinations* serve to characterize these therbligs and more clearly define their behavior. Thus, in the mug packing example, the *Therblig* is specified by the *Agent* performing it (the robot), the *Thing* being moved (the mug), and the *Destination* it is moved to (the shipping container). Thus, the combination of high-level parameters and the *Therbligs* serve to symbolically define the action to the engineer. In *Authr*, we consider an *Agent* to be any physical actor in the work environment that performs a relevant action within the context of the *Plan* and has a type (human or robot). *Things* are regarded as objects within the environment that are manipulated by *Agents* in the *Plan*. *Destinations* are operationalized to combine semantic labels, such as the described location (e.g., the “shipping container” in the example above) with a concrete position and orientation that a robot could act on. Due to this representation, *Authr* enforces a strict set of spatial expectations on the workspace, meaning real-time dynamics and variability are not considered in the current version of the system. As such, this solution works well for clearly defined workspaces (e.g., kitting), but not for ones with variable *Thing* counts or positions (e.g., bin-picking).

Plan, Tasks, and Therbligs—At the highest level of our HTA approach is the *Plan*, which in *Authr* reflects the entirety of the human-robot collaborative work being designed. The *Plan* is composed of *Tasks*, which represent high-level descriptions of behaviors used to achieve specific processes in the *Plan*. *Tasks*, in turn, are composed of *Therbligs*. The full set of 18 *Therbligs* includes physical actions, cognitive processes, and behaviors that are both physical and cognitive (Gilbreth and Gilbreth, 1924). For the current implementation of *Authr*, we focus on physical actions, resulting in the following list of *Therbligs*: (1) *Transport Empty*, (2) *Transport Loaded*, (3) *Grasp*, (4) *Release*, (5) *Rest*, and (6) *Hold*. These *Therbligs* are also listed in Figure 3.1 along with their descriptions. By focusing exclusively on physical *Therbligs*, our task

space is generally constrained to pick-and-place-type tasks (*e.g.*, kitting, assembly, palletization). Some limited tool use can be created in an *ad hoc* manner (*e.g.*, grasping a screwdriver and defining screwing rotation through multiple *Transport Loaded Therbligs*), but tasks requiring cognitive evaluation (*e.g.*, force-sensed peg-in-hole or component inspection) are currently not addressed in our representation. Further work is needed to operationalize cognitive and mixed cognitive-physical *Therbligs*.

Setting an *Agent* for the high-level parameter of a *Therblig* has the effect of allocating it to that *Agent*, and leaving it empty prompts automated allocation. Other high-level parameters, such as *Things* and *Destinations* are required. These high-level parameters are used to generate pre- and post-conditions of each *therblig*, which serve to describe when the *Therblig* can be performed and what the effect on the workspace will be. In the mug packaging example from above, given that the action was configured with the robot, the mug, and the shipping container, we can say that for this action to be performed, the robot must be both holding the mug, and the space for the mug in the container must be empty. At the end of the action, both the mug and the robot will be positioned at the container. The full breakdown of parameters, pre-conditions, and post-conditions for our *Therbligs* are shown in Figure 3.1.

Alongside high-level parameters (*Agent*, *Things*, and *Destinations*), we need a way to standardize and compare the quality of *Therbligs* to sufficiently allow for shared representation of these collaborative tasks. Low-level parameters support this reasoning by providing low-level information required to execute an action by the robot, but may not be required by the human (*e.g.*, gripping effort). In this way, if a robot can theoretically perform the task, and if it is allocated the *Therblig*, it has the necessary information to perform the task. Low-level parameters also provide comparative power to *Therbligs* allocated to separate *Agents*. Time to complete an action can be simulated by the robot, but knowing the time for the human to perform the task would be necessary for determining which *Agent* is fastest at performing it. Likewise, if a task is hard for a human but easy for the robot (or vice versa), being able to weigh these values is necessary for thoughtful allocation

of tasks. One common metric of difficulty is ergonomic strain, and being able to flexibly define this cost for a given *Therblig* and *Agent* can empower the engineer to construct programs that provide robotic assistance where it is needed most.



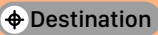


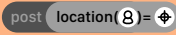





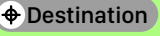
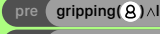




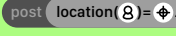










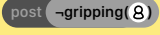






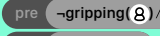





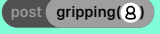



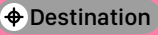
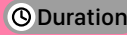
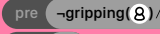











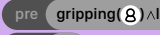





Therblig	Parameters	Pre-/Post-Conditions
 Transport Empty <i>Reaching for a thing with an empty hand or gripper</i>	 Agent  Destination	pre  gripping() post  location()= 
 Transport Loaded <i>Moving a thing with a hand or gripper to a destination</i>	 Agent  Thing  Destination	pre  gripping()  location()=location() post  location()=   location()= 
 Release <i>Letting go of a thing from a hand or gripper</i>	 Agent  Thing	pre  gripping() post   gripping()
 Grasp <i>Grabbing a thing with a hand or gripper</i>	 Agent  Thing  Effort	pre   gripping()  location()=location() post  gripping()
 Rest <i>An inactive period or pause at a destination</i>	 Agent  Destination  Duration	pre   gripping()  location()=   post 
 Hold <i>An inactive period while holding a thing</i>	 Agent  Thing  Duration	pre  gripping()  location()=location() post 

Figure 3.1: A description of the *Therbligs* implemented in *Authr*, including parameters, pre-conditions, and post-conditions.

2. Enabling effective task allocation in human-robot teams

A critical challenge in authoring human-robot collaborative tasks is the gap between engineers' ability to construct single-agent programs and the know-how of designing interactive tasks. Even if some tasks or sub-tasks are only executable by one *Agent*, the rest of the interaction needs to be planned in a way that incorporates those restrictions on agent allocation. Since many engineers from the manufacturing domain have access to task specifications (albeit non-interactive, manual ones), we sought a representation that translates this type of non-agent focused representation into an interactive plan. Our operationalization of *Therbligs*, along

with the parameters we specify, allows for a direct translation from their task specifications to the shared *Therblig* representation, from which the interactive *Plan* is constructed. This construction process requires any non-specified *Agents* to be allocated to a given task, all while accommodating specified (*i.e.*, parameterized) *Agents* and considering cost and time estimates.

Our proposed allocation process is performed in a series of steps. First, the *Plan* is checked using the SMT solver Z3 (De Moura and Bjørner, 2008), in which the pre- and post-conditions of each *therblig* are translated into first-order logic and verified. This same algorithm is used continuously during plan construction to provide feedback about program correctness to the user. Next, the *Plan* is further checked that all needed parameters are defined, since not all parameters need to be set for verification to succeed. Following this parameter check, the *Plan* proceeds to allocation. For this purpose, a breadth-first search through the interaction is utilized, resulting in a set of possible interaction traces. In the worst case, the state size upon applying each *Therblig* t_n of t_1, t_2, \dots, t_n has an upper bound of 2^n for two agents. However, we observe that users typically chain together consecutive *Therbligs* for an *Agent* into individual *Tasks* (*e.g.*, pick-and-place: *Transport Empty* \rightarrow *Grasp* \rightarrow *Transport Loaded* \rightarrow *Release*). Due to the pre- and post-constraints, *Things* act as tokens constraining the growth of the state space, and the state space grows instead with 2^m where m is the number of *Tasks*. Since these traces are modeled as a single sequence of consecutive actions (for computational efficiency), the *Plan* is then parallelized such that allocated *Therbligs* are performed as soon as possible while maintaining first-order logic. The resulting traces are compared for overall time and cost, using the provided time and cost weights, and the optimal interactive *Plan* is returned.

The method described above was chosen because it most closely matched the formulation of the *Plans* as provided by the users, namely an initial state and a set of non-allocated *Therbligs* to perform. With some additional work, and some caveats, the *Plans* can be converted into standard planning-based problems. The first caveat is that due to the nature of our approach, the ordering of *Therbligs* is constrained for a given *Agent*. Combined with the token-like nature of *Things*, this means that users

can specify sequences of *Transport Loaded Therbligs*, thereby creating waypoints, with certainty of the ordering that the *Agent* will visit them. Additionally, if some intermediate placement of an *Agent* or *Thing* is required, but not captured in the final state, a coarse planning will not result in this configuration, unless segmented into multiple planning problems or supplying additional explicit goals.

While the allocation and parallelization algorithms specified were sufficient for the complexity and size of *Plans* considered in this study, it is important to consider how such methods compare to more conventional planning approaches. To this effect, we ran benchmarks with our process and a *Multi-Objective Divide-and-Evolve* (MO-DAE) algorithm, which is an evolutionary algorithm which supports multi-objective planning (Dréo et al., 2011; Khouadjia et al., 2013). Since interactive design is a key component in the user’s workflow, we needed any algorithm to be sufficiently fast. Thus, we capped the maximum compute time at 90 seconds for quick user feedback. As input, we modeled three *Plans* (shown in Figure 3.2) in *Authr* based on real-world manufacturing tasks. Each *Plan* was evaluated five times with each algorithm. Our algorithm was deterministic, so there was no variation other than slight differences in compute time.

The first *Plan* models a kitting task (assembling objects into containers or kits) in which there is a grouping of four toys and four batteries on the left side of the workspace. The goal of this *Plan* is to move one toy and one battery into four separate boxes, located to the right. The second *Plan* models a circuit board assembly task. The initial state of this *Plan* consists of a group of four nuts located to the right of a PCB, and two cables positioned above the board. To complete this assembly process, one nut must be screwed onto each corner of the circuit board, and each of the two cables connected to the circuit board. The third *Plan* models a repair task in which two faulty components are replaced by new components on a circuit board. The two components are functionally different, with one requiring considerable cost for the human to place but not remove. This repair task starts with the two distinct faulty components attached to the circuit board and the two distinct new components off to the side. The goal of this *Plan* is to remove both faulty components, placing them to the right, and replace them with the new components of the same type, located

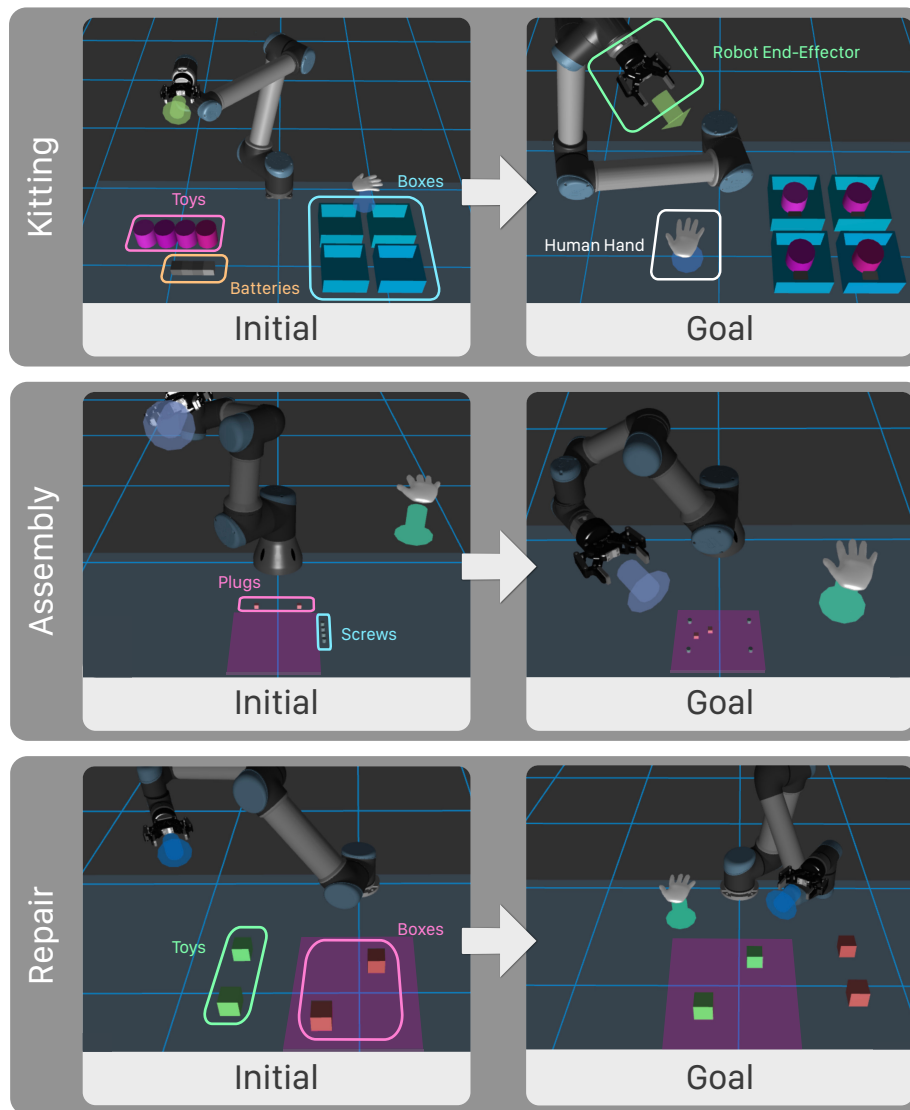


Figure 3.2: For the technical evaluation, we constructed three manufacturing tasks: Kitting, Assembly, and Repair. For Kitting, *top*, a toy (cylinder) and a battery pack (cube) were moved to each container. In Assembly, *middle*, screws (grey cylinders) are placed in each of the four corners of a PCB and rotated, while two cables (pink cubes) are placed in the center. Finally Repair, *bottom*, features two faulty components (red cubes) being removed and replaced with new parts (green cubes).

to the left.

In order to estimate the *Therblig* times for the human *Agent*, we set up a physical representation of each task and recorded the amount of time it took for a human to perform each *Therblig*.

For each of the three *Plans* we evaluated, we targeted different time and cost metrics. In the first *Plan* (kitting), the objective was focused on minimizing time, so the optimization weights for time and cost were set to 0.6 and 0.4, respectively. In the second *Plan* (assembly), the objective was to minimize cost, so time and cost weights were configured at 0.05 and 0.95, respectively. Of concern in this *Plan* was the ergonomic strain for humans in screwing in the nuts. Thus, the cost of this action was configured to be higher for human than robots (0.9 vs 0.2). In the third *Plan* (repair), we set the cost weight to 0.6 and the time weight to 0.4. In this *Plan*, we were interested in the effect of a high cost related to a single action and *Agent*, as opposed to a class of actions.

Results of both methods (*Authr* Allocation versus *MO-DAE*) for the three different *Plan* types are shown in Figure 3.3. This evaluation showed that while the two methods were roughly comparable for optimizing in *Plan* time, cost, and overall score, the compute time for these similar metrics was less for our implementation. Since a focus of *Authr* is to enable the exploration of *Plans*, especially through iterative refinement, we chose to utilize the simpler implementation outlined above for further testing. However, we note that while this method was sufficient for these purposes, alternative methods may be superior with *Plans* of different size or complexity.

3. Implementing task plans into a collaborative robot.

Authr connects to a server developed for Robot Operating System (ROS) (Quigley et al., 2009), running on Ubuntu. We chose to develop our system in ROS to enable future integration with physical robots. For robot trajectory planning, estimating *Therblig* action time, and simulation, our implementation uses MoveIt (Chitta et al., 2012), specifically using Open Motion Planning Library (OMPL) (Şucan et al.,

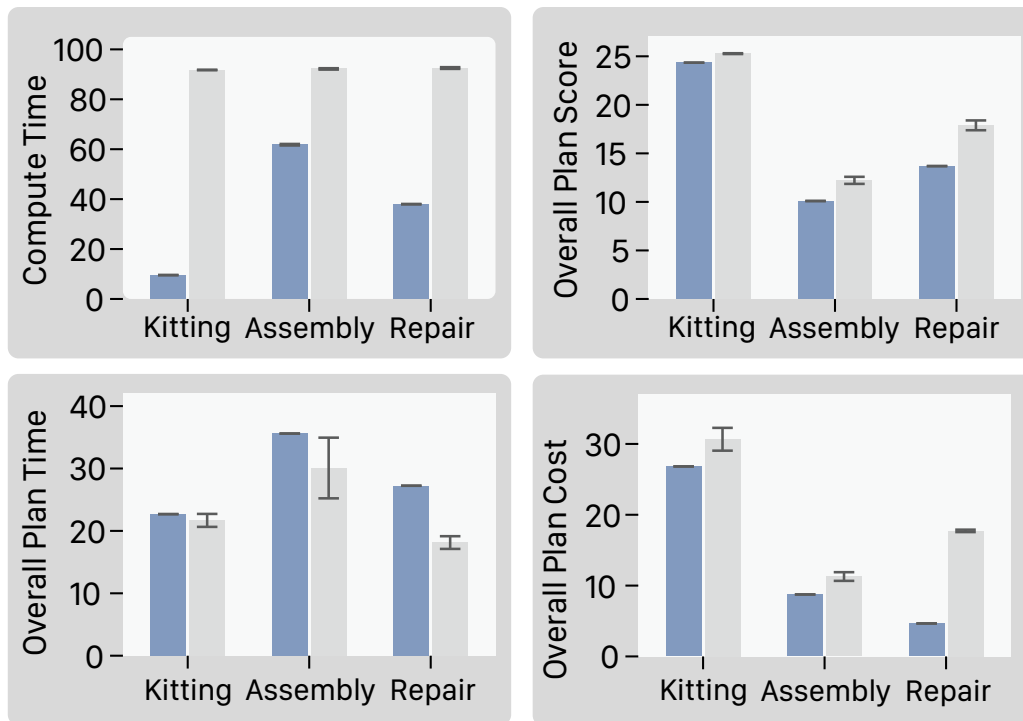


Figure 3.3: We evaluated the *Authr* Allocation and Parallelization algorithms (blue) versus a *MO-DAE* planner (grey) with three different *Plans* (*Kitting*, *Assembly*, and *Repair*) on 4 different metrics (*Compute Time*, *Overall Plan Score*, *Overall Plan Time*, and *Overall Plan Cost*). Lower scores for all metrics are desirable.

2012). Because MoveIt is freely available and configurations are easily made, this choice makes adding additional robots to *Authr* straightforward. While the current implementation allows the user to choose from the Franka Emika Panda or Universal Robots' UR3, UR5, and UR10, any robot with a MoveIt configuration could be added. Utilizing a standard inverse-kinematics and motion-planning tool enables us to achieve our goal of a shared representation by converting our spatial-semantic representations to robot-specific control. Thus, *Therblig* behavior implemented on top the motion planner achieves *Agent*-agnostic functionality.

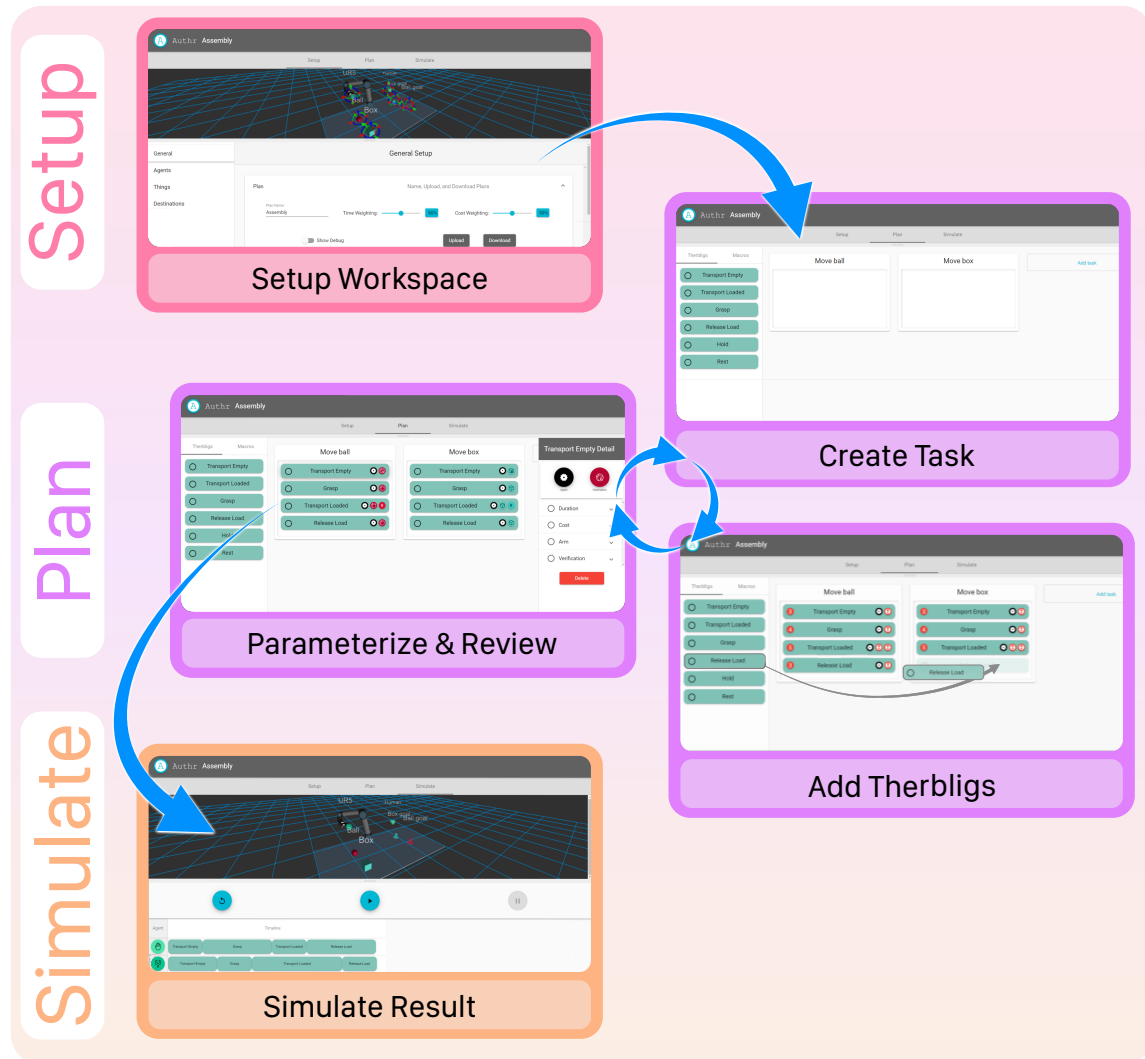


Figure 3.4: The three modes in *Authr*. In *setup*, users first configure the workspace; *Destinations* are able to be added, deleted, and modified, and each *Agent* and *Thing* gets assigned an initial location in the scene. Moving into planning in the *Plan* Tab, *Tasks* are represented as containers for *Therbligs* and are ordered from left to right. Within each *Task*, *Therbligs* are ordered from top to bottom. *Therbligs* and *Tasks* are also configured. In *simulate*, after designing an interaction, users can simulate the actions of human and robot *Agents*.

4. Facilitating the exploration of human-robot task plans.

Authr integrates the above representations and technologies into a visual programming environment. This environment is built using the Angular web framework (Google, 2019) as a browser-based application, which connects to a ROS-based server using Robot Web Tools (Toris et al., 2015). *Authr* has three modes, setup, plan, and simulate, which a user works through in five main steps, (1) setting up the workspace, (2) creating tasks, (3) adding therbligs, (4) parameterizing, and (5) simulating the result (Figure 3.4). The first step is for a user to set up a workspace, as the *Agents*, *Things*, and *Destinations* created in this step will be needed for the following steps. Next, the user can create a *Task* and a set of *Therbligs* that will be performed in this *Task* by dragging them from a library and dropping them in the *Task* container. The user can then parameterize the *Therbligs* and review any errors identified by *Authr*. Based on these errors and the remainder of the *Plan*, the user can either decide to create another *Task* or continue adding *Therbligs* to an existing *Task*. At this point, the user can also navigate to the simulation view to see the plan played out. After reviewing the simulation, the user can create another *Task* to add to their *Plan* if desired. Below, we detail how users would perform each step.

Workspace Setup—This phase lets users set plan-level parameters and configure *Agents*, *Things*, and *Destinations*. One or two *Agents* (one human and/or one robot) must be defined. Users also specify *Things*, which include cubes, spheres, cylinders, and containers and can be customized with size and color. When *Agents* or *Things* are created, *Destinations* that specify their initial locations are automatically created. Additionally, users can specify new, unpaired *Destinations* as waypoints or goals. While configuring *Destinations*, users can inspect the robot action times for all possible *Destination* pairs in a table generated by the motion planner. If a *Destination* is unreachable, the robot time entry is marked invalid, prompting adjustment from the user. Users are able to adjust the placement of *Agents*, *Things*, and *Destinations* within a 3D simulation view or, alternatively, through manual entry.

Creating Tasks—Users develop their collaborative plans through a drag-and-drop mechanism. Users can create any number of *Tasks*, which will be executed from left

to right. As users are developing their *Tasks*, they may find that they are repeatedly creating *Tasks* containing the same sequence of *Therbligs*. Users create macros by exporting a *Tasks* as a template of parameterized *Therbligs*. When a user then drops a macro into a *Task* container, the macro expands back into a sequence of those parameterized *Therbligs*.

Adding Therbligs—Users can drag *Therbligs* from the source drawer and drop them into *Task* containers. *Therbligs* can be rearranged within and across *Tasks*.

Parameterizing—While the user is developing the task structure they may open a contextual menu by clicking on an element. If the element is a *Therblig* in the source drawer, then the contextual menu provides an informational description of the *Therblig*. Selecting a macro from the source drawer displays the sequence of parameterized *Therbligs* saved within. Clicking on a *Task* brings up the ability to export it as a macro. Finally, selecting a *Therblig* contained within a *Task* provides access to its parameters.

The *Therblig* contextual menu affords configuration of both high- and low-level parameters. High-level parameters (*Agents*, *Things*, and *Destinations*) are presented as icons with a drop-down list for configuration. All *Therbligs* request an *Agent* parameter and may also request *Thing* and/or *Destination* parameters. Unique to *Agent* parameterization is an option to defer to the allocation algorithm, presented as an *optimize* option in the *Agents* drop-down list. Low-level *Therblig*-specific parameters (*e.g.*, time, cost, effort) are presented when applicable. For time and cost, when a user provides a human as the parameterized *Agent*, the contextual menu simply requests the time parameter. However, when the *Agent* is deferred to allocation, both time and cost for the human *Agent* and cost for the robot *Agent* need to be specified.

The parameter view also provides the user with feedback on any errors associated with that *Therblig*. In addition to identifying missing parameters, the same Z3-based verification algorithm used in the allocation process is executed upon plan updates, and provides helpful error messages, *e.g.*, “*Agent* must not be gripping.” As a visual shorthand, the interface also indicates the presence of errors for a *Therblig* with a red notification icon within its tile.

Simulating—Users enter the simulation phase to evaluate their resulting *Plan*. On entry of this phase, *Authr* runs the *Agent* allocation algorithm on the designed *Plan*. With successful allocation, the user may start, pause, stop, and reset the simulation in real-time. Robot simulation is handled through MoveIt, and human simulation is simply linear interpolation between *Destinations*. Also shown in the simulation view is a timeline for each *Agent*'s allocated *Therbligs*. The timeline representation, *à la Interaction Blocks* (Sauppé and Mutlu, 2014), was chosen due to the inherent temporality that it affords. Clicking on a *Therblig* within the timeline will expand a context menu displaying its duration and cost. While simulating the *Plan*, the *Therblig* being performed in the 3D simulation by an *Agent* is also highlighted within the timeline.

If the user enters simulation with an invalid plan, the view is replaced with a list of errors detected. While the user is simulating the *Plan*, they may find that their *Therblig* sequence is not performing as they intended (*e.g.*, they forgot to indicate a *Transport Loaded* to a way-point *Destination*). The user may then switch to either the setup or planning phase to fix the error or refine their plan.

3.3 Evaluation

To gauge the ability of our technical solutions to support the creation of task plans for human-robot teams, we carried out two evaluation studies. The first study focused on our solution to the first technical challenge, creating a shared representation, and assessed the extent to which *Authr* provided users with an appropriate vocabulary to model tasks. The second evaluation focused on our solution to the second technical challenge, translating task models into human-robot task plans, and evaluated *Authr*'s ability to effectively allocate task steps to human and robot *Agents*. Both evaluations also measured the general usability of and user experience with *Authr*.

Evaluation 1: Shared Task Representation

The first evaluation aimed to assess the ability of our HTA- and *Therblig*-based framework to support the modeling of manual tasks as well as the general usability of the software. To achieve this goal, we asked engineers and engineering students to implement a simple kitting task using *Authr*. This evaluation used a version of *Authr* without the Simulate Mode and was constrained to manual allocation of *Therbligs*. This version provided a simulation view in setup where users could move the robotic arm for virtual kinesthetic teaching.

Participants

A total of eight participants were recruited from a university campus. All participants (5 males, 3 females) were native English speakers with an average age of 27.63 (SD = 21.61). They either held or were pursuing a degree in either industrial engineering or mechanical engineering.

Procedure

After providing informed consent, participants interacted with an early version of *Authr*, which lacked the simulation and optimization components considered in the later evaluation. Participants were shown a short 9-minute video explaining how to use the software and the different types of *Therbligs* they could use. This video walked users through designing a simple pick-and-place task with a single *Thing*. Next, participants watched a video of a human actor, see Figure 3.5, performing a kitting process with three different types of *Things*, and were then asked to implement that process as a *Plan* in *Authr* that was performed by a robot. While full simulation was not present in this version, participants were able to utilize a simulated robot (Universal Robots' UR5) for defining the locations of *Agents*, *Things*, and *Destinations*. After completing the task, participants received compensation at rate of \$12/hour.



Figure 3.5: Participants viewed a video of an actor performing a simple kitting task and used *Authr* to translate it to a human-robot task.

Measures

Participants were given as much time as they needed to design their *Plans* and were asked to verbalize their thoughts in a think-aloud procedure (Ericsson and Simon, 1998; Van Someren et al., 1994). After the task, users completed the System Usability Scale (SUS) (Brooke, 1996; Bangor et al., 2008), USE (Lund, 2001), and a short demographic survey.

Results

Video data was transcribed and coded for emergent themes. These themes are discussed below.

Theme 1: Planning and Strategy—All eight participants created generally similar *Plans*, with a few differences. Only one chose to group all their *Therbligs* into a single *Task*. The remainder chose to group their *Therbligs* into separate *Tasks*, based on the item being moved.

One participant switched from a single *Task* design to a three-*Task* design after setting up the first group of *Therbligs* in a *Task*. At the time, the singular *Task*

contained (*Transport Empty, Grasp, Transport Loaded, and Release*), as well as an additional *Transport Empty* which returned the robot back to its initial position to prepare for the next sequence:

So I suppose I could do 3 *Tasks*—that’d probably be pretty easy. *Grasp, Transport Loaded, Release*, go back to neutral position. I suppose that kind of makes “Only Task” [Name of the one *Task*] not make much sense. Let’s grab *Transport Empty, Grasp, Transport Loaded...* So that kinda makes this *Transport Empty* not make any sense to do, because then all I am going to do is say *Transport Empty* again right at the start of this [The next *Task*]. (P05)

In this excerpt, the participant made two adjustments. The first was the aforementioned switch to three *Tasks*, instead of one. In so doing, the participant also realized that the *Transport Empty* they were performing, which returned the robot back to its initial location, was actually unnecessary, as it was immediately followed by the first *Transport Empty* of the next *Task*.

When structuring their *Tasks* in this way, participants also tended to notice parallels between the *Tasks* they created, prompting many to comment or request some way to either loop through or copy *Tasks*:

It seems like I am doing the same actions over and over, so it would be nice if I could use the same *Task*. (P02)

Exporting *Tasks* as copy-ready macros had been planned but not implemented by the time of these evaluation sessions. These comments provided justification for adding this in the next evaluation.

Theme 2: Destination Configuration—The most commonly cited difficulty participants mentioned centered not around *Therbligs*, but rather the specification of *Destinations* in the 3D workspace. The challenge was that to move a *Destination* or *Thing* in the workspace, users had to click and drag various toggles around the entities. This challenge could be due to lacking a metaphor that they were familiar

with (Wingrave and LaViola, 2010). The controls were not immediately intuitive to users:

I am going to move...how do I...Oh that's not it. Um oh I see, OK. I didn't know how to move it at first, and now I see that you have to move it like mutually orthogonal in either of the 3 Cartesian directions. (P05)

In order to constrain the space of *Destinations* to the smaller set of valid *Destinations* for a given robot arm, we used a procedure in which users moved a marker around the scene, and the arm attempted to match that pose. Setting the position and orientation would copy the robot pose to the *Destination*, as a direct parallel to kinesthetic guidance (Muxfeldt et al., 2014) which could be used in a physical workspace to specify locations to the robot. However, this approach did not seem intuitive to users in this context, as suggested by the following excerpt:

But it is hard to know what moves what. I got it eventually. And then the robot it isn't super clear like where the base is and the where the head starts, and then you have to move the robot to set the *Destination*, which...And then like checking how far it can go—that didn't really make sense to me. (P03)

To address this confusion, for the final evaluated version of our tool, each *Destination* was manipulated directly, and an indicator showed when it was reachable by the robot *Agent*. However, for future versions of *Authr*, this capability may be added back in, for when users have a physical robot on the scene and wish to use the physical robot to configure the destinations and object locations more easily, much like interfaces such as Polyscope and RAZER (Steinmetz et al., 2019).

Theme 3: Simulation—A common comment by participants was that upon completion, many wished to confirm the accuracy of their *Plans* by seeing it in action through simulation:

OK, it looks like it is all good, but I do not know how to test it. (P06)

Indeed, one such participant made an error in their *Plan* that likely would have been discovered through simulation. In specifying the goal *Destinations* for the *Transport Loaded Therbligs*, they incorrectly used the initial locations of the objects as *Destinations*, instead of the goal location specified. This does not create an error, since a *Transport Loaded* to the same *Destination* is valid, albeit non-useful. The resulting *Plan* would have shown no movement of items in the scene to the goal location. This provided further justification for supporting iterative refinement through the design of *Authr*.

The quantitative data from the measures of usability and user experience can be seen in Figure 3.6. The sub-scales of USE had the following scores: Usefulness ($M = 4.56$, $SD = 1.35$), Ease ($M = 4.63$, $SD = 0.856$), Learning ($M = 5.81$, $SD = 0.579$), and Satisfaction ($M = 4.29$, $SD = 0.990$). The average SUS score was 67.3, ($SD = 14.1$).

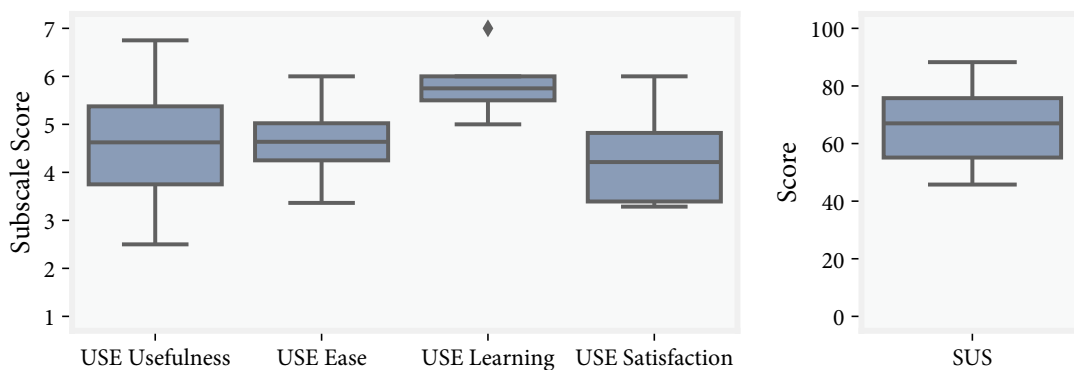


Figure 3.6: USE and SUS scores from Evaluation 1.

Evaluation 2: Agent Allocation

For the second evaluation, we turned our focus toward automatic *Agent* allocation. Specifically, we studied the ease to which participants author manual *Agent* allocation *Plans* in comparison to authoring automatic *Agent* allocation *Plans*. To

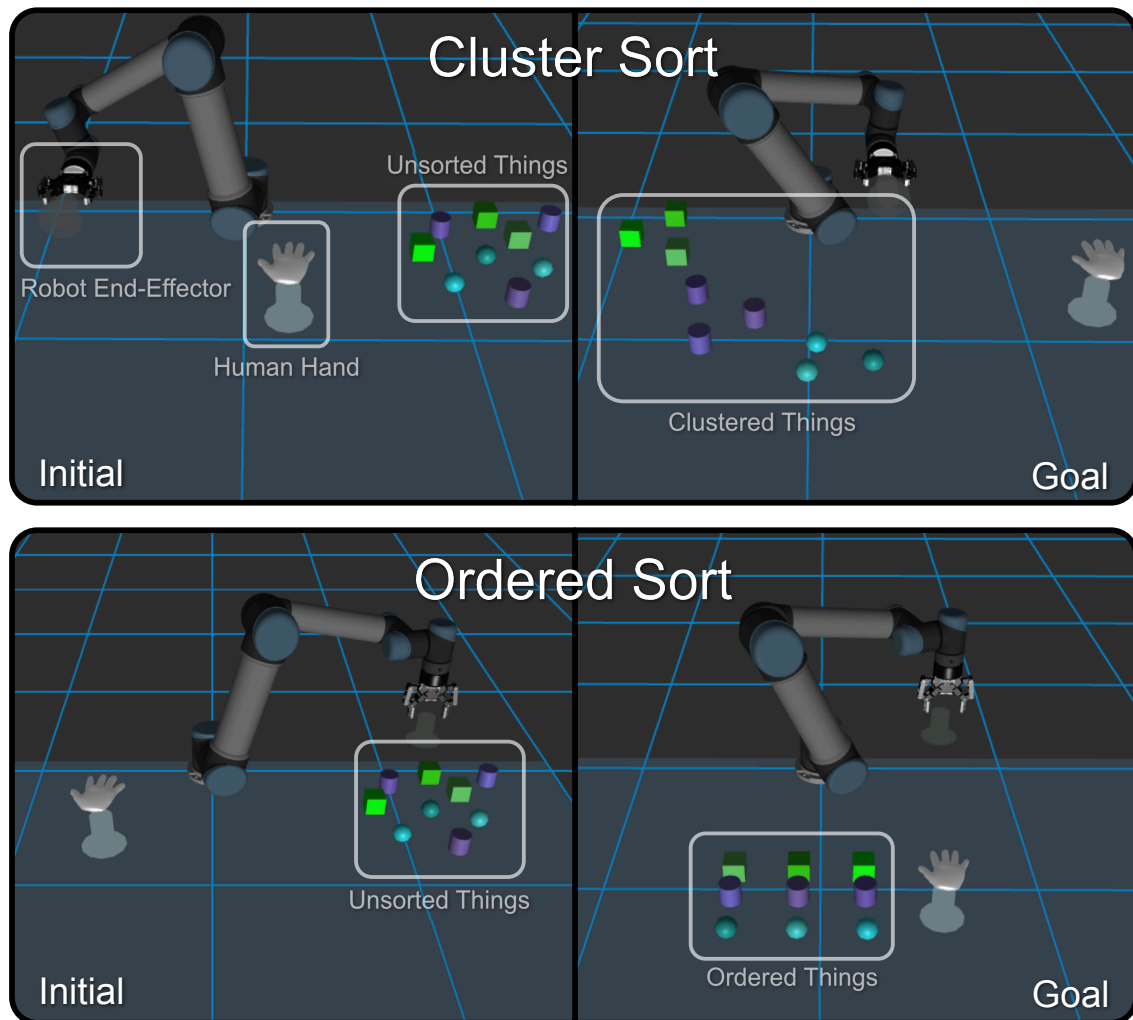


Figure 3.7: For Evaluation 2, we constructed 2 comparable tasks, Cluster Sort and Ordered Sort. For Cluster Sort, top, participants organized blocks into clusters by type, and in Ordered Sort, bottom, participants organized blocks into a grid.

understand how engineers may use *Authr* to perform these tasks, we started participants with a more complex sorting *Plan* where *Agents*, *Things* and *Destinations* are already defined. We then asked them to implement the *Tasks* necessary to complete the *Plan*. Once completed, the experimenter would load in a different sorting *Plan* and repeat the experiment with a different allocation type. The four conditions in

the experiment were counter-balanced.

Participants

Another eight participants (6 males, 2 females), aged 21 on average ($SD = 0.93$), were recruited for Evaluation 2. All participants were native English speakers and either held or were pursuing degrees in industrial or mechanical engineering.

Procedure

After providing informed consent, participants interacted with the next version of *Authr*, which added full simulation, verification, automatic *Agent* allocation, and the adjustments to the interface based on feedback from Evaluation 1. The version of *Authr* described in the *Technical Approach* section was used in this evaluation. A Universal Robots' UR5 was used for simulation and design. As in Evaluation 1, participants were first shown a short eight-minute video explaining how to use the software, and the different types of *Therbligs* they could use. This video walked users through designing a different basic pick-and-place task from Evaluation 1. Next, participants were instructed on how to view the robot action time table and use macros by the experimenter. The experimenter worked through one example, and the macro was deleted after demonstration. Two different sorting tasks, see Figure 3.7, were provided each with nine *Things* (three cubes, three cylinders, and three spheres) that needed to be moved to their goal state according to the condition. The *Things* had initial positions off to the right side of the simulated workspace. In both cases the initial positions of *Things* were identical. In the Cluster Sort, participants were asked to move the *Things* from the unsorted cluster into three sets of clusters, based on type. Similarly, in the Ordered Sort, participants were asked to move the *Things* into a grid formation.

To start the task, participants were provided a reference document containing the task description with the sorting objective, defined *Plan* workspace, a time estimate table for human actions, and cost tables for both *Agents*. Time and cost tables were the same between *Plans* except for robot timing due to differences in *Destination*

positions and orientations. When constructing a *Plan* in the manual allocation condition, the participant was asked to explicitly define the *Agents* without the allocation algorithm. The experimenter also provided the participant with scratch paper and a calculator, and instructed the participant to solve the allocation to the best of their ability. When constructing a *Plan* in the automatic allocation condition, the participants were instructed to only use the automatic allocation. For both allocation conditions, participants were given as much time as they needed to work through the *Plan*. Participants were also allowed to use simulation throughout their design process. Once they felt that their *Plan* was finished they were given a series of questionnaires assessing their experience with the tool, after which they would move onto the second case. Except for the sorting objective and method of *Agent* allocation, the procedure for the second *Plan* was the same as the first. After the participant completed the second *Plan* and the associated questionnaire, they received compensation at rate of \$12/hour.

Measures and Analysis

Participants were given as much time as they needed to design their *Plans*. As in Evaluation 1, participants were asked to verbalize their thoughts in a think-aloud procedure. After completing the task, users completed the SUS (Brooke, 1996; Bangor et al., 2008), USE (Lund, 2001), and NASA Task Load Index (TLX) (Hart and Staveland, 1988), as well as a short demographic survey.

To gain a more complete view of how users navigated the system, we also utilized a quantitative ethnographic data analysis approach called Epistemic Network Analysis (ENA) (Shaffer, 2017; Shaffer et al., 2016; Shaffer and Ruis, 2017) to analyze the usage data, particularly the sequence of actions and utterances by the engineers. ENA models the structure of connections in data and has a number of requirements, namely that (1) the data can be structured into meaningful features (Codes), (2) that the data has a local, or temporal structure, and (3) that the connections between these codes within that local or time-sensitive frame is important. ENA produces a weighted network of co-occurrences, along with associated visualizations for each

unit of analysis in the data. Critically, ENA analyzes all of the networks simultaneously, resulting in a set of networks that can be compared both visually and statistically.

While originally built to address challenges in understanding learning analytics, ENA has since been used in a variety of different contexts, such as gaze coordination during collaborative work (Andrist et al., 2015), and communication among health care teams (Sullivan et al., 2018). ENA is therefore an appropriate technique for any context in which the structure of connections between relevant information is meaningful. ENA is thus a useful in this domain because it can model the relationships among user actions and utterances as they utilize *Authr*, allowing us to better understand how engineers navigate *Authr* in the various conditions.

We started the epistemic network analysis by coding transcripts and user activities to generate a set of five informative codes, shown in Figure 3.8. One, *Planning*, was generated from transcript data, through the use of the nCoder online tool (Shaffer et al., 2015). During this process, an automated coder was trained until a $\kappa = 0.92$ was achieved between the human and automated coder. For the remaining codes, interface actions and states were utilized. Using these codes, we then applied Epistemic Network Analysis (Shaffer, 2017; Shaffer et al., 2016; Shaffer and Ruis, 2017) to our data using the ENA 1.6.0 web tool (Marquart et al., 2018). Units were all lines of data associated with either condition (Manual or Automated Allocation) for each participant. The ENA algorithm uses a moving window to construct a network model for each line in the data, showing how codes in the current line are connected to codes that occur within the recent temporal context (Siebert-Evenstone et al., 2017). A moving window was chosen at four lines (each line plus the previous three lines).

The ENA model normalized the networks for all units of analysis before they were subjected to a dimensional reduction, which accounts for the fact that different units of analysis may have different amounts of coded lines in the data. For the dimensionality reduction, we used a singular value decomposition (SVD), which produces orthogonal dimensions that maximize the variance explained by each dimension (see Shaffer et al. (2016) for a technical explanation of the method).

Code	Definition	Examples
Building	Adding to and constructing the plan. This includes adding tasks and subtasks, as well as configuration and weight-setting.	<ul style="list-style-type: none"> - Assigning agents to Therbligs - Configuring Therbligs with Things - Setting Therblig duration or effort
Planning	Verbal articulation by the engineer of their thoughts as they consider the problem space or devise strategies.	- “So for transport empty probably just want to use the robot because it costs less and it seems like it takes the same amount of time.”
ContextSwitch	Engineer switches between tabs or screens as they construct their plan.	<ul style="list-style-type: none"> - Switching from External Reference to the Plan tab - Switching from the Plan tab to Setup
ExternalReference	The action of switching to any external resource other than the <i>Authr</i> interface.	<ul style="list-style-type: none"> - Viewing the plan specification requirements - Using a calculator or notepad
InternalReference	Referencing information generated or defined in the <i>Authr</i> interface.	<ul style="list-style-type: none"> - Viewing Agent information - Viewing Thing locations - Looking up TOFs

Figure 3.8: Codes generated using system states and nCoder.

Networks were visualized using network graphs where nodes correspond to the codes in the model, and edges reflect the relative frequency of connections between the codes. This visualization results in two related representations for each unit of analysis, the first being a plotted point representing the location of each unit’s network in the low-dimensional projected space, and the second being a weighted network graph. The positions of the network graph nodes are determined by an optimization routine that minimizes the difference between the plotted points and their corresponding network centroids and then fixed. Since all network graphs are co-registered in the projected space, the positions of the network graph nodes, and the connections they define, can be used to interpret the dimensions of the projected space and explain the positions of plotted points in the space. Our model had co-registration Pearson correlations of 0.99 for the first dimension and co-registration Pearson correlations of 0.97 for the second, indicating that there is a strong goodness of fit between the visualization and the original model.

Results

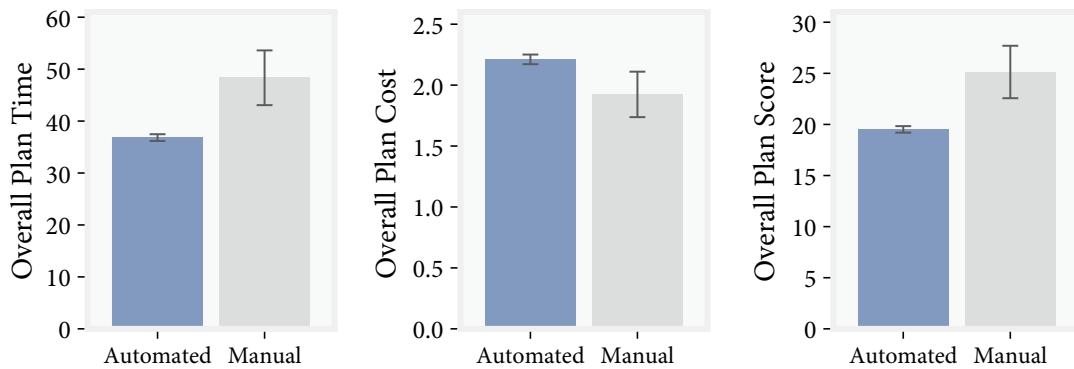


Figure 3.9: Resulting Overall *Plan* Cost, Time, and Scores for Automatic and Manual procedures. Cost refers to the objective corresponding to effort or wear (depending on *Agent*), and Score refers to the overall score, based on the weighted Time and Cost. Lower scores for all measures are more desirable.

Outcome measures were computed for each produced *Plan*: Plan Time, Plan Cost, and Plan Score. Plan Score reflects the weighted average of Plan Time and Plan Cost that the participants attempted to minimize. Additionally, scores for the SUS, USE sub-scales, and the NASA TLX sub-scales were computed. Each outcome measure was analyzed with a repeated measures one-way Analysis of Covariance (ANCOVA), modeling allocation method while controlling for *Plan* type (*Sorting* or *Ordering*). Plan Time, $F(1, 6) = 6.8274$, $p = .0400$; and Plan Score, $F(1, 6) = 6.9791$, $p = .0384$, were found to be significant, where Automatic procedures outperformed Manual in these cases. No other measures were significant. We note that these results are based on a small sample and require further validation and contextualization within qualitative findings.

Video data were transcribed and coded for emergent themes for each *Plan* implemented by the participants. Given the complexity of the *Plans* compared to Evaluation 1, engineers constructed *Plans* that showed greater variation. This included both boundaries for where the *Plans* were divided up into *Tasks*, and when

manually allocating *Agents*, how they chose to handle prioritizing duration and cost. Frequently, participants would express distaste for the workload when performing manual allocation. These themes are discussed below.

Theme 1: Workload—When performing allocation *manually*, many participants, after fully understanding the problem space, articulated negative attitude or frustration with the process. One such participant stated that they did not want to perform all the necessary calculations:

Okay. I don't really want to do all the calculations for figuring out exactly which ones. ...So many different possibilities. (P15)

This apprehension towards the problem resulted in a variety of approximation strategies, detailed below. Others expressed distaste for the work required to compare charts of times. In the following excerpt, a participant who first performed the automated allocation procedure remarks about how it differed from the manual process:

It's definitely less frustrating calculating the cost than it is trying to figure out the quickest path going through the the chart the first way...Cause like quick mental math, I personally liked that better than trying to stare at the chart...(P10)

When using automated allocation, participants simply had to provide accurate times for the human, and costs for the robot and human. While this certainly amounted to some tedious calculation of effort and data entry, as seen here, participants tended to prefer this to the complex considerations of the manual approach. We will discuss options for reducing this potentially tedious activity in the future directions.

That is not to say that the automatic method was without its difficulties. A small number of participants did initially have some trouble orienting to the more abstract representation of the task. This problem of representing these more abstracted, agent-agnostic actions is therefore in need of further research.

Theme 2: Strategies—Due to the workload required in the manual allocation procedure, most participants developed various heuristics or algorithms to determine the best allocation.

Some participants chose to approximate by looking at trends. In the following excerpt, one participant averaged rows in the robot times, to get a rough expected duration. This was compared to the human times:

...and with the sphere, all of the robot numbers look like they're kind of a little bit higher than three for where they're moving it to. Um, if they like start at the sphere and like move to it. So I was gonna make the human do that one. (P10)

Others took a more conditional approach, looking for cases where patterns held or did not hold:

What's the cost of moving the robot? The cost of moving the robot is always cheaper. Wait...If it's always cheaper... Yeah. Then if time is cheaper than cost should be cheaper for the robot. We should only consider it when...Since we're only working with these four and the robot's cost is lower than these. So anyway...yeah. Only if the human time is shorter then we take cost into consideration. (P16)

Heuristics weren't the only way to approach the problem, however. Other participants opted to iterate on a *Plan*, moving *Agents* from *Task* to *Task*. In some cases, participants would leverage the simulation to assist with this process.

Theme 3: Mistakes and Errors—As seen above, most participants, when manually allocating *Agents*, developed various heuristics to determine the best allocation. Unfortunately, not all these heuristics were valid, or consistently applied.

The excerpt from the discussion of workload also illustrates this line of thinking:

So I'm just going to put these as the robot because it seemed cheaper...Yeah. I'm just going to give the robot two of them. Sure. We'll say that's good. (P15)

This a common error seen in these heuristics. When participants used less concrete heuristics, they tended to place a higher-than-necessary emphasis on cost than described in the instructions. While cost was almost always lower for the robot (the exception was *Rest*), the cost was usually small compared to duration, while weights were equal.

Yet, even larger errors were sometimes made. In the following case, the participant incorrectly opted to assign all *Therbligs* to the robot, in order to minimize the overall cost, but forgot to consider benefits of concurrency:

I feel like I'm not going to use the human at all. I feel like the time that it's using—like the 0.66 seconds—will offset the cost of 0.2 versus 0.05 for the transport. That's my...I know...I'm going to go forward with moving only the robot. Cost seems like a better benefit. (P10)

This error resulted in a nearly two-fold increase in total *Plan* score (38.6) when compared to their automatically-allocated *Plan* (20.2). This illustrates the large potential for errors when dealing with such complex allocation tasks in a completely manual manner.

Theme 4: Modified Workflow— In addition to the differences in the tactics of problem solving, participants appeared to display a more streamlined reference → calculate → configure process in the automated condition and a more fragmented approach in the manual one. This was shown in the results of our ENA. In ENA, networks can be compared using network difference graphs, which are calculated by subtracting the weight of each connection in one network from the corresponding connections in another, visualized in Figure 3.10. To test the significance of the difference between graphs along the X axis, a two-sample t-test assuming unequal variance showed that the Automated Allocation condition ($M = -0.55$, $SD = 0.47$) was significantly different from the Manual Condition ($M = 0.55$, $SD = 0.79$; $t(11.42) = 3.36$, $p = 0.01$). There were no differences in the Y axis.

The differences in these network graphs illustrate the differences between the two conditions. As users constructed their plans manually, they had to assemble information from a number of sources (internally and externally) and use it to

construct a program that satisfied the requirements of the specification, while considering how their choices affected the time and cost of the program. Thus, there were strong connections in the network between *InternalReference*, *ExternalReference*, and *Planning*, as well as a stronger connection between *Planning* and *Building*. The relative importance of *Planning* in this network is important, as it corresponds to the cognitively demanding aspect of the design process, while *Building* corresponded to the actual process of articulating those choices in the program. In contrast, the network for the Automated Allocation condition is much more localized to a dyad of *Building*, *External Reference*, with *ContextSwitch* representing the action of switching between them. As such, engineers using this method were better able to directly translate the program specification into the implementation, without cognitively demanding overhead.

Qualitatively, we can consider the following process for *P16* as they configured a series of therbligs:

The participant begins by creating a series of empty *Therbligs*. *P16* then considers the scene from the *Setup* tab, decides to work on the second cylinder first based on its location, and then tentatively proceeds by assigning the robot, choosing to alter the plan later if desired. *P16* switches to consult the reference document (on a separate browser tab), but then needs to check where the robot arm would be. Next, the participant needs to check the table of robot times to see the time estimate based on that information and then switch back to the reference to calculate the expected time and cost parameters:

“What are we working with... the cylinder. Four seconds.”

Finally getting the information, *P16* switches back to the *Plan* view and enters the information. Throughout the process, the participant performs multiple context switches to reference different information from different sources (some internal to the *Authr* tool and some external), which requires the participant to frequently verbally re-situate after each context switch.

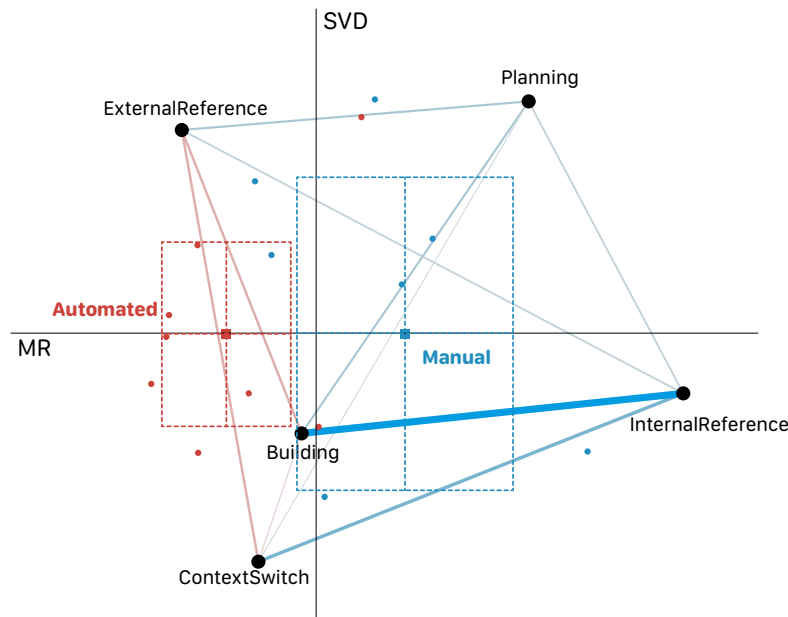


Figure 3.10: A comparison of the activity networks for Automatic Allocation (red) versus Manual (blue) conditions. Each network is displayed as both a network graph and box, indicating the mean and confidence intervals of the networks within the projected space.

Compare this pattern with the same participant configuring a plan using the automated process as follows:

The participant creates a series of *Therbligs* in a *Task*, configuring the high-level parameters for *Things* and *Destinations*. *P16* then switches to the reference document in the separate tab. The participant obtains the estimated time and cost parameters and returns to the *Plan* tab to enter the information.

Because the automated process utilizes its own information (*i.e.*, robot time estimates) in the allocation, participants did not need to reference, handle, or utilize this information, reducing the need to reference internal data. While the participant had to perform some additional calculations, as seen in the themes above, this

process was preferred over the manual approach. This preference was likely in part because of the more streamlined reference → calculate → configure process of the automated method.

3.4 Discussion

Our evaluation studies provided us with a better understanding of the extent to which the technical solution we have developed enabled users to create human-robot task plans. Our findings from Evaluation 1 indicate that the shared task representation that incorporated ideas from HTA and work modeling served as an appropriate framework to model tasks that users observed from video. The representation not only provided an effective set of building blocks to construct plans, but it also helped users identify *Therbligs* that were unnecessary in achieving the task goal. The evaluation also identified a number of usability issues, specifically in positioning and assigning *Destinations* for *Therbligs*, which informed the improvements we made in *Authr* prior to Evaluation 2. Finally, participants repeatedly expressed a desire to simulate the actions they were modeling, providing evidence that simulation is a critical component of any task-planning approach or environment.

The second evaluation, which focused on assessing the effectiveness of our technical solutions in enabling engineers to allocate task steps to a human-robot team, revealed that automated methods for such allocation is critical. Participants in our study were overwhelmed by the combinatorial complexity when considering the conjunction of task step ordering, cost, time, and *Agent* allocation. To handle this complexity, many participants developed heuristics to simplify the allocation problem, but these heuristics were ineffective or detrimental when they were not applied consistently or correctly. Finally, we found that designers using the automated approach showed a more streamlined workflow for designing and configuring their plans. Overall, our findings indicate that automated methods that handle the complex computational process of task assignment to multiple *Agents* are essential for any task-planning environment.

Limitations and Future Directions

Authr offers a novel approach to the design of collaborative robotic programs, but it does have certain limitations that motivate a number of future research directions.

First, our solution to the shared task representation problem only focused on physical manipulation tasks, although real-world tasks will require additional capabilities such as perception and tool use. The original set of *Therbligs* that informed the development of our solution proposed a set of *cognitive Therbligs* such as *Search* and *Inspect* as well as a set of *cognitive and physical Therbligs* such as *Use* (Gilbreth and Gilbreth, 1924). In our future work, we plan to extend the current set of *Therbligs* with the ability to perform cognitive actions and tool use. The *Use Therblig* in particular is worth exploring as it can potentially handle a large number of tools and be applicable outside of manipulation-type tasks. The challenge with operationalizing *Use* in an agent-agnostic manner is that a semantic description of tool use is generally enough for humans to act but is insufficient for robot instrumentation. That is, the tool’s behavior would need to be algorithmically defined. We suggest two potential ways to approach the operationalization of *Use*. In the first approach, *Authr*’s high-level parameters could be extended with *Tools*. With well-defined behaviors, the *Use Therblig* may be sufficiently descriptive as it could utilize the capabilities each tool defines. An alternative approach is to decompose *Use* into a larger number of *Therblig-like* actions (e.g., *Use-Screwdriver*), each addressing a different behavior.

We designed *Authr* on the extendable infrastructure called ROS. While our current solution does not directly connect to physical robot devices, a minor configuration change and the inclusion of an additional (usually standard) driver would allow for this capability, since the control is based on inverse kinematics and simply publishes joint instructions. Our evaluations have thus far not tested this implementation, but in future work we plan to test the inclusion of a physical device as a component in the *Authr* task creation and evaluation processes. This plan includes the potential to use the physical robot as an input device for object and goal specification via PbD, as well as a more grounded output for simulation.

PbD could also be investigated as a way of simultaneously accumulating data on human activity time and effort, potentially reducing the amount of parameterization required.

Our solution also does not address the complexities of variations in object attributes, positions, and counts that would be encountered in complex, real-world tasks. Some of these problems could be modeled by extending our simulation process to account for variance, resulting in more robust time estimates. Furthermore, future work should incorporate computer vision and sensing capabilities to respond to this variation and the movements by the human collaborator in real-time.

Low-level parameterization of *Therbligs* is also currently limited to manual entry of values except robot action time. Several options to provide sensible default values instead of relying on user entry could be explored. First, *Things* can provide relevant information such as grip-effort based on their intrinsic properties. Second, a human simulation could be used to generate time estimates similar to the current approach for robot *Agents*. Lastly, *Authr* could allow users to specify cost functions algorithmically, taking into consideration time, weight, grip effort, and so on, providing engineers with flexibility to define their own metric without the burden of manual entry.

A shop-floor human-collaborator mode for *Authr* should also be explored. Currently, *Authr* generates a static plan in simulation with the assumption that human *Agents* can follow it precisely. Future work should investigate methods of informing workers on their current goals, tasks, and future robot-collaborators actions. How human collaborator's task performance and cognitive load are affected while performing *Plans* developed in *Authr* alongside a physical robot should also be studied. Achieving these would require *Authr's* representation to handle both human synchronization and errors not addressed with the static programs currently generated. This behavior can potentially be exposed to engineers using either *cognitive Therbligs* or introducing more general programming control flow. *Authr* should also be extended to provide safety awareness such as minimum separation monitoring during execution and considered when allocating *Agents*.

Finally, our user evaluations modeled simple tasks (ordering and sorting) that

may not represent the complexities of real-world tasks, *e.g.*, an assembly task. Our technical evaluation of the *Agent* allocation algorithm used tasks derived from more realistic situations (kitting, assembly, and repair), demonstrating greater representational capability. In the future, we plan to carry out evaluations of *Authr* with engineers from the local industry in their own environments (instead of our laboratory) and ask them to create human-robot plans for the tasks that their organizations perform. We expect such evaluations to provide us with guidance on how to extend *Authr's* capabilities to further support complex real-world tasks.

3.5 Chapter Summary

The *Authr* tool presented in this chapter represents our first attempt to take the concept of human-robot task allocation algorithms and translate them into a tool that can be used by engineers to create human-robot task plans. As shown in this research, a major issue is that in any design process, a multitude of features and metrics must be managed in the task specification (authoring) process. This is partially a matter of knowledge, as seen in the ways that users had to frequently reference materials or respond to errors identified by the verification engine. It is also a workload issue, where individuals failed in their attempts to create reliable heuristics for balancing multiple competing objectives.

These challenges motivate the type of solution we presented, which leveraged an approach to optimization-based allocation based heavily on that of Pearce et al. (2018) and *Therbligs* (Gilbreth and Gilbreth, 1924). This solution was a new tool aimed at improving the types of collaborative plans that engineers could produce, providing end-to-end support for human-robot task teaming. To better understand this process, we have evaluated *Authr* with engineering students across two user studies. Our findings from these evaluations indicated that the shared task representation that we developed served as an appropriate framework for modeling existing tasks, and our automated agent-allocation approach facilitated the translation of these task models into plans that humans and robots can collaboratively perform, offloading the complexity of various logical constraints, as well as the

multi-objective balancing, thereby improving the workflow for users, as well as the quality of the output.

Whereas *Authr* sought to improve the workflows for engineers, and improve the overall process for creating human-robot task plans, the next chapter will focus on improving both the knowledge of the user through scaffolding, and the quality of the interaction with rich feedback through the use of a combined programming-learning environment called *CoFrame*.

4 COFRAME

More than a third of the facilities that use robotic technology employ collaborative robots (cobots) (Miller, 2021) and cobots deployed within the manufacturing context are expected to continue to grow in market share. Industry seeks to mitigate labor shortages (Autor, 2021) while improving work-cell performance and reducing human worker’s health risks. To this end, a new generation of manufacturing robots designed to work in a shared space alongside human operators as collaborators are replacing conventional caged robots. Work traditionally done by a human can be parceled into tasks that consider the skill-sets uniquely brought by humans and cobots (Pearce et al., 2018). Although much research and engineering effort has been done to bring these robots into the work-cell, the training-procedures, tools, and practices to support human operators have lagged behind (Michaelis et al., 2020). This lag results in a “skills gap” for operators working alongside cobots without the knowledge and skills to customize the robot’s behavior to better accomplish the task (Wingard and Farrugia, 2021).

Research has identified specific occupations such as craft work, where the skills gap in utilizing robotic technology is most pronounced (Holm et al., 2021), and how individuals differ in their specific skills and preferences regarding the use of cobots (Giannopoulou et al., 2021). Other research has sought to better understand this skills gap, specifically to address the question, “what do workers need to know in order to effectively utilize these systems?” Siebert-Evenstone et al. (2021) interviewed experts in collaborative robotics—including engineers, implementers, and trainers—to identify the skills, tools, and perspectives they utilize in troubleshooting and programming cobots, developing an “expert model” of collaborative robotics. Such models serve as opportunities to develop training, programming, and control interfaces for HRI research. In order to address the skills gap in collaborative robotics, we developed a digital training environment called *CoFrame* that aims to prepare traditional and non-traditional students as operators of cobots. In this chapter, we present the model of expert skills and knowledge that

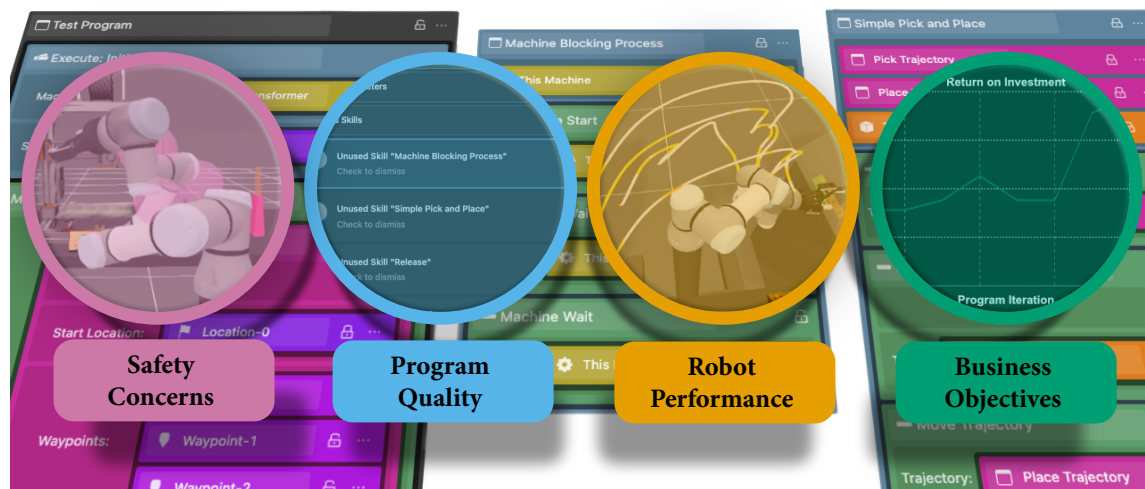


Figure 4.1: In this chapter, we describe a system called *CoFrame* that integrates a set of *Expert Frames* in collaborative robotics, focusing on *Safety Concerns*, *Program Quality*, *Robot Performance*, and *Business Objectives*, to train operators in using, programming, and troubleshooting cobot applications.

serves as the basis of our automated expert feedback system illustrated through several system capability case studies.

The contributions described within this chapter include¹:

- An operationalization of the *Safety First* expert model as *Expert Frames* to address the cobot skills gap;
- Design and implementation of a design-learning environment that incorporates the *Expert Frames*²;
- A set of case studies demonstrating system behavior and how it provides feedback during authoring/learning.

¹The research discussed in this chapter is derived from published work by myself and Nathan White, Curt Henrichs, Dr. Amanda Siebert-Evenstone, Dr. David Shaffer, and Dr. Bilge Mutlu. All authors contributed significantly to the conceptualization, design, implementation, evaluation, analysis, and/or the writing of the original manuscript.

²Code available at <https://github.com/Wisc-HCI/CoFrame>

Target Users

In this work, the design of *CoFrame* was meant to more specifically target users classified as automation experts, who have experience in traditional robotics, but not necessarily in collaborative robotics. As stated above, these individuals have many skills that are required in this adjacent domain, but would benefit from additional training and support to better bridge this gap as they work to implement collaborative robotics in their workspaces.

Research Questions

CoFrame, unlike *Authr* takes a different approach to addressing its research questions. While work on *Authr* focused on specific assessments that evaluated the effectiveness of representations and systems, work on *CoFrame* focused on the question of how one might translation an empirically defined model of expert knowledge into a functional training and programming system with rich feedback and assistance. It also considers what processes might be involved in its use through a set of hypothetical case studies. In line with its limitations, it does not evaluate the effectiveness of the system as a whole, which instead is left to future work.

4.1 Background

In this section, we review related work on recent developments in collaborative robotics, the skills gap that results from the introduction of these technologies into workplaces, interfaces for cobot authoring and programming, and the state of the art in cobot training systems.

The Emerging Field of Industrial Collaborative Robotics

Since their introduction to the market in late 2000s, cobots have found widespread adoption across industries, including manufacturing (Simões et al., 2020), logistics (Lappalainen, 2019), and medicine (Ernst and Jonasson, 2020). Research in the last

decade has investigated the safety and ergonomics of the use of cobots (Matthias et al., 2011; Fryman and Matthias, 2012; Gualtieri et al., 2021), how work can be structured to enable humans and robots to work together (Shi et al., 2012; Pearce et al., 2018), and how cobots can be integrated into production lines (Wojtynek et al., 2019; Horst et al., 2021). Key insights for the success of cobots from this body of work include the promise of cobots to improve both the efficiency and ergonomics of some manual processes (Pearce et al., 2018) (mostly in medium-sized production volumes (Fast-Berglund et al., 2016)); the need for establishing well-defined levels of collaboration (Christiernin, 2017; Shi et al., 2012) and forms of task interdependence (Zhao et al., 2020); and the importance of employee-centered factors such as the fear of job loss and ensuring an appropriate level of trust in the robot (Kopp et al., 2021). Overall, this is a rapidly emerging field involving the development of new technology to enable the integration of robots into work environments; study their safety, ergonomics, and effectiveness; and work toward understanding of how they affect human workers.

The Emerging Worker Skills Gap

Cobots offer many benefits across several industries, including productivity benefits to organizations and health and safety benefits to human workers, but the introduction of advanced technologies, particularly technologies involving automation, robotics, and advanced interfaces, into workplaces is creating a “skills gap”—a gap between the skills necessary to utilize these technologies and the skills of the existing workforce (Ras et al., 2017; Michaelis et al., 2020; Wingard and Farrugia, 2021). In the U.S., nearly half of the job openings, totaling 2.2 million positions, in manufacturing remain open due to a shortage of workers with the skills necessary to effectively utilize such technologies (Giffi et al., 2018). In the context of robotic technologies, the skills gap exists at all levels, from robot operators to researchers (Shmatko and Volkova, 2020). Although education and training have been proposed as the primary means of closing this gap (Chrisinger, 2019), a recent analysis of existing educational programs found a lack of emphasis on critical technical and

non-technical, or “soft,” skills in these programs (Andrew et al., 2020). Despite showing that industry lacks the appropriate means, including curricula, materials, and knowledge, to offer such training, this analysis also highlights the importance of work-based, hands-on training and apprenticeships. Our work aims to capitalize on this promise by creating a training system that situates the learning in a real-world or simulated work environment.

Robot Programming Tools and Environments

An area of collaborative robotics where the skills gap is significant is the programming of cobots for new tasks (El Zaatari et al., 2019). Existing approaches to addressing this gap primarily involve the development of intuitive and ease-to-use robot programming that borrow ideas and concepts from end-user programming, such as the RoboFlow and Code3 visual programming languages (Alexandrova et al., 2015; Huang and Cakmak, 2017), and the application of these approaches to the programming of industrial robots (Weintrop et al., 2017). Evaluations of the effectiveness of these approaches to enable adult novices to program cobot applications show them to be more effective, usable, learnable, and satisfactory compared to the existing cobot programming interfaces (Weintrop et al., 2018). Research into end-user programming tools also include highly advanced robot programming tools that enable semantic skill demonstrations (Steinmetz et al., 2018, 2019), task allocations to human-robot teams (Schoen et al., 2020), and AR-based interfaces that leverage workspaces as augmented programming surfaces (Perzylo et al., 2016; Gao and Huang, 2019; Senft et al., 2021b,a). However, these systems target intuitive and rapid programming of robots and do not address the skills gap by advancing the skills of the user in programming and troubleshooting cobot applications.

Robotics Training Systems and Programs

Prior approaches to addressing the skills gap in robotics highlighted the unique challenges of working with robotic systems, including manipulating real-world entities using software programs and situating these skills into real-world problems.

For example, Dagdilelis et al. (2005) developed a program that integrated visual programming to teach robot programming concepts to high-school students. Cobot systems have also been explored as a medium to teach students at the college level engineering design (Ziaeeefard et al., 2017). To address the challenge of situating learned skills in real-world problems through hands-on learning, prior work has proposed the concept of a “Teaching Factory” that offers a factory-like classroom environment (Mavrikios et al., 2013; Chryssolouris et al., 2016). These environments offer trainees genuine systems, constraints, and problems to work on and opportunities to interact with both instructors and practitioners. Prior research in situating learning in genuine environments also includes the development of virtual- and augmented-reality based learning environments that enable trainees to perform work tasks and processes (Matsas and Vosniakos, 2017), although these systems aim to train workers in collaborating with robots rather than providing the skills necessary to program and troubleshoot them. Cobot manufacturers provide training programs targeting specific skills necessary to utilize their products, e.g., Universal Robots Academy (Universal Robots, 2021). These programs are used in vocational training (Słowikowski et al., 2018), although effectiveness of these resources to address the skills gap is unknown, and experiences of early adopters of cobot systems indicate that they are not sufficient (Michaelis et al., 2020). Some research has explored methods for translating expert knowledge to robot operators (Fantini et al., 2017), although these methods have not been applied in training systems.

We previously discussed opportunities and challenges in collaborative robotics, particularly the need to address the skills gap that has become a bottleneck in the widespread adoption and utilization of cobots. Although the growing body of research in end-user programming tools can make it easier for workers to use cobots, addressing the skills gap requires new training programs and technologies that can help workers obtain expert problem-solving skills and apply them in real-world settings. Our training system, *CoFrame*, aims to address this need for cobot operators.

4.2 Expert Model

Collaborative Expert Model

Our implementation of the *Expert Model* relies on the findings of an ethnographic study by Siebert-Evenstone et al. (2021) regarding how experts think about cobot application design. They found that expert thinking falls into a *Safety* + structure. Specifically experts keep *Safety* concerns (collaborative/shared space collisions, pinch-points, risk-assessment, force sensing, tool/part manipulation) in mind while considering other aspects of the program, such as *Performance Objectives* (cycle-time, speed, payload), *Business Objectives* (robot wear-and-tear, cost, ROI, efficiency), and the *Application* (problem-solving, flexibility/adaptability, robot reach, human interaction, positioning).

Experts bring a deep systematic understanding to their application design to balance a variety of critical safety points with concerns of cost and flexibility, and usability. They examine the use case to determine if a cobot is preferable over a traditional robot, which specific robot(s) to deploy, what sensors and integrations are needed, the process structure, and how that impacts safety. Traditional manufacturing robots are either physically caged or use sensors to detect entry into an exclusion zone, but cobots are designed to safely work around the human operator with appropriate programming, provided the integrated end-effector tooling and workspace are also safe.

In designing the application, experts weigh the cost of the engineering challenge for a robot to manufacture the part versus having a human operator perform the activity. Experts typically deploy operators in low interaction roles such as setup, starting robot, inspection, and ending the process; and emphasize that they should avoid getting in the way of the robot. Experts consider where the human operator is within the workspace, their role, and how they are trained to perform it. They design their applications to make use of external sensors (vision systems) and tools (conveyors, CNC mills). When an error occurs they have knowledge of what it means and are able to reason about appropriate solutions.

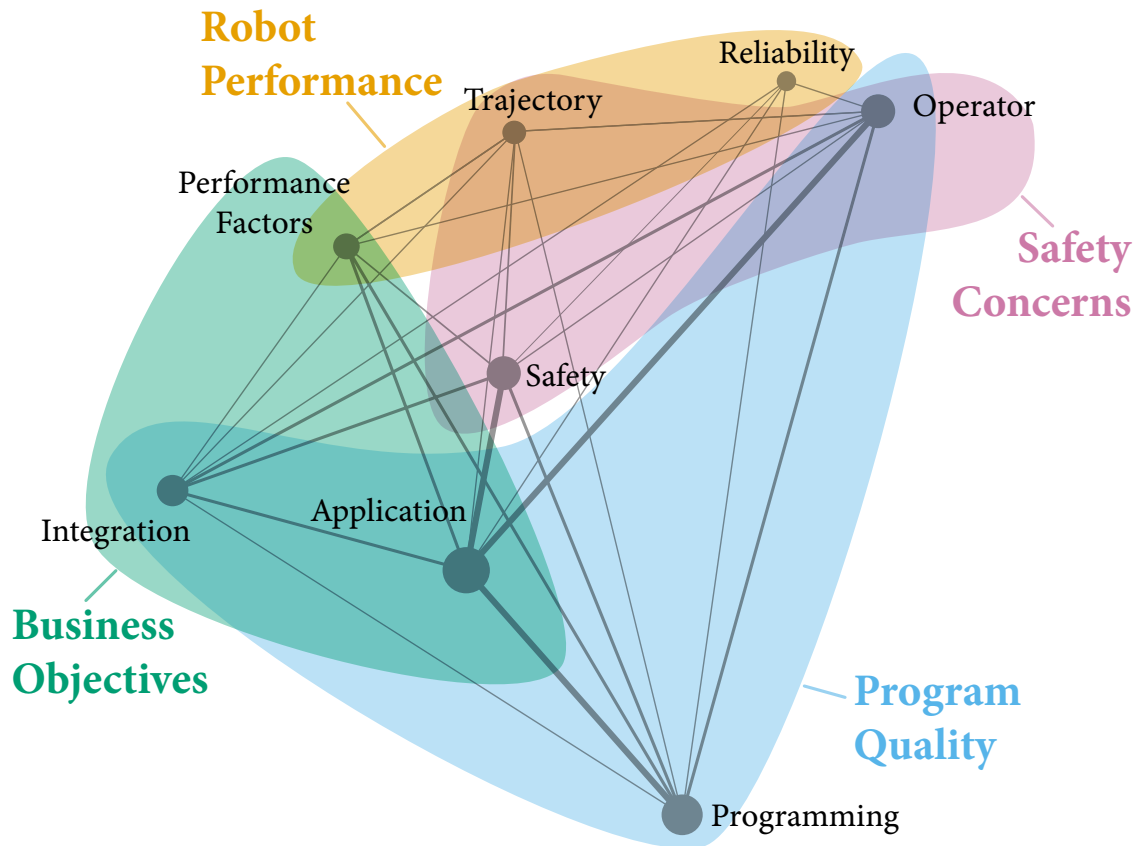


Figure 4.2: The mapping of the themes from the *Expert Model* (Siebert-Evenstone et al., 2021) into each of the four *Expert Frames*: *Safety Concerns* (pink), *Program Quality* (blue), *Robot Performance* (yellow), and *Business Objectives* (green). Figure adapted from Siebert-Evenstone et al. (Siebert-Evenstone et al., 2021).

To translate the *Expert Model* developed by Siebert-Evenstone et al. (2021) into a form compatible with learning outcomes, we reorganized these concepts into four *Expert Frames*: *Safety Concerns*, *Program Quality*, *Robot Performance*, and *Business Objectives*. This mapping can be seen in Figure 4.2. Each of these frames represent a lens with which to assess the robot collaboration.

Expert Frames

Safety Concerns This frame is heavily based on the *safety* theme from the *Expert Model* but incorporates aspects of the *operator* and *trajectory* themes. The *Expert Model* focuses on safety, going so far as to place it above the others in terms of importance. One expert indicated that “I could buy a collaborative robot, but if I’m moving around steak knives, it’s no longer collaborative, so there’s no point to using a collaborative robot” (Siebert-Evenstone et al., 2021). Thus, the collaboration incorporates the orientation and safety of the robot’s tool, the safety of the individual objects that the robot may be carrying, whether the robot’s trajectories include possible pinch points or collisions, the robot’s space usage during the program, and how that interacts with the human’ collaborator’s space. Key to this frame is providing clear and concrete feedback about the safety of the resulting program; as individuals with less cobot programming experience may not design a task to be safe, unaware that they are violating certain safety heuristics (Siebert-Evenstone et al., 2021).

Program Quality Several themes from the *Expert Model*, including *programming*, *integration*, *application*, and *operator* combine to create this crucial and practical feedback frame. Many other frames depend on proper specification of the program to provide meaningful feedback. This frame includes simple program attributes, such as the parameter satisfaction, and more complex ones, such as how the robot must adapt to the duration of various machine processes. Specifically, we evaluate the program based on missing parameters and code blocks, unused skills and features, empty code blocks, and any logical issues regarding integration with machines.

Robot Performance With a focus on robot execution quality and ability to perform actions, this frame includes aspects of *performance factors*, *reliability*, and *trajectory* from the *Expert Model*. In many cases, these performance metrics relate to the other frames, especially *Safety Concerns* and *Business Objectives*, and include qualities such as reachability, the speed of the joints and end-effector tool, payload, and space use.

Business Objectives These outcome-oriented feedback metrics guide the design by enabling operators to consider how their changes to the program affect the profitability of the robot, or how wear-and-tear might be impacted. This frame is informed by the *performance factors*, *application*, and *integration* themes from the *Expert Model*, and focuses specifically on cycle and idle times of the robot, and the return on investment.

Frame Relationships

In addition to specifying themes commonly discussed by cobot experts, the *Expert Model* describes the relationships among them. As already noted, expertise is usually identified not so much by the knowledge of isolated facts or heuristics, rather by a deep and complex understanding of the relationships between them within a domain (Chi et al., 1981; Council, 2000; diSessa, 1988). We operationalized these relationships by linking the individual sections for frames to other sections within or outside the parent frame. In some cases, these section relationships are practical. For example, a coherently defined program with regards to machine logic (*Program Quality*) enables calculation of total cycle time (*Business Objectives*). In other cases, they reflect the logical sequences of expert concerns. For example, determining whether a certain robot pose could introduce pinch points (*Safety Concerns*) is only relevant after considering whether the robot can reach the pose (*Robot Performance*).

In our *Expert Frames*, we use these dependencies and relationships to guide the operator to work through the logical dependencies whilst developing the connections and associations between them. A full list of dependencies between the sections of each of the frames can be seen in Figure 4.3.

4.3 System Design & Implementation

We developed *CoFrame* using the *React* framework that communicates with a back-end running a PyBullet (Coumans and Bai, 2016) simulation. Visualizations are

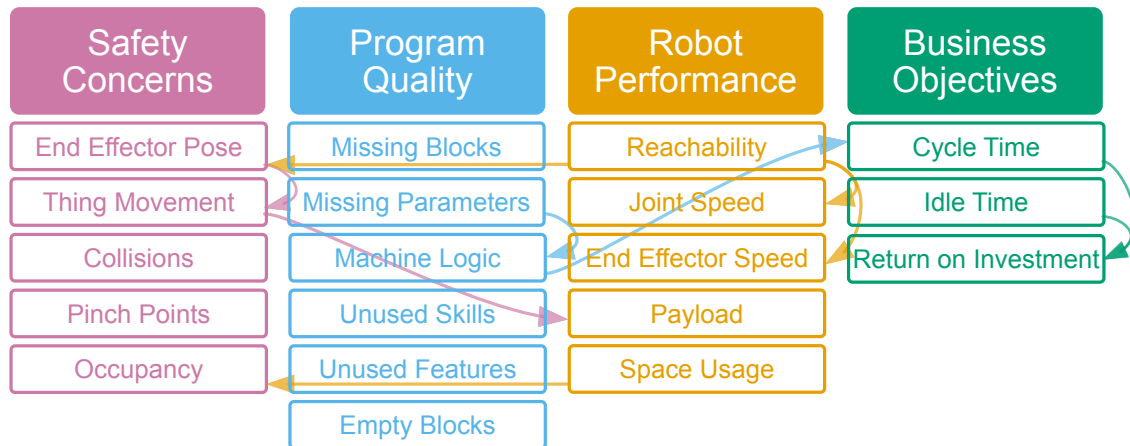


Figure 4.3: The four *Expert Frames* of *CoFrame*, and the relationships between them. As operators address concerns in each frame, they unlock other considerations. For example, only after addressing whether a Location or Waypoint is reachable (*Robot Performance*), do they address issues with the pose of the end effector.

rendered in *Three.js*. Our system is designed around four “tiles” as shown in Figure 4.4: the Program Editor tile (G), the Simulation tile (B), the Contextual Information tile (C), and the Expert Frames tile (A).

Programs and Program Editor

We implemented a block-based visual programming language for operators to build their programs, heavily inspired by other commonly used tools like Blockly (Fraser, 2015) and Scratch (Resnick et al., 2009).

Code Block Design

We utilized a drag-and-drop mechanism that allows an operator to easily construct viable programs (Fig. 4.4 F) and visualize the connections between different blocks. This is accomplished by dragging blocks from the block drawer (Fig. 4.4 D) into the canvas (Fig. 4.4 E). Each block type is given a distinct color and icon in order to assist in this visualization process. As operators build their program, they can

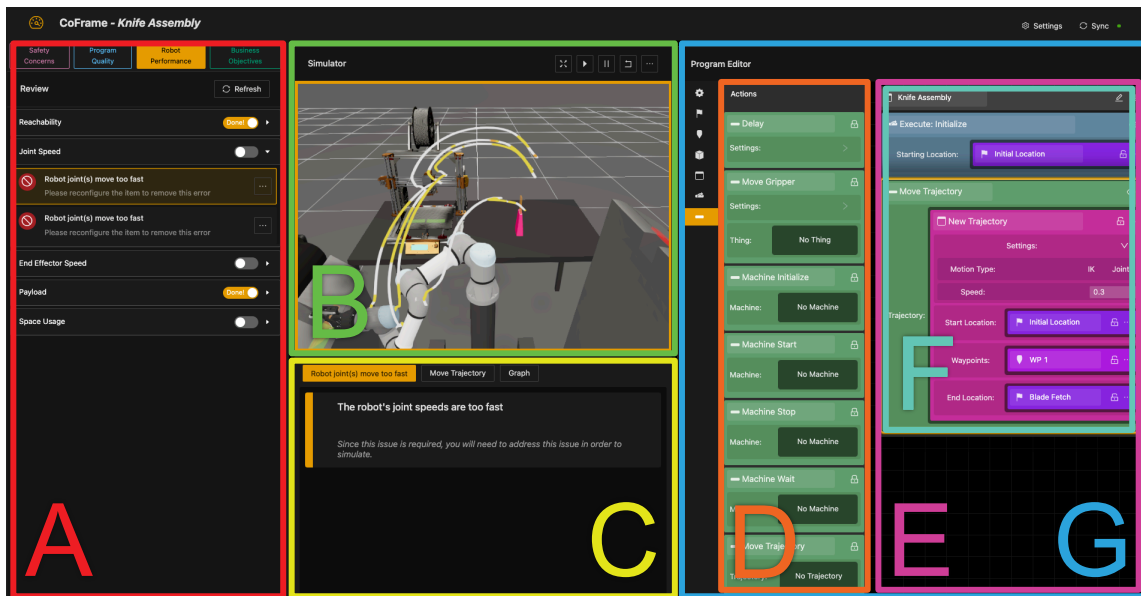


Figure 4.4: The layout of the *CoFrame* interface. Operators can use the Program Editor tile (G) to construct their program, and can visualize the results in the Simulator tile (B). The Expert Frames tile (A) allows them to swap between different *Expert Frames* and view issues in each frame. When not viewing issues, the Contextual Information tile (C) shows relevant frame-related information, and when viewing issues also provides detailed information about the issue and suggestions for changes. Within the Program Editor (G) operators can drag blocks from the Block Drawer (D) into the Program Canvas (E). The Program Canvas contains the program (F) along with implemented skills.

highlight blocks in the program editor to receive more information in the Contextual Information tile, as well as visualize the action in the Simulation tile if fully specified.

Block Types

There are two main categories of blocks—item and executable. Item blocks refer to objects in the workspace (e.g., things, trajectories, machines, locations, and waypoints). These item blocks are used to parameterize the executable blocks, such as actions like “Move Gripper” (accepts the *thing* being gripped or released by the tool), “Move Trajectory” (accepts a trajectory item block), or “Machine

Start” (accepts the machine to start). Some actions take additional numerical or configuration parameters, such as movement speed or gripper position (e.g., in the “Move Gripper” action).

Machines are item blocks that create and modify *things* (parts and materials the robot interacts with) within the program. In our simulation, these machines are the 3D printer, conveyors, and the assembly jig. They specify recipes with inputs, outputs, and processing times. Things are objects that can be produced as output of machines, consumed as inputs, and moved by the robot. They also specify various properties, such as weight and safety (e.g., a blade is unsafe to carry unsheathed).

Locations and waypoints are positions in 3D space that represent where the user may want the end effector to be, both in terms of translation and rotation. Locations are presented to the operator as places where the robot would start and stop its movements, such as the end of the conveyor machines or the assembly jig, and are used in trajectories. Waypoints are used to specify intermediate positions between locations, usually to guide the robot to perform more desirable motions. Trajectory blocks accept parameter item blocks (locations and waypoints). They represent the motion the robot will execute; beginning at the start location, navigating through the an ordered list of waypoints, and stopping at the end location. They are also parameterized with movement speed and interpolation type (inverse-kinematic or joint-based).

Executable blocks within *CoFrame* fall into three categories: actions, groups, or skills. Actions, the smallest indivisible units of code, accept various parameters specifying their behavior. Actions that affect robot state can be simulated when selected in the program editor and fully parameterized.

The “Delay” action stalls the program for an adjustable amount of time, allowing the operator to explicitly specify a time that the robot is inactive, e.g., to allow the human to perform a task. Similarly, “Breakpoint” actions are a debugging action that stops the execution of the program at that block.

The “Machine Initialize, -Start, -Stop, and -Wait” actions all take in a *machine* parameter. “Machine Initialize” is equivalent to turning on the machine, and needs to be performed before any other machine actions. “Machine Start” begins the

associated recipe with any matching *things* required. “Machine Stop” indicates that a machine has completed but cannot be executed until after processing from the paired “Machine Start” ends. When completed, the output *things* are available to be interacted with. “Machine Wait” allows the operator to specify that the robot waits for whatever time remains on a machine process and guarantees that the “Machine Stop” does not occur before the minimum processing time of the machine has completed.

Regarding robot control, operators have access to the “Move Gripper”, “Move Trajectory”, and “Move Unplanned” actions. “Move Gripper” allows the user to manipulate the gripper to either grasp or release a specified target *thing*. It takes additional parameters for the desired final distance between the gripper fingers, and its movement speed. “Move Trajectory” takes in a *trajectory* parameter for the robot to execute, and moves the robot along the specified motion. “Move Unplanned” takes in a *location* parameter and is a way for the operator to specify starting locations or human-driven operation.

“Group” blocks are action blocks that function as a container for other actions, allowing the operator to group actions into coherent code blocks. These can be collapsed to reduce visual clutter and be previewed in the simulator when fully specified.

“Skill” blocks behave similar to functions in other programming languages, defining a set of parameters, a context of use for them, and action blocks that are executed. “Skill Call” blocks are generated for each “Skill” block in the operator’s program. Like other action blocks, these blocks accept item parameters and pass them along to the paired “Skill” blocks.

Simulation

The Simulation tile visualizes the robot; its movements, actions, the environment; and frame-based or issue-based feedback. At the onset, the simulation visualizes the robot going through the execution of the program. The simulation also connects with the Expert Frame tile to provide any extra visual information while showing

the relevant robot animation.

Contextual Information

The Contextual Information tile provides operators with frame-specific feedback and suggestions about selected blocks and items. The displayed information changes based on what they select and interact with in the Simulation, the Program Editor, and the Expert Frame tile. This includes definitions for terms and phrases novice operators are exposed to, frame dependant information, as well as various graphs for selected issues. Definitions are provided for words that are commonly used by experts and within robotics programs as well as explanations for how they relate to other terms.

Based on the selected frame, the Contextual Information tile provides additional prompts to help the user think about the concepts within each frame. For example, when adding waypoints to trajectories with the *Safety Concerns* frame selected, it will show, “Pay special attention to placing waypoints around the occupancy zone of the human, since this is more likely to result in undesirable conflicts between the human and the robot.”

Expert Frames

Prior work highlighted the need for programming environments to provide users with a comprehensive list of *issues* and to automatically collect and display information about program execution (Ko and Myers, 2005). The design of the Expert Frame tile (Fig. 4.4 A) builds on these guidelines, automatically providing feedback about the operator’s program through *issues*. Each issue represents a unit of feedback regarding an expert concern for a particular element of the program, and is complimented with textual or visual data to give information about how the program was executed. For example, when viewing issues about pinch points, the corresponding “Move Trajectory” block in the program editor is highlighted, and the operator is presented a simulation view of the robot moving through a trajectory that highlights the pinch points to draw attention to the issue.

Issues are marked as either warnings or errors. When marked as warnings, they are displayed in the Expert Frames tile with gray icons. When marked as an error, they are displayed in red. Warnings and errors also differ functionally, as warnings can be manually marked as fixed, allowing the operator to address other issues in the Expert Frames tile, while errors require the operator to make adjustments to their program.

Safety Concerns

End effector pose issues refer to cases where the gripper moves quickly in the direction of its fingers. Each trajectory timestep is scored and shown as a graph in the Contextual Information tile when the issue is selected. The simulation displays an animation of the robot moving through the trajectory along a line marking gripper trajectory, which is colored to visually indicate the ratings of the different portions of the trajectory. Operators are first required to address any *reachability* issues before addressing end effector pose.

Thing movement issues represent cases where the robot moves potentially unsafe objects through the space. While progressing through the program, checks are made for whether gripping the *thing* is possible given the orientations of both the gripper and *thing*. If in the program the robot executes a “Move Trajectory” action whilst carrying an unsafe object, it is flagged with an error and visualized in the 3D scene.

Each valid trajectory is analyzed with PyBullet collision detection (Coumans and Bai, 2016) to detect potential pinch points. These are visualized as moving dynamic spheres placed around the robot as it moves along the trajectory, such that larger, darker spheres are higher priority.

Collision issues occur when a trajectory causes the robot to collide either with itself or the environment. When selected, the corresponding trajectory is highlighted in the program editor, a graph depicting the robots proximity to itself and the closest object in the environment, and the simulation displays an animation of the robot moving through the trajectory along with lines marking the path each of the robot’s

linkages take. Darker colors along the lines indicate closer proximity.

Occupancy issues refer to instances where a robot trajectory overlaps with the human occupancy zone and share the same visual cues as collisions issues. Occupancy issues have a dependency on *space usage*, as the higher space usage correlates with an increased likelihood of entering the occupancy zones.

Program Quality

Missing block issues identify cases where the operator has failed to supply a necessary block where needed, such as in a “Move Trajectory” action. Similarly, missing parameters refers to instances where the code block does not have all the required parameters.

Machine logic issues identify where the program specifies invalid interactions with machines, for example when a machine is stopped before it finishes processing. Before an operator can address machine logic issues, they must first address any missing parameters.

Unused features help operators identify cases of unused item blocks. Similarly, unused skills refer to operator-defined skills that are not called in the executed program. When these types of issue are selected, the corresponding unused block is highlighted in the program editor.

Empty blocks refer to instances where either the program, a group, or a skill does not contain any actions. Selecting this type of issue highlights the empty block in the program editor.

Robot Performance

Reachability issues refers to instances where the robot is not able to move to a given waypoint or location because a solution cannot be found, or the position is out of the robot’s reach. When selected, the corresponding waypoint or location is highlighted in both the simulation and program editor, and a window opens allowing the operator to adjust the waypoint or location’s position in the environment.

Joint speed issues are instances where the robot's joints exceeding a threshold value for a trajectory. When selected, the corresponding trajectory in the program will be highlighted, a graph of each joint's speed over time is displayed in the Contextual Information tile, and an animation of the robot executing the trajectory with lines for each of the joints colored by their speed is shown in the simulation. Before operators are able to address these issues, they are required to first fix any issues with *reachability*, as the execution of trajectories is dependent on reaching the locations and waypoints along the way. End effector issues function similarly to joint speed, instead showing the speed of the end effector. When selected, it shows similar visuals to joint speed issues, with a graph and animation of the end effector. For similar reasons to joint speed, operators must first fix any issues with *reachability*.

Payload issues occur when the robot lifts *things* that approach or exceed its carrying capacity. If such a violation occurs in a "Move Trajectory," it is highlighted in the program editor and simulation. Payload issues require operators to first address issues with *thing movement* to fix other non-safe manipulations.

Space usage issues refers to the percentage of the robot's workspace utilized at any point during a given trajectory. When selected, the program editor highlights the corresponding trajectory; the Contextual Information tile displays a graph of how the utilization changes over the course of the trajectory; and the simulation shows a convex hull of the trajectory.

Business Objectives

Cycle and idle time issues are always generated and are encoded as persistent warnings. Cycle time refers to the total time that it takes for the robot to complete the operator's program once, while idle time refers to just the amount of time a robot is spent idling during program execution. Similarly, return on investment (ROI) issues are always generated and refer to the ratio of product value to total cost building the product, including the cost of robot wear-and-tear - which is based on the robot's acceleration. When these issues are selected, the Contextual Information

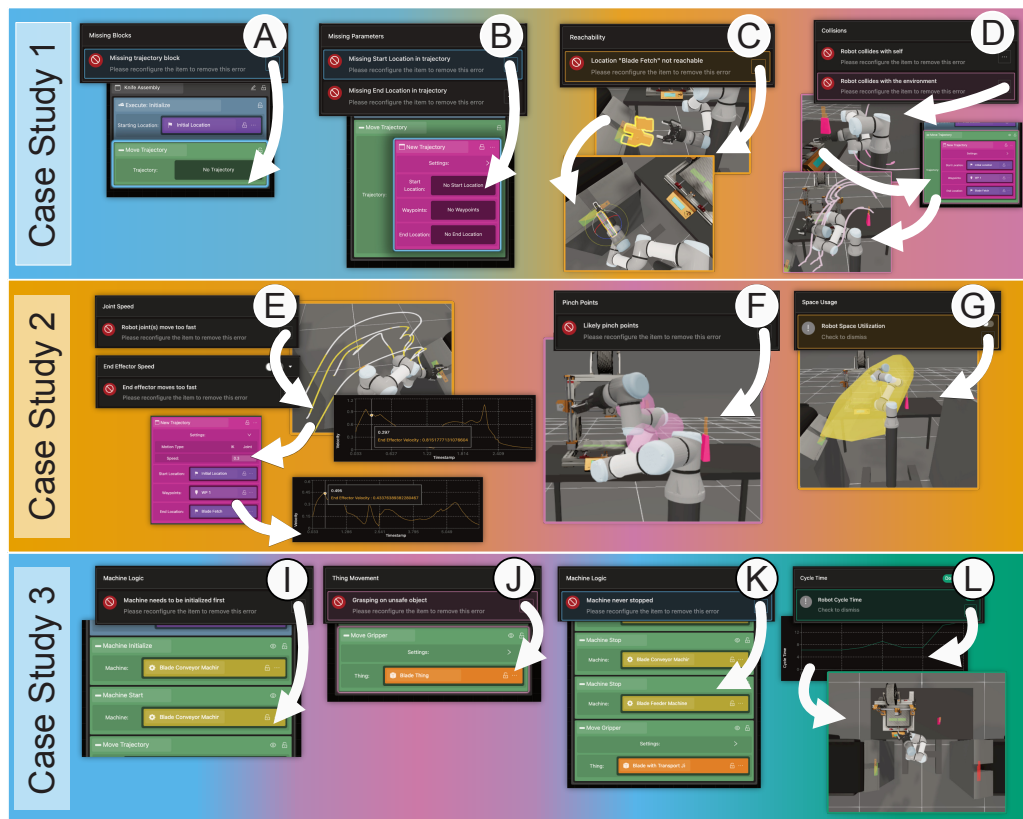


Figure 4.5: Three case studies showing the process of evaluating feedback from the system and informing adjustments to the operator's program. The gradient background of the figure denotes the switching between *Expert Frames* by the operator, from *Safety Concerns* (pink), *Program Quality* (blue), *Robot Performance* (orange), and *Business Objectives* (green). In Case Study 1, the operator begins by addressing a missing trajectory block (A), followed by filling in its parameters (B). The operator then addresses reachability concerns (C). They finish by addressing issues with robot collision (D). In Case Study 2, the operator begins by addressing joint speed issues and visualizes the speed (E). They transition to solving pinch point issues (F). They finish by addressing issues with the robot's space usage (H). In Case Study 3, the operator begins by solving issues with uninitialized machine logic (I), then addressing problems with thing movement (J). They return to addressing a machine logic for a non-stopped machine (K). The operator finishes by viewing the robot's cycle time (L).

tile displays a graph showing how the time changes as the operator adjusts their program. ROI issues require operators to first address both cycle and idle times issues, as ROI is dependent on them.

4.4 Case Studies

To demonstrate the behavior of *CoFrame* as a learning-programming environment, we developed three case studies that illustrate how it detects and responds to issues generated by the operator to support learning. To accomplish this, we specifically designed a task to prompt certain issues that the operator will have to address. The task is based on an expert's comments (Siebert-Evenstone et al., 2021) and has the robot assembling a knife from a set of components, namely a blade and two halves of the handle. The knife itself is unsafe to carry and the system will notify the operator if attempted. They will instead have to use a safety transport jig that covers the blade making it safe to carry. Blades arrive from a conveyor, handles are produced with a 3D printer, parts are assembled in a jig, and the finished knife is deposited on another conveyor. Figure 4.5 depicts each of the case studies detailed below.

Case Study 1: Defining a trajectory

One of the first substantive actions an operator will attempt is creating a trajectory to move the robot. They would likely start by considering the 3D scene and inspecting each machine to observe where the robot needs to move first. They will see that they need to move to the blade receiver, which catches the blades as they arrive from the conveyor. Then they open the program editor's block drawer and drag a "Move Trajectory" block into the program. They click on the "Refresh" button in the Expert Frames tile refreshing the program, showing a number of errors. They realize the action requires a "Trajectory" block. They refresh the feedback and are prompted to parameterize the trajectory with a start and end location. Unfortunately their end location is unreachable. They click on the issue to bring up the location for

editing and notice the robot got stuck in a joint state preventing it from reaching the location. The operator adjusts the location. *CoFrame* then displays the new joint state that aligns the gripper with the location. At this point, the operator refreshes. They see the trajectory has been fully specified. However, the trajectory causes collisions with both the environment and itself. They add a waypoint, guiding the robot above the table and avoid colliding with itself. After a last refresh, the collision issues have been downgraded to warnings.

Case Study 2: Debugging a movement

After specifying a syntactically valid trajectory, an operator may want to evaluate its performance. They click the *Robot Performance* frame showing active issues on joint and end effector speeds. Clicking the joint speed issue plots lines for each joint position through time in the 3D scene. The operator also clicks the end effector issue and observes the graph in the information section. The operator tweaks the speed parameter to resolve the issue. They switch back to the *Safety* frame to address pinch point violations. The operator adds a waypoint to better coax the robot to a joint state that prevents the issue. After adding a waypoint, they click the feedback “Refresh” button to update the trajectory visualization. The operator is now prompted to address robot space usage for the trajectory. They notice that the robot extends out into the workspace more than necessary so they again tweak the waypoints; iterating over speed, collision, and pinch point concerns.

Case Study 3: Working with machines

The operator revisits the scene to consider the machines’ operations. They click on the blade conveyor and see that it “produces blades,” which they want to move to the assembly jig. They open the block drawer and place a “Machine Start” action after their trajectory; parameterizing the action with the machine’s item block. They then place a “Machine Wait” action. Refreshing the feedback, they see an error that the machine needs to be initialized before use. The operator adds the necessary action block, then adds a “Move Gripper” action parameterized with the blade,

followed by a second “Move Trajectory” action. They iterate over the new trajectory in a similar fashion to the first one, though they add and adjust waypoints before seeking frame feedback.

After refreshing the feedback, they encounter a thing movement issue called “Grasping unsafe object.” Realizing the issue is with the blade being grasped, the operator focuses on the simulation to find the blade receiver machine, which converts a blade and a transport jig into a safe blade with transport jig *thing*. They then add a new set of machine actions (initialize, start, and wait) to the program. Refreshing feedback, they find an error “Machine never stopped” for both the conveyor and receiver. They add the necessary actions and inspect program operation. Curious about the *Business Objectives* frame, they toggle the frame and inspect the cycle time. The operator views the graph in the Contextual Information tile, and decides to find a more optimal program. They tweak the order of the blade conveyor “Machine Start” to happen before moving the robot to the blade receiver, reducing cycle time.

4.5 Chapter Summary

Multiple industries currently face a skills gap in effectively utilizing cobots in the workplace. Designing cobot applications requires considerable expertise that few workers currently have, presenting difficulties for companies looking to incorporate cobots alongside their human workers. Critically, workers generally lack the expertise to effectively construct, adapt, and debug cobot programs. However, this situation also presents opportunities for job creation if effective instruction methods can be created which narrow this skills gap.

To better understand what content these methods must communicate and teach, research by Siebert-Evenstone et al. (2021) identified a set of themes that form a *Safety First Expert Model* of cobot expertise. We translated this model into a set of *Expert Frames* that can be used to instruct novice cobot programmers in the content and relationships of the *Expert Model*. Next, we presented an implementation of a combined learning-programming environment that provides interactive textual and visual feedback for operators’ programs in accordance with the *Expert Frames*.

Finally, we provided a set of case studies that illustrate the pathways through these *Expert Frames* that operators may trace, thereby creating and reinforcing associations of content within the *Expert Model*.

The process of performing this translation and development was informative as well. While the majority of relationships between the content of each frames is derived from the *Expert Model*, a number of others arose naturally from the design of the system and the requirements of providing feedback. For example, a number of *Program Quality* attributes (e.g. full parameterization) are required as a necessity of deriving higher-level feedback on things like *Machine Logic* and *Cycle Time*. Furthermore, designing frame-based feedback at multiple levels of detail and at various levels of program completion requires clear, concrete, and data-driven outcome measures.

At a high level, while the *Authr* project focused on the organization of work, and worked to support decision-making by the cobot programmer, *CoFrame* focuses on the design of the program itself. The *CoFrame* system supports the operator in the construction of their program, with special attention to the overall safety, efficiency, and quality of the interaction. Recognizing that automated planning algorithms may not be fully sufficient to balance all these needs, it aims to build the skills of these cobot programmers, so that when hand-crafted behaviors need to be implemented or analyzed, they develop the intuition and expertise to do so.

In the time since the original publication of this work, the algorithms used by *CoFrame* have been improved, aiming to address some of the limitations discussed in the original paper. For example, improvements have been made to the underlying behavior compilation algorithm to both improve speed and portability, but also robustness with regards to consecutive trajectory movements. Whereas the original algorithm could fail when using variables in trajectory endpoints, the new algorithm can handle these cases by tracking the dependencies and changes of individual code blocks, selectively refreshing those in need of it. The portability of the system has also been improved, removing the requirement of a ROS server backend, thereby making the system easier for individuals to initialize and use. This change was made possible by the development of a new motion synthesis algorithm built for

both web-based and server-based systems, called *Lively*, which will be discussed in the next chapter.

Other limitations still persist, however. The system still does not address issues like slip and uncertainty related to gripping real-world objects, nor does it integrate with physical robot systems. The latter was a choice made to balance the intended motivation of the system with the gain in functionality. Building support for physical robots, while certainly increasing the utility of the system, presents a significant amount of additional work and complexity, given the variety of robot models and control methods. Focusing on usage with physical robots also presents certain accessibility issues as well, since not everyone may have a robot with which to do this testing and development, or individuals may want to test certain types of robots before purchasing them. Given the focus on learning and skills development, we felt that the benefits of a web-based system outweighed the benefits of a physically situated system at this time. Future work, however, could always revisit this feature set.

5 LIVELY

As robots increasingly work in human environments, they will need to execute a wide range of highly configurable behaviors while communicating effectively with their users. A worker collaborating with a robotic arm may have preferences for how the robot positions itself when they are nearby (Lasota and Shah, 2015). A collaborative robot assisting a person unloading the dishwasher might use slight movements of its gripper to communicate that it is ready to pick up or receive items (Strabala et al., 2013). A social robot may display idle motion with its body to indicate that it is active and lifelike (Michalowski et al., 2006). When conversing, a robot may look away to signal that it is thinking (Andrist et al., 2014). Prior research in human-robot interaction has found such “lifelike” motions to improve perceptions of the robot (Terzioğlu et al., 2020; Sauer et al., 2021). Thus, lifelike motion or configuration of a robot’s links and joints are key design elements for robots utilized in human environments. Successful execution of combined tasks and social actions requires balancing these types of goals with practical concerns, such as avoiding collisions and maintaining smooth motion.

In this chapter, we explore how *Lively* can support the generation of lifelike but feasible task motions for collaborative and social robots.

Since physical task-based activities are frequently spatially rooted in the workspace, robot control requires converting these Cartesian goals into joint-space instructions. For example, the ability of a robot’s arm to deliver an object to a collaborator depends on its ability to first reach the position of the object, and then travel to the person’s outstretched hand. Similarly, a social robot may point in a certain direction using referential gestures by changing the position and orientation of its hand or gaze. This conversion is commonly achieved with an approach known as *Inverse Kinematics (IK)*. Conventional IK approaches structure this conversion as a search in joint-space constrained by the position and orientation of the robot’s gripper. This approach encourages solutions exhibiting desired position and orientation goals on the gripper, but cannot guarantee finding a solution in all cases.

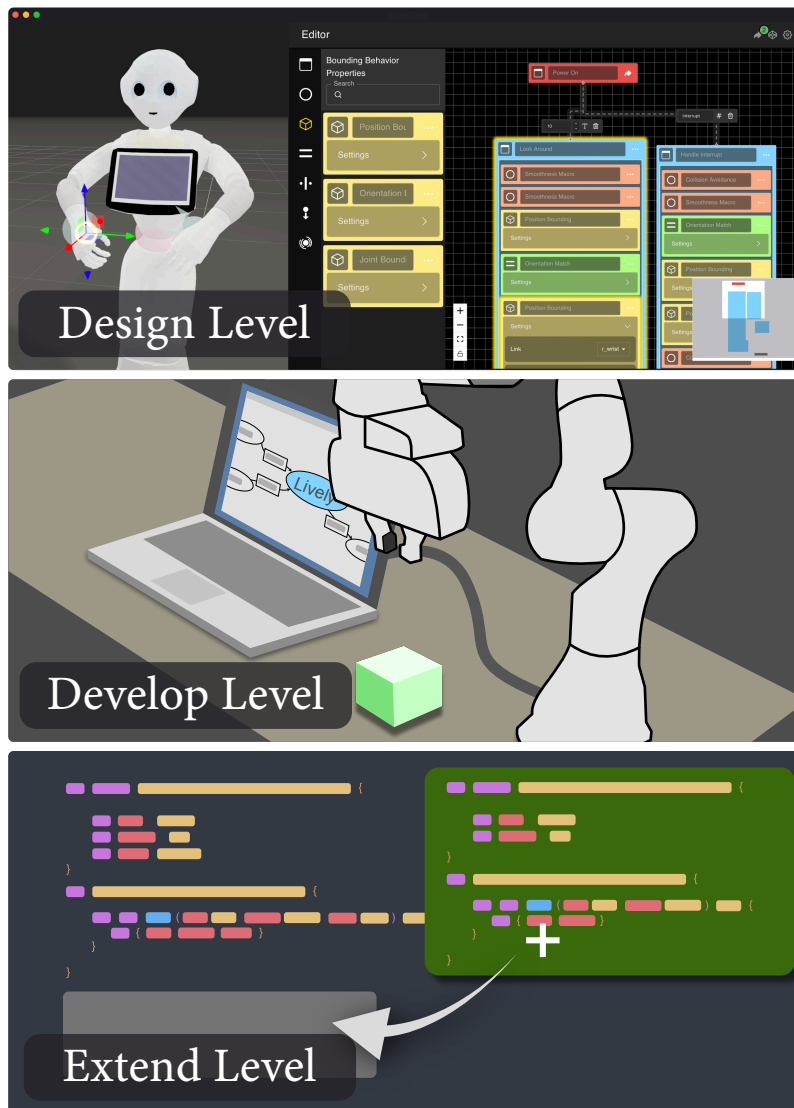


Figure 5.1: We present *Lively* for real-time motion generation that balances task and communicative goals while maintaining feasibility. We provide three levels of interfaces to address varying use cases. The *Design Level* enables programming robots using a state-based approach. The *Develop Level* is configurable and portable, usable in applications such as ROS-based control and web-based simulation. The *Extend Level* supports the addition of new characteristics and goal specifications for greater customizability and extensibility.

To communicate certain attitudes or states with physical motion, such as the human-robot interaction scenarios discussed above, the entire kinematic chain may be required, so simply considering the position and orientation of the gripper is insufficient.

Combining these social and task-based goals into functional robot motion requires not only knowledge of how motion is interpreted but also the technical ability to translate those qualities onto robot platforms. While robotics application developers may possess skills in both areas, domain experts may not have the same level of technical ability to bring their vision to fruition. An interface that is intuitive to both roboticists and other experts, such as animators, artists, or designers, can bridge this divide. Additionally, novel approaches to designing and implementing robot motion may be needed as robot capabilities evolve. Therefore, a design system with the flexibility to grow with these new approaches is required.

We present a new motion specification and generation framework, called *Lively*, that combines task-based and social goals while maintaining kinematic stability in real time (Figure 5.1). The framework leverages Perlin noise (Perlin, 1985, 2002) and integrates an existing per-instant pose optimization tool called RelaxedIK (Rakita et al., 2017) to achieve both primary and secondary motion goals in real-time. To support robot-application designers and developers, we developed three levels that expose the capabilities of *Lively* to users with different needs and levels of expertise. At the first level, *LivelyStudio* provides users with less technical ability an accessible, interactive, visual interface to design primary and secondary motions and control modalities used with the robot. At the second level, we present a development- and execution-focused framework, and at the final level, we provide an architecture that supports extendability and customizability.

The contributions of our work are summarized as follows¹:

- A *visual interface* called *LivelyStudio* that allows designers to interactively

¹The research discussed in this chapter is derived from published work by myself and Dakota Sullivan, Ze Dong Zhang, Dr. Daniel Rakita, and Dr. Bilge Mutlu. All authors contributed significantly to the conceptualization, design, implementation, evaluation, analysis, and/or the writing of the original manuscript.

construct state-based robot programs ²;

- An *open-source robot-agnostic library* that can be used by developers to specify real-time robot behavior that combines goal-oriented joint-space or Cartesian control with motion quality attributes in a feasible manner³;
- A modular software architecture that supports straightforward augmentation and contribution for custom control.

In the remainder of the chapter, we review previous approaches to this problem, contrasting them with *Lively*. We discuss the implementation of *Lively* and outline its cases for use along three different levels of programmatic accessibility, including the design of a tool called *LivelyStudio*, iteratively designed with a formative evaluation with roboticists and animators.

Target Users

In contrast to the *Authr* and *CoFrame* systems, the design of *Lively* and *LivelyStudio* was instead meant to address a different set of target users, namely those with experience in motion design, such as individuals from the fields of animation, gaming, and even dance. While perhaps not as familiar with the technical aspects of robotics, these individuals nevertheless have a great deal of relevant expertise that can be of use in this domain. That being said, a key feature of the *Lively* system is that it operates at multiple levels of accessibility, thereby allowing similar representations and models to be used by individuals with different levels of expertise. One could imagine scenarios in which a designer might use *LivelyStudio* to create a set of motions, and then a developer or roboticist might use the *Lively* library to integrate those motions into a larger system and applications.

²Code available at <https://github.com/Wisc-HCI/LivelyStudio>

³Code/Documentation available at <https://github.com/Wisc-HCI/lively>

Research Questions

The evolution of the *Lively* and *LivelyStudio* systems was an iterative one, notably a key formative evaluation featuring a set of experts from the fields of animation, gaming, and robotics. This formative evaluation asked the question of how to best balance the goals of matching the needs and expectations of these target users, while also conveying the capabilities the underlying system and the design considerations specific to real-time robotics applications. Like *CoFrame*, formal summative evaluations represent future work. Due to the proximity of this design space to art, an important consideration is also how summative evaluations may be performed on such systems in general. For example, what is the crucial outcome, the process for the user, or the quality of their result? Is a discrepancy between their vision and the result a result of the system, or the limitations of the more structured and rigid robotics environment? While these questions for evaluation are not answered in this chapter, they underscore the need for more discussion within this space.

5.1 Background

In this section, we review related work on expressive and functional motion including lifelike motion, inverse kinematics, and the operationalization of each.

Lifelike Motion

Whereas primary motion is an intentionally performed behavior, such as the process of handing a letter to a friend, standing in place, or looking to the right, secondary motion is defined as activity resulting from that primary motion (Johnston and Thomas, 1981). Secondary motion covers a wide range, such as the rippling or creasing of one's shirt as the arm is outstretched, the idle shifting of posture while standing, or slight movements of the pupils.

Secondary motion is known to be highly important to how humans interpret animated or robotic characters. In their paper, Heider and Simmel animated a

set of shapes to perform choreographed motions, such as following one another and moving into boxes while exhibiting additional subtle affine and rotational movements (Heider and Simmel, 1944). Most participants viewing the animation described the behavior of the simple shapes in human or anthropomorphic terms. Similarly, work with puppets has informed our understanding of how small motions and characteristics can influence viewers (Duffy, 2003; Duffy and Zawieska, 2012). The effectiveness of secondary motion motivated its inclusion in the principles of 3D animation by Lasseter (1987).

Animation utilizes many principles for secondary motion and lifelike behavior, initially requiring hand-drawn or hand-animated specification of behaviors. However, a growing number of methods make this process less demanding. Witkin and Popovic (1995) proposed a method to warp a keyframe animation to match new spatio-temporal constraints by systematically mapping underlying motion curves. This allows an animator to adjust a character's posture from happy to sad throughout an animation using only a sparse set of inputs instead of enumerating keyframes. Gleicher (1998) presented a method that maps motion from one articulated figure to another, even if they have vastly different scales or geometries. The method uses non-linear constrained optimization to minimally displace an input motion (e.g., motion capture data) to match the specifications of the new articulated figure. Additionally, motion has been added to computer generated characters using Principal Components Analysis (Egges et al., 2004), and traditionally animated characters have been augmented with secondary motion through a 3D intermediate process (Jain et al., 2010). However, these solutions all represent post-hoc methods of adjusting input motions, and are allowed certain freedoms given their virtual, non-rigid context.

Considerable work has also focused on effective ways of augmenting agents and characters with secondary motion in a generative manner. The most common method to do this was created by Ken Perlin (Perlin, 1985, 2002). Originally designed for texture generation, Perlin noise was quickly adopted for motion as a way of creating personality in animated characters (Perlin, 1995; Bodenheimer et al., 1999). Perlin noise is particularly well-suited for this domain, being a non-

repeating, but smoothly changing generative method. Furthermore, by modifying the speed at which the input value (usually a function of time) changes, animators can predictably control the characteristics of the noise function. By using smooth noise, such as Perlin noise, as a function of time, offsets from static or dynamic configurations (*i.e.*, character joints) can be calculated, thus augmenting these characters with subtle motion. This process was extended by *Improv*, which featured a method for incorporating smooth noise into animated characters' behaviors (Perlin and Goldberg, 1996). Studies in robotics have shown smooth noise to improve a variety of outcomes in robotics, including likeability and presence (Cuijpers and Knops, 2015; Terzioğlu et al., 2020). Many commercially available collaborative and social robots do not have fully articulated faces with which to communicate social-emotional states, so it is particularly important that there be alternative ways for modeling them.

Specific characteristics of motion, such as “jerkiness” or “velocity,” have been outlined as important for the recognition of certain emotional states in humanoid robots (Beck et al., 2013b,a). When viewed by individuals, faster speed in robots was interpreted as greater excitement or arousal (Sial et al., 2016). Originating in dance theory, Laban Movement Analysis (LMA) (Von Laban and Lange, 1975) and the component of motion shape have since been validated as informative for affect detection in humans and used in animation (De Meijer, 1989; Melzer et al., 2019; Truong et al., 2016; Chi et al., 2000). While not motivated by LMA, the directionality of a simple robot was shown to have a strong emotional impact (Harris and Sharlin, 2011), and has been used to generate profiles of expression movement in mobile robots (Knight and Simmons, 2014).

Smooth, lifelike motion can also function as a signalling mechanism for system states (Ishiguro and Minato, 2005; Belpaeme et al., 2013). For example, if the robot is on but not moving, secondary motion may serve as an indicator to users that the robot is merely idle, while also preventing surprise when the robot moves. Idle behaviors have also been extracted from human ethnographic work (Song et al., 2009), and have been shown to improve aspects of child-robot interactions (Asselborn et al., 2017).

While smooth joint noise can improve the liveliness of agents, it does not capture the full range of expressivity. According to LMA, many features of movement, such as shape directionality, are not relevant in the joint-space of the robot, but rather the *pose* (e.g., Cartesian space) of the robot, making applying these types of features difficult if operating in joint space. Other informative features, such as speed or jerkiness, may be obscured if joint-based noise causes varying speed or jerkiness in Cartesian space.

Similarly, the addition of smooth noise for secondary motion in joint-space can result in problematic configurations or collisions, even if joint limits are respected. For humanoid or bipedal robots, simply adding offsets to individual joints on the lower limbs quickly results in unstable posture, and even falls.

Solutions to Lifelike Motion in Robotics

One solution to these challenges is to simply pre-record or define keyframes for specific motions and interpolate between them as needed. This approach has been employed in prior research (Terzioğlu et al., 2020) and in proprietary software (e.g., Softbank Robotics' NaoQi Autonomous Life (Softbank Robotics, 2022)). As an alternative to manually generating activities, *Geppetto* utilized a user interface to enumerate and visualize possibilities for expressive gestures with the goal of allowing more productive exploration of the potential behavior set (Desai et al., 2019). For bipedal robots, motion on limbs presents an additional challenge due to instability caused by uncoordinated joint movements. As a result, motion is typically either disabled from the waist down, entirely pre-defined, or the issue is avoided by adopting a sitting position and focusing activity on the upper body (Beck et al., 2013b,a). While sufficient for short interactions, pre-scripting these behaviors can have a number of issues. First, without enough keyframes, the behavior can quickly become repetitive, which breaks the illusion of autonomy (Duffy, 2008). Second, when combining activities, conflicts between joints and kinematics might arise. This makes interleaving existing motion with novel, real-time instructions difficult. For example, an early approach attempted to resolve

these conflicts between activities and motions through a hierarchical model (Snibbe et al., 1999). While effective at interleaving the behaviors with motion, the system was not fast enough to run in real-time. These cases illustrate the limitations of previous efforts to balance lifelike motion with task-goals.

Inverse Kinematics

In contrast to specifying the gripper pose indirectly through the setting of joint angles, Inverse Kinematics (IK) solvers attempt to directly specify the gripper pose, and solve for the joint configuration that satisfies that pose. IK solvers, while more easily interpretable in Cartesian space than joint-space methods, can encounter issues such as kinematic singularities. These joint-space issues occur when the robot loses the ability to instantaneously move its gripper in some translational or rotational dimension, because (1) not all poses in the robot’s area can be reached through a combination of joint states, and (2) a movement in Cartesian space may not be possible as a smooth interpolation of joint-space values.

A method that utilizes an IK solver is ERIK, which uses a pass-based approach to integrate joint movements with end-effector goals (Ribeiro and Paiva, 2017). *RelaxedIK* is another IK solver with a different approach. Using an optimization-based method, *RelaxedIK* places importance on both accuracy of the motion (*e.g.*, matching the pose of the gripper), as well as the feasibility of motion (*e.g.*, avoiding self-collisions or kinematic singularities) (Rakita et al., 2017). It is generalizable such that additional objectives can be added, *e.g.*, handling dual-robot systems where one arm controls a camera, optimizing the location and orientation of the camera such that a remote user has a clear view of the task being performed by the other robot arm (Rakita et al., 2018).

5.2 System Design & Implementation

Lively inherits its philosophy from *RelaxedIK* (Rakita et al., 2017) by framing the goal of the joint-space calculation as an objective, but generalizing its implementation

across a greater set of objective types and attributes of the robot's state. Furthermore, while *RelaxedIK* assumed a position and rotation goal on the gripper of each robot arm, and a set of joint smoothness objectives, *Lively* makes fewer assumptions with its *à la carte* approach, giving the programmer greater ability to compose these goals in creative ways for behavior generation.

To explore the capabilities of the system, we will consider three main levels of possible interaction with the system: the *Design Level*, the *Development Level*, and the *Extension Level*.

Design Level

The outermost interaction level is the *Designer Level*, and is the least technical way to explore and utilize the system. We designed *LivelyStudio* as a method inspired by conversations with a set of experts across the fields of animation and robotics. It is meant to support and illustrate many of the capabilities of the *Lively* framework, while maintaining its accessibility. This is done by using a state-based approach, wherein users can compose combinations of social, task-based, or functional behaviors, called *Behavior Properties*, and specify how transitions may occur between these states.

Design Iteration

Our current version of *LivelyStudio* builds upon previous iterations through a small formative evaluation with four professional roboticists and animators involving a mixture of system overview, think aloud, and semi-structured interview, lasting 60 to 90 minutes. The initial design, shown in Figure 5.2, featured a simulator and configuration window, where users could independently curate a set of *Behavior Properties*, a set of states (called modes), and goals (task-based instructions). While states were supported through modes, there was no clear relationship between them, and animators in particular had difficulty translating their keyframe-focused experience to this design: "It's hard to see how poses would be created so separate from the animation (P3)." More generally, how specific goals could be combined

with the *Behavior Properties* was unclear. Additionally, certain interface elements, such as the standard 3D viewer did not have the affordances desired by animators, or had minor usability issues. This feedback was used to create a more effective and intuitive version of *LivelyStudio* for users of varied backgrounds and levels of experience through a more explicitly state-based configuration process, and use of a new custom 3D viewer and updated components.

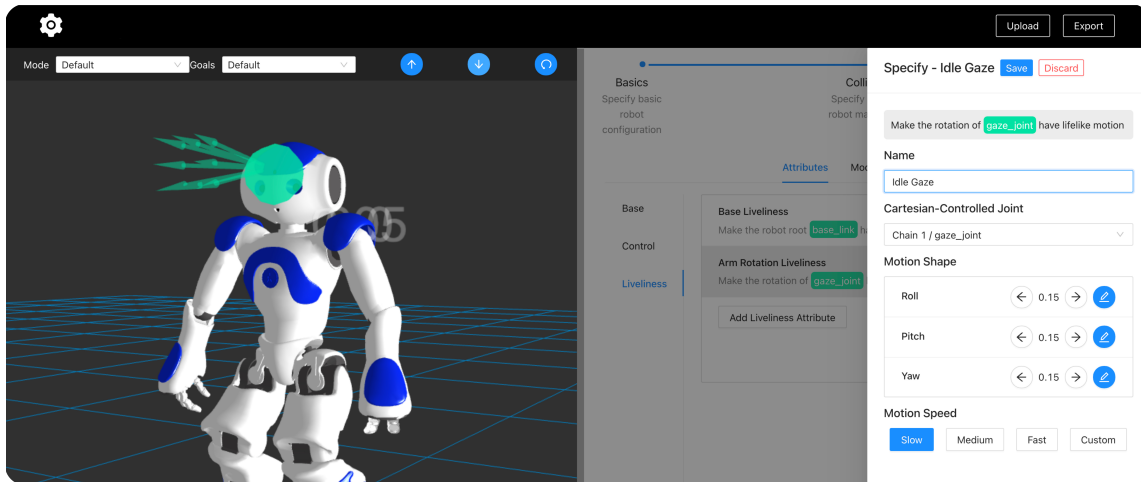


Figure 5.2: An early version of *LivelyStudio* that received feedback from animators and roboticists, which led to a redesigned 3D environment, more explicit state-based design process (states as graph nodes), and bundling of behavior attributes with specific goals and weights.

LivelyStudio Interface

The results of our formative evaluation suggested that a state-based visual programming environment that allows users to develop series of states similar to keyframing would be the most intuitive approach to the design. The state-based approach shares similarities with many other programming environments (Porfirio et al., 2018; Pot et al., 2009; Datta et al., 2012; Glas et al., 2016), which may be familiar to roboticists, but also enables an intuitive design approach for users who are less familiar with typical programming environments like animators, digital artists, or

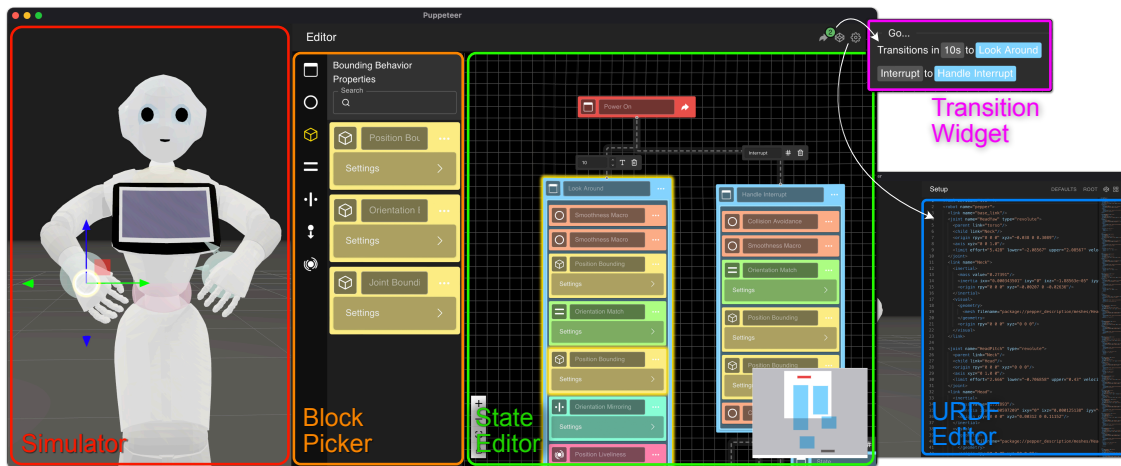


Figure 5.3: The layout of the *LivelyStudio* interface. From left to right, a *Simulator* window shows the robot in the currently selected state; the *Block Picker* allows dragging structural blocks like *States* or *Behavior Properties* like *Position Bounding*; the *State Editor* canvas that allows for states to be dragged around and modified. At the top-right, a menu that reveals a *Transition Widget*, which lists transitions from the current state, and a settings button that reveals a full URDF editor.

other types of designers. *LivelyStudio*'s programming environment contains three primary parts: (1) a selection of state and behavior property nodes, (2) a state-based programming window, and (3) a robot scene. By defining states, and adding *Behavior Properties*, designers can define how a robot will move, or the position it should take in each (Figure 5.3). Improving on the early version of *LivelyStudio*, specific goals and *Behavior Properties* are merged for clarity, and weights are inferred from their relative ordering within states and through usage of priority groups. Designers can specify arbitrary Universal Robot Description Files (URDFs), but visualization of meshes is limited to a discrete set that could be expanded in the future.

Behavior Properties

LivelyStudio allows for a wide range of robot *Behavior Properties* with which users program robot motion. These 24 properties, which serve as building blocks for

defining the behavior and motion of the robot, fit into six categories:

- *Basic behavior properties* revolve around the fluidity of robot motion by limiting rapid changes and considering possible collisions between the links of the robot.
- *Bounding behavior properties* limit the space within which joints can assume angles and links can move or be oriented.
- *Matching behavior properties* specify exact positions and orientations of links or angles of joints.
- *Mirroring behavior properties* allow users to mirror the current state of a link's position or orientation in a different link, or the current angle of one joint in another.
- *Liveliness behavior properties* allow the addition of smooth, coordinated motion to joint angles or link poses.
- *Force behavior properties* simulate the effects of physical forces acting upon the robot.

The function of each *Behavior Property* is visualized in Figure 5.4.

States and Transitions

The state-based programming window starts with a power-on (*i.e.*, initial) state, and a power-off (*i.e.*, final) state. Users can add additional state nodes to their program and populate them with *Behavior Properties*. For example, one state may contain a property that sets the gripper of a robot arm in a pick-up area, while another state sets the gripper position to be near a drop-off area. Once a series of states is created, the user can define how the power-on, power-off, and custom states are connected by dragging transitions from one state to another. These connections can also be given timers, which act as triggers to automatically begin a transition from one node to the next. In this way, a state can function both conventionally,

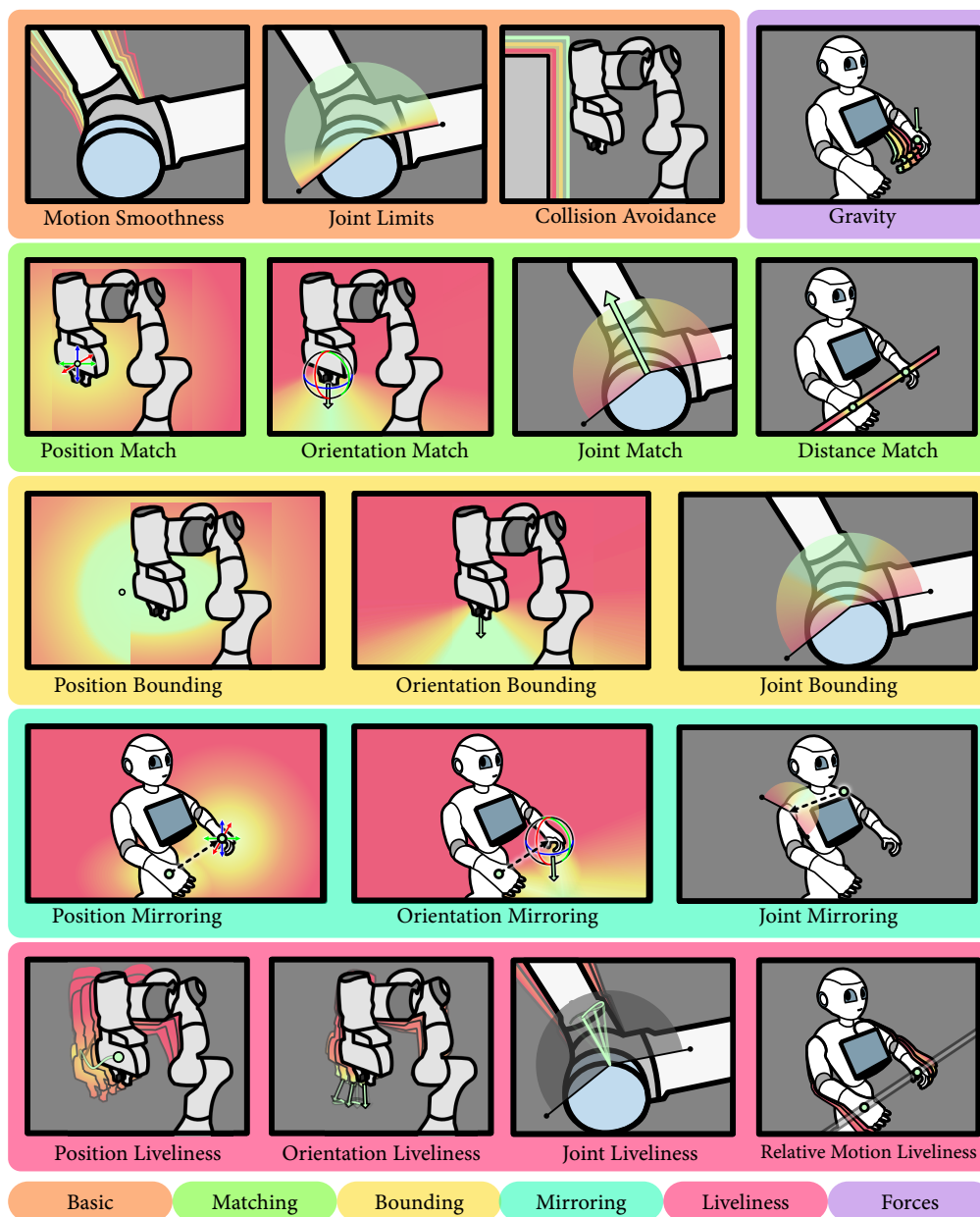


Figure 5.4: *LivelyStudio's* set of *Behavior Properties* that match *Objective Functions* within *Lively*. Note, *Velocity Minimization*, *Acceleration*, and *Jerk Minimization* come in both joint-based and robot root variants, and while usable separately, are included within the *Smoothness* macro property.

defining a set of characteristics the robot will exhibit for an unspecified amount of time, but also as a single keyframe in a timed series. States can have any number of both timed and nominal transitions (simulating event triggers, *e.g.*, a person approaches), and the program will transition states given the first simulated event triggered or timer that expires, whichever occurs first. Of note, while this does simulate how the robot could respond to events, *LivelyStudio* does not currently interface with physical robots, or listen to external events.

Develop Level

For robot programmers desiring greater control over the robot than that afforded by the previously described *LivelyStudio* interface, or looking to control a robot in a more conventional ROS-based approach by creating a control node that publishes joint values, the *Development Level* allows for direct control using *Lively*.

Design & Usage

Lively is written in Rust (Matsakis and Klock, 2014), and accessible as a crate, with bindings in both JavaScript through WebAssembly (Haas et al., 2017) and Python (Sanner and others, 1999). To use *Lively*, a Solver is imported and constructed with any valid URDF, persistent scene objects, objectives, and other solver settings. Execution of the solve method, which accepts the current goals, weights, time, and real-time collision data, returns a robot state that best satisfies those goals given the current weightings and previous robot state. This approach allows for *Lively* to be used in a variety of contexts, including ROS (Quigley et al., 2009), web or simulation, and directly on hardware. Solve times with randomly arranged colliders are shown in Figure 5.5.

Objectives, Goals, & Weights

To achieve a high degree of customization and dynamic control, we introduce the concepts of objectives, goals, and weights. Whereas *LivelyStudio* abstracted away these features as *Behavior Properties* for the purpose of accessibility, the core

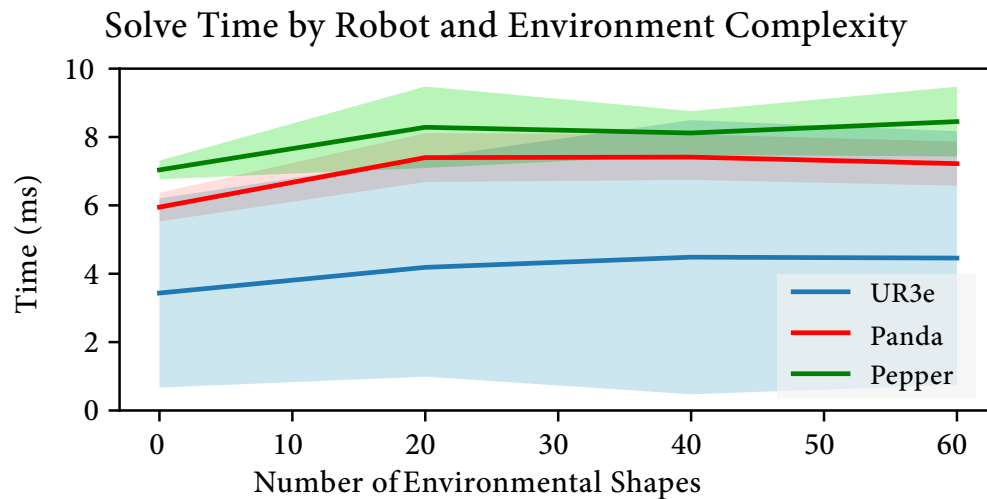


Figure 5.5: Solve times for the UR3e, Panda, and Pepper robots, with randomized locations of environmental colliders. Of note, speed is largely unaffected by shape count.

framework allows for more direct control. The identities of the individual objectives match with the set of *Behavior Properties* enumerated in Figure 5.4, and goals are summarized in 5.1. Importantly, while *Behavior Properties* encoded the discrete goals (e.g., the position for *Position Match*, or the scalar for *Joint Match*) associated with each *Behavior Property*, and the weights are inferred by the ordered ranking within states, these are separated at the framework level. Thus, the previously mentioned position goal can be determined in real time through external means, such as sensing, and passed as an update within each iteration of the solver. Similarly, the developer in real time can adjust other goals, such as a position bounding ellipsoid (*Position Bounding*), joint values (*Joint Match*), or size (for *Position Liveliness*), and weights, allowing for prioritization of certain goals or the deactivation of others, based on the current development needs. Because objectives are organized by key, and atomic updates are possible for goals and weights, only the needed changes must to be included each round.

Objective Configuration

The complete set of objectives feature a wide range of configurable attributes, beyond simply their goals and weights. The simplest objectives focus on safe and smooth motion, corresponding to the set of *Basic Behavior Properties*, and do not accept additional parameters. Those corresponding to *Matching*, *Bounding*, and *Gravity Behavior Properties* are configured with the joint or link with which they are paired. *Mirroring Behavior Properties*, defining relationships between pairs of links and joints, accept a pair of each. Finally, *Liveliness Behavior Properties* feature an additional field, frequency. This value functions as a temporal scaling value that increases or decreases the rate of change in the Perlin noise generator functions for that objective. Combined with the goal values passed into liveliness objectives, developers can access a wide range of motion profiles. Importantly, because the formulation of the liveliness objectives is not dependent on having a concrete goal attached to the same link or joint, it is possible to add movement to otherwise uncontrolled parts of the robot.

Collision Avoidance

Lively implements the *PROXIMA* collision detection algorithm, which allows for time-efficient collision and proximity detection for robots (Rakita et al., 2022). The

Table 5.1: Goal Types

Entry	Description
Translation	A 3-vector representing coordinates
Rotation	A Quaternion representing rotation
Scalar	A float value
Size	A 3-vector representing scale of a 3D shape
Ellipse	A structure designating a rotated ellipsoid, with <i>Translation</i> , <i>Rotation</i> , and <i>Size</i> components
RotationRange	A structure including a center <i>Rotation</i> , as well as a float value indicating allowed delta in radians from that rotation.
ScalarRange	A structure including a center float value, and float value representing allowed delta from that value.

Collision Avoidance objective serves to utilize the data generated from this collision detection algorithm to prevent collisions. *Lively* employs a three-fold approach to handling modeling collision objects. The first is input from the URDF during the initialization of the solver, which supports default shapes like boxes and cylinders as parts of the collision model when parsed. For cross-platform and web-based reasons, mesh-based colliders are ignored during URDF import. Additional colliders can be specified during solver initialization, including basic shapes and convex hulls, and can be attached to the world or any link in the robot. Finally, as an optional input to the solve method, developers can provide real-time updates to the collision model, adding, deleting, and moving colliders.

Extend Level

For robotics developers seeking to modify the behavior of the existing *Lively* objectives, or wanting to increase functionality by creating completely new objectives, *Lively* has a modular and configurable approach to supporting the *Extension Level*.

State Model

As discussed, *RelaxedIK* utilizes an optimization approach, with the robot state S_R being represented as a vector in the joint space S_J of the robot internally. *Lively* takes a similar approach, but an additional six dimensions representing the transform of the root link are added to create the optimized vector x . However, this vector representation is not always the most natural way to evaluate the state, and to ease the computation each objective performs, this vector is converted into a more comprehensive state representation containing joint states, link transforms, and proximity information, described in Table 5.2. This state, as well as previous states, are provided in each call to objectives.

This formulation of the state allows for straightforward creation of additional objectives. It is also possible that additional features of state may be needed for the creation of certain new objectives. The *Robot Model* handles the generation of new robot states from the vector x . For example, if a force-based objective was desired,

Table 5.2: State Properties

Entry	Description
Frames	A lookup table of each link's position in both world and local coordinates
Joints	A lookup table of each joint's value
Origin	The transform of the root link. This data is also included in frames
Proximity	A vector of data representing pairwise proximity between the robot's links and other robot parts and the environment. Each entry contains distance, as well as the closest points between the pair of colliders
Center of Mass	A 3-vector representing the center of mass of the robot in the world frame

Table 5.3: Objective Description

Entry	Description
update	Function, accepts the current timestep and performs any updates to its internals that are necessary, as in the case of Perlin noise-based objectives
set goal	Function, accepts the goal value supplied by the user. Each objective accepts a specific goal type
set weight	Function, accepts a new weight value, if updated by the user
weight	Float, indicates the scaling value for the objective cost value
call	Function, accepts a State and Variable data object, returning a numerical cost value. The Variable object contains a record of previous states and information about the robot

the robot model would have to be extended to output a state that provides the data the objective would have to operate on.

Objective Formulation

Similar to *robot state*, each objective adheres to a well-defined convention that can be used to extend the capabilities of *Lively*, as shown in Table 5.3. As previously discussed, each objective is paired with a specific goal type (e.g., *Position Bounding*

with *Ellipse*, and *Position Liveliness* with *Size*), and the goals are enumerated in Table 5.1. Additional goal types can also be added to support new objectives and functionality, as long as they have a predictable structure (e.g., a pointcloud goal could be an array of any length with structure $[[x : f64, y : f64, z : f64], \dots]$).

5.3 Case Studies

Design Level

Users of a wide range of experience levels can engage with our system using *LivelyStudio*. Artists, character designers, and animators, who may not be familiar with traditional programming tools, may particularly benefit from *LivelyStudio*'s accessible user interface.

Kiosk Robot

Suppose a user is creating a program for a social robot providing general assistance in a public area. Here, the robot may have states such as idle, greeting, or thinking. The user can begin by creating state nodes within the state editor. One such state could be labeled "Idle" to represent the idle status of the robot within the overall program. From here the user can begin adding *Behavior Properties* to the state. First, the user may apply the *Position Liveliness* property to the torso of Pepper as a visual indication that it is powered-on and functioning. Next, the user may add the *Joint Liveliness* property, and configure it to "Head Yaw" to make the robot's head sway from left to right and signify that it is looking around for people to assist. Finally, the user can select the *Smoothness Macro* property to ensure that the robot's motion remains smooth and natural, and the *Collision Avoidance* property to prevent collisions. The user may also create a "Greeting" state, which directs the robot's gaze toward a nearby person. Once these states are generated, the user can create a connection between them and add a label to identify a triggering condition. The user may want Pepper to transition from the "Idle" state to a "Greeting" state when a person approaches. During this transition, Pepper can reduce head sway from

the "Idle" state, direct gaze toward the user in the "Greeting" state, and maintain the liveliness motion included in both states. This process can be repeated with any number of states and complex transition patterns.

Cobot Keyframing

In another example, a user may want to create a program for a robotic arm such as the Panda robot that functions as a series of states, similar to keyframing. The user can create an initial state, add the *Position Match* property, and configure a specific position for the gripper. The user can complete this process to define all waypoints for the gripper of the Panda robot as separate states. Given space constraints in the deployment environment, the user may also want to design their program to limit the space in which certain links will move. Thus, the user may apply the *Position Bounding* property to specific links so that the robot limits its spatial footprint while moving. Finally, the user may need the robot to interact with an object from a specific grasp point. Therefore, adding the *Orientation Match* property to the gripper enables it to manipulate an object from a reasonable angle. Once all the states are created, the user can create timed connections between states, such that transitions will occur automatically.

Develop Level

While all users may find use in *LivelyStudio*, those with substantive experience programming and planning robot motion will be able to leverage the capabilities of *Lively* directly. We consider two example use cases to explore how *Lively* may be used.

Real-Time Robot Control

Using a UR3-e series robotic arm, a developer seeks to devise a system that, on button-press, scans the area using a camera attached to the last robot link, and finds any of a set of items. Any item it finds is picked up and placed in a nearby box. The developer creates a ROS-based setup with two nodes. One node receives a

camera feed and transform data from the robot, while publishing all valid items and their transforms that it detects. A second, *Lively*-focused control node listens to this set of items, and publishes transforms of the robot to be consumed by the first node. The control node defines a *Lively* solver, configured with the robot description, and an additional camera collider that is attached to the last link. The solver is configured with *Position Match* and *Orientation Match* objectives on the final link, and a *Position Liveliness* objective on the forearm link. Finally, the set of objectives is completed with *Smoothness*, *Joint Limits*, and *Collision Avoidance* objectives. On button press, a preset collection of positions and orientations are sequentially passed to the corresponding objectives in the solve method, along with instructions to turn the liveliness weight to zero. The resulting state is parsed and converted into TF messages, which are passed via a topic to the data parsing node. Upon calculation and communication of scene items to the control node, the node selects the first item to move, passing the position and orientation to the solver, followed by the goal position of the items, then repeating until no items remain. Once complete, the position and orientation goals are moved to a neutral pose, and the weights relaxed, while the liveliness objective weight is increased.

Browser-Based WOZ

A developer wants to create a ROS-based wizard-of-oz GUI interface that allows actions to be selected and executed on a robot in real time, but also want the robot to respond to potential collision objects in the environment and exhibit certain lifelike motions. The robot, Pepper, has two arms, wheels, and a head, and the developer already has an existing library of joint-based trajectories. However, they want to include additional liveliness in orientation space around the head and position liveliness (a swaying motion) on the torso. Objectives for each controlled joint are created, as well as some basic objectives. The developer's GUI initializes a web-based version of the solver. A web-based ROS connection is formed to the robot, starting a subscription to sensor data, and a publisher that sends real-time joint instructions to the robot. Selecting an action updates the goals for each joint,

and the set of all potential colliders that the robot gets from the sensors are updated each invocation of the solve method. Joint instructions from the result are passed to the robot after each solution is found. To accommodate all goals simultaneously, the system will attempt to reach the specified joint values, while adding in liveliness and avoiding collisions.

Extend Level

The current functionality of *Lively* and *LivelyStudio* address most user needs when programming robot motion. However, if additional functionality is desired, a developer could easily extend our system's capabilities by defining new objectives and goals. We outline two examples of extensions that would be feasible within *Lively*.

Center of Mass Objective

Lively can be greatly extended through the development of additional objectives. Because the robot state already includes a vector representing the center-of-mass of the robot, it is straightforward to create a new objective, implementing the methods defined in Table 5.3, that operates on it, which could be useful in cases where the robot's balance must be maintained, or as a way to center the robot near its base. The specified objective would accept a *Translation* goal, and use the default implementation of *update*. The *call* method would be implemented by calculating the distance between the goal value and the center-of-mass vector in the robot state, returning a cost that grows with distance. Finally, the objective is added to the set of Objectives. The resulting objective would attempt to produce poses that are centered as much as possible on the goal vector provided.

Perspective Noise

While the *Position Match* and *Orientation Match* objectives together are capable of creating a lifelike appearance, a developer may desire to create a lifelike behavior that exhibits positional and rotational motion around an offset focal point, as if

inspecting the properties of an object located there. Doing so requires the addition of a new goal type, which would encode the focal length to maintain the position of the focus, and the amount of rotational/translational movement allowed. The objective's call method would use these goals and a Perlin noise generator function to project the needed position and orientation in space to achieve the specified rotation around the focus at a given time and compute the radial and translational distance from those values, returning a cost value. The resulting objective would attempt to produce poses that adhered to this dynamic pattern as a function of time.

5.4 Chapter Summary

In this chapter, we presented *Lively*, a system for generating smooth, lifelike, and customizable secondary motion in a variety of robotic applications by formulating them as goals in an optimization framework. Because of this optimization-based approach, and a Cartesian space representation, we can produce robotic motions that have the potential to be more easily readable by viewers, without sacrificing task-based goals or guards against collisions or singularities. We also presented *LivelyStudio*, a state-based visual programming and configuration environment that allows for exploration and design. Developers can utilize *Lively* directly in multiple programming and execution environments, in applications ranging from traditional keyframe-based to real-time control.

In many ways, the goals of *Lively* and *LivelyStudio* are similar to those of *CoFrame*, being interested in allowing more novice individuals to still contribute to the design space of mixed social-collaborative robots. However, they differ in their approach and primary focus. Whereas *CoFrame* was meant as a complete programming-learning environment, and mostly focused on the overall safety, quality, and efficiency of the cobot program, *Lively* is primarily a library that can be integrated into other systems, focused exclusively on motion specification. Despite the narrower focus, it is nevertheless integral to the quality of the overall cobot program. Both high-level and low-level attributes (*e.g.*, *CoFrame* and *Lively*, respectively) are

needed for a complete system, and indeed, new versions of *CoFrame* utilize *Lively* for its motion specification.

LivelyStudio bridges this gap in a way. Being a state-based visual programming environment, it lowers the barrier for entry, prioritizing immediate feedback about the configuration of certain robot motion behaviors. The formative evaluations of the *LivelyStudio* interface demonstrated a disconnect between the keyframe-based approach of animators and the real-time or procedural needs of more dynamic social-collaborative robots. The solution explored in subsequent versions of *LivelyStudio* was to support a blending of these two approaches in the state-based design, thereby allowing the designer to switch between keyframe-based specification and states incorporating real-time procedural behaviors. This is accomplished by conceptualizing each state as a collection of composable behaviors, both static and procedural, made possible by the flexibility of the *Lively* framework.

That being said, future work could improve the state-based representation used by *LivelyStudio* by exploring the right representation for how arcs are triggered. *LivelyStudio* only handled timed and keyword-based transitions, which while useful for *LivelyStudio*'s goals of supporting exploration and design with the *Lively* system, it is not sufficiently robust for full cobot programs. State machines which allow for a greater specification of these constraints could be explored, including options like Petri Nets (Peterson, 1977), which are discussed more extensively in the *Allocobot* chapter.

As related note, the design of *LivelyStudio*, as well as the improvements made to *CoFrame* in the time since the original publication, have motivated the design of a tool for designing visual representations of the programs themselves. This tool, called *OpenVP*, is discussed in the next chapter.

6 OPENVP

Visual programming is a common approach for the specification of programs in robotics. These programs can come in many forms, including more traditional imperative programs, state machines, or even behavior trees. Existing well-established and ubiquitous tools like Blockly (Fraser, 2015), and Scratch (Resnick et al., 2009) do suffer in their limited interoperability within the context of larger applications and relatively poor flexibility in usage.

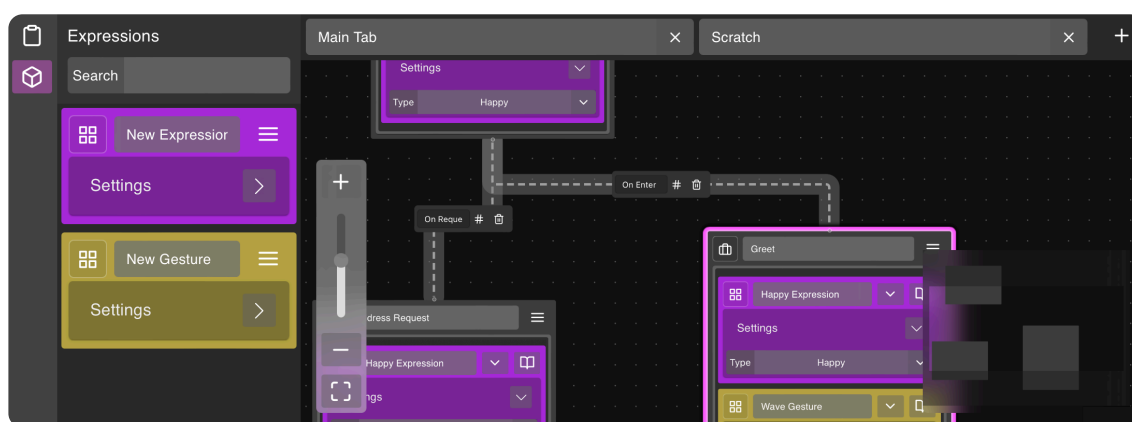


Figure 6.1: An example flow-based programming system designed with *OpenVP*, illustrating a simple logic about how a robot should behave if a patron enters a store.

Even in the context of greater emphasis on voice-based or chat-based design of programs, there will still be a place for visual programming systems. For example, after designing an entirely voice-based prototype system, Porfirio and colleagues found that while spoken language had benefits, it was generally inefficient or poorly suited for complex specifications, leading the team to ultimately construct a multi-modal system instead (Porfirio et al., 2023).

In the process of designing various tools such as *CoFrame* and *LivelyStudio* in the robotics space, we have iterated and improved on a component library for easily specifying highly customized and tightly integrated web-based visual programming

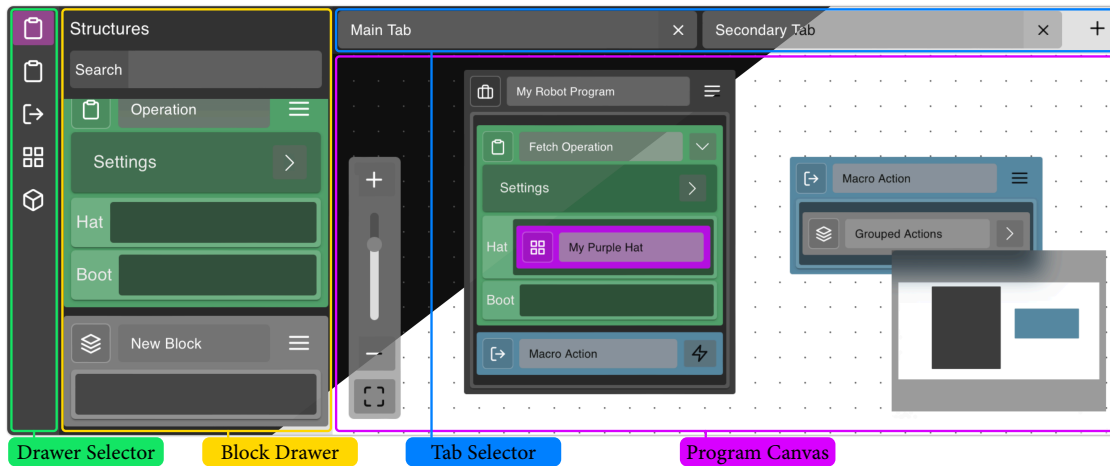


Figure 6.2: Overview of *OpenVP*'s *Environment* layout, highlighting the four main sections: the *Drawer Selector*, where the active drawer can be set, the *Block Drawer*, where blocks in the current set can be selected from, the *Tab Selector*, where individual tabs can be added, removed, hidden, and edited, and finally the *Program Canvas*, where the program is visualized and edited. Full customization of the theme is possible, as shown in the light/dark modes.

environments within larger applications. This system, called *OpenVP*, is a React component library that can easily be integrated into robot programming systems. It provides a common block-based programming environment suitable for a range of program designs, such as imperative programming and flow-based or state machine programs.

OpenVP has been designed with multiple goals in mind¹. These include:

- A high degree of straightforward configurability;
- Serializable and portable program representations;
- Tight integration with the rest of the interface;

¹The research discussed in this chapter is derived from published work by myself and Dr. Bilge Mutlu. All authors contributed significantly to the conceptualization, design, implementation, evaluation, analysis, and/or the writing of the original manuscript. In addition, due to *OpenVP*'s relationship to the development of *CoFrame*, Nathan White assisted with specific logic for block deletion handling.

- Abstraction of basic interaction details;

OpenVP addresses these goals by creating a system by which a clearly defined *Program Specification* can be used to define the functionality of the intended visual programming experience, and interoperability with the rest of the greater interface is key to its architecture. In the following sections we will articulate some of the characteristics of this system, and how it can be customized for a variety of applications.

Target Users

The design of *OpenVP* was driven in many ways by the needs of the *CoFrame* and *LivelyStudio* systems. These two systems, while differing considerably in their intended target users, shared a common need for a visual programming environment that could be tightly integrated into the rest of the interface. The *OpenVP* library is in many ways a meta-level library, built for roboticists, researchers, and developers like ourselves to support us in creating the types of systems we need in order to support other users.

Research Questions

In creating *OpenVP*, we consider a number of questions about the design of visual systems for collaborative robotics. First, what are the unique affordances that visual programming systems should provide when focusing on robotics applications? Second, and largely hypothetical, is the question of how higher-level abstractions around the design of such robotics-focused visual programming systems might bootstrap the design and exploration of future systems and paradigms. The answer to that question will only be known in time, but we hope that *OpenVP* can serve as a catalyst.

6.1 System Design & Implementation

OpenVP has been designed both for usability by end-users, but also to balance the usability by developers with its capability. This results in a number of high-level characteristics that guide its design.

Overview

The focus of the *OpenVP* library is the *Environment* component, which serves as a single entry-point to using the system. The *Environment* itself contains a number of other built-in elements, shown in Figure 6.2. Central to the environment is the *Programming Canvas*, where the entire program can be visualized on an infinite canvas. Users can pan and zoom this canvas to see a high-level overview of their program, or focus on a particular block. The canvas also features a mini-map and canvas navigation widget. To support editing, a number of other elements are present. The first on the left is a *Drawer Selector*. Activating drawers exposes the corresponding *Block Drawer*, which features a filterable list of blocks that can be dragged onto the canvas. Finally, a navigational *Tab Selector* at the top allows for the creation, editing, and removal of tabs.

Block Types

A key feature of *OpenVP* is the customizability of the system for a variety of possible programming paradigms or applications. As such, it is important to support configurability of the various block types that can be used in each instance. As such, *OpenVP* uses a two-part approach to configuring blocks, separating out the program data model, which aims to be serializable for server-based applications, from the program representation data, for which configurability benefits from the introduction of full javascript functionality. We call these two components the *ProgramData* and *ProgramSpec*, respectively. Within the *ProgramSpec* is contained information about the drawers provided in the interface, as well as all types available to the users.

Each type specification inherits from one of two primitive types, either *object* or *functions*, and their specifications include information about the properties of each, as well as customizable rendering information for their *instance* and *reference* blocks (for *objects*), and *declarations* and *calls* (for *functions*). For example, it is possible to create three different *object* types (e.g. a *ProgramType* for the top-level entry-point, an *OperationType* to represent some behavior, and a *TargetType* to represent something for the *OperationType* to act upon). If allowed, each of these types could include separate visuals for how *instances* and *references* are rendered. Similarly, it is possible to generate multiple types of the primitive *functions*, if needed. Note, the configurability of functions is still driven to a large part by the end-user, since a key feature of entries inheriting from the *function* primitive is that arguments can be added and removed to the declaration itself from the interface.

Drag and Drop

Drag and Drop is central to the design of *OpenVP*, borrowing from similar tools such as Blockly (Fraser, 2015) and Scratch (Resnick et al., 2009). In *OpenVP*, users can select and drag blocks from the drawer into the canvas. Depending on the needs of the application, some block variants can be designated as *canvas* blocks, meaning that they can be dragged directly onto the canvas and organized on the 2D grid. Other blocks can be limited to *non-canvas* blocks, meaning that they are only applicable as children to *canvas* blocks, or other *non-canvas* blocks. When dragging a block, valid drop points are highlighted in the interface, giving a visual reminder of where they can be deposited. Upon hovering the block onto a valid drop zone, a preview of that block in the specified location is shown. Hovering over drop zones without a held block provides a tool-tip that visually shows which blocks are valid at that location.

As mentioned before, types inheriting from the *function* primitive allow editing of their arguments. Function *arguments* can be seen in the header of the corresponding function declaration. As would be expected, arguments within a function's context can be dragged anywhere within that function, but are not available to be

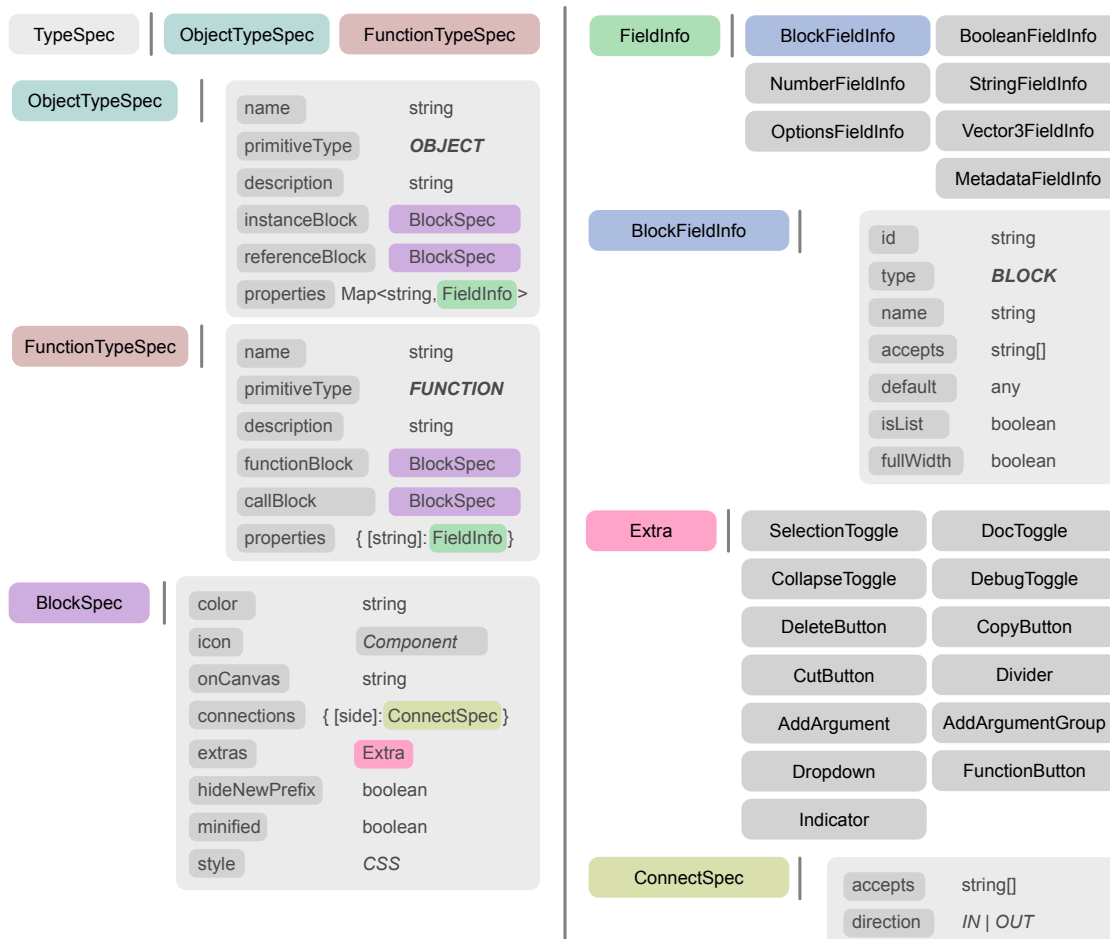


Figure 6.3: Overview of block customization via their associated *TypeSpec* data. For brevity, some variants are not included, notably non-block *FieldInfo* structs, (e.g. *NumberFieldInfo*, *StringFieldInfo*, etc.). Also not shown is the *Extra* and *ConnectSpec* fields, discussed elsewhere.

dropped outside that context. Conversely, block *references* from outside that context can still be dragged in and used within a function.

Block Design

Blocks are highly customizable, with their appearance and behavior specified within the *ProgramSpec*. An overview of configuration of a block's *TypeSpec* can be found in table 6.3. Breaking down this specification, each variant (*ObjectTypeSpec* and *FunctionTypeSpec*) includes a set of two *BlockSpec* entries. Each of these entries can independently specify the color of the block, the icon, whether it appears on the canvas, any connections it can make with other blocks, menu items, whether newly spawned items have the "New" prefix attached to the name (e.g. "New Operation" or "New Robot Function"), whether it features a compact design, or any other CSS style overrides that are desired.

Parameters

Considering the two *TypeSpec* variants, each block can specify the parameters of that block, included through the inclusion of *FieldInfo* data. These structs come in a variety of forms, including *BlockFieldInfo*, *NumberFieldInfo*, *StringFieldInfo*, *OptionFieldInfo*, *BooleanFieldInfo*, *Vector3FieldInfo*, and *MetadataFieldInfo*. The contents of the *BlockFieldInfo* data structure is shown in Figure 6.3, which dictates how other blocks can be dropped into that block, either as a list of blocks or singular parameters.

Menus and Documentation

Menus for each block can be configured separately for each *BlockSpec* entry, and include a set of basic, prescribed functionality like *selection*, *deletion*, *documentation*, *copying*, and *cutting*, as well as more customized cases like *adding arguments to functions* and *custom javascript functionality*.

The *TypeSpec* can also provide a *description*, which is a markdown-flavored text string, which can be used in the in-editor documentation. This markdown supports major features, as well as a customized link usage such that links to valid types will create a hyperlink to that type's documentation.

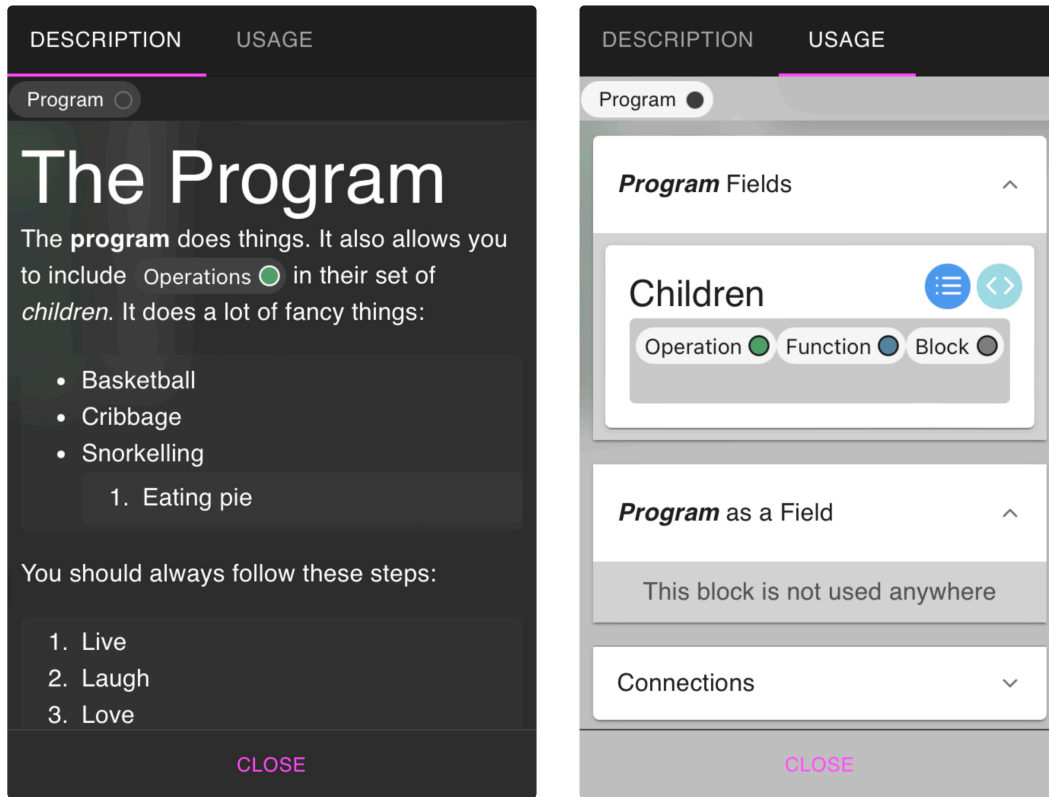


Figure 6.4: An example of a Documentation section generated for an example *Function* block. The documentation automatically curates how that block is used in other blocks, and what blocks it uses. Additionally, the *Description* tab will render the textual markdown description from the *TypeSpec*.

Connections

For each of a *TypeSpec*'s *BlockSpec* structures, it is possible to configure how that block can connect to other canvas-based blocks. This data structure is relatively straightforward, including a set of block types that are allowed to connect, and whether that connection is incoming or outgoing. With this capability, it is possible to design flow-based programs, in addition to imperative ones. An example of such a design can be seen in Figure 6.5.

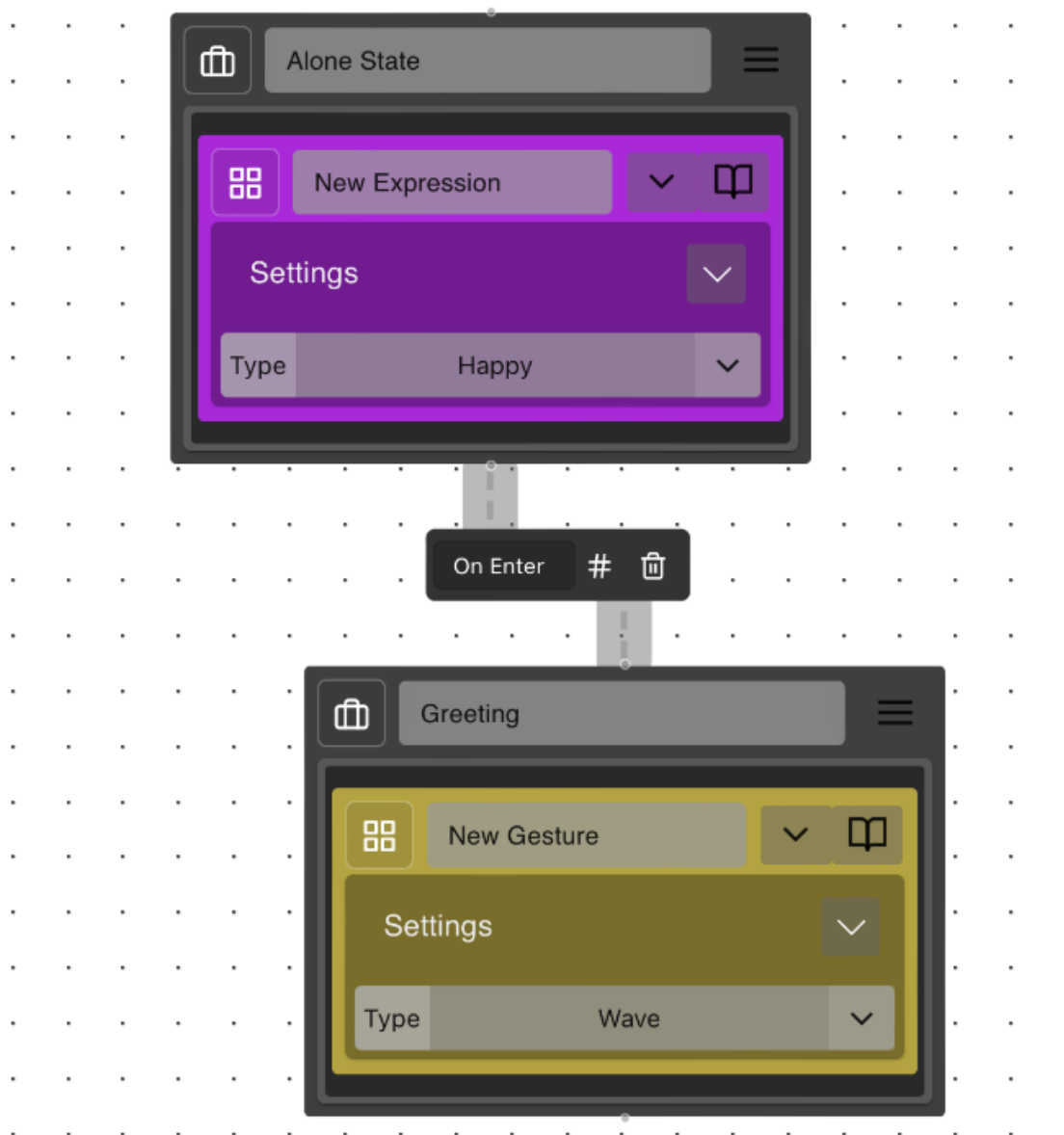


Figure 6.5: A small example flow-based program, illustrating the ability to draw connections between canvas-based nodes. Connectivity is configured within *Block-Spec* structs.

Integration

OpenVP was built with the understanding that it needs to operate within the context of a larger design application. This type of capability is essential if, for example, it is desired that when an error is found and selected, the corresponding block highlights. Alternatively, perhaps it is desired that while the actual process is running, the current progress of a given block could be updated. *OpenVP* solves this by making the internal data model, including both the *ProgramSpec* and *ProgramData*, accessible or editable from outside the component.

Data Store

The above functionality is achieved through the use of a flexible data store model called (Kato and Henschel, 2019). All the behavior for the store, including the internal actions, are contained within this store, which is provided in the library. If you want access to the internals, substitute your own version of this store as a property of the *Environment* component. Full information on how to configure this in applications will be provided in documentation.

External Blocks

Suppose a designer wishes to provide a visualization of a certain block from the *Environment*, but outside the *Environment* itself. For this purpose, we provide an *ExternalBlock* component, which when connected to the correct data store, renders a full block, minus the *Environment*.

Execution Progress

Robotics generally involves a number of long-running processes, and it can sometimes be useful to receive feedback about these processes within the interface itself. A part of the store is reserved for configuring the progress of any blocks in the *Environment*. This is done as a simple lookup of block ids to either numbers, or clock-sensitive javascript functions.

6.2 Source Code and Usage

The source code for *OpenVP* is provided freely on Github², and can be added to existing projects via the node package manager (NPM). Documentation is hosted on github³.

OpenVP uses a MIT license. As a high-level library itself, it makes use of many other libraries, such as ReactFlow (Webkid GmbH, 2019) and Zustand (Kato and Henschel, 2019). The former provides a fee-based Pro tier for usage by commercial entities (it is free for research and academic purposes), so all commercial usage of *OpenVP* should abide by those rules as well.

In conclusion, *OpenVP* seeks to provide an extensible, configurable, and forward-facing tool for visually specifying programs common in robotics applications. Early versions of this library have already been used in widely different robotics programming applications, such as *CoFrame* and *LivelyStudio*. It is our hope that by making this software available more widely, others can benefit from and contribute to its further development.

6.3 Chapter Summary

OpenVP grew out of a need to create a flexible, configurable, and extensible visual programming library for use in a variety of robotics applications. The two systems it was specifically utilized in, and which guided its design, were *CoFrame* and *LivelyStudio*. A commonality between these systems is obviously the VPE aspect, but also the integration of that VPE in the larger interface to support feedback, concept linkage, and other contextual mapping. For example, with *CoFrame*, we wanted to be able to highlight specific areas of the program that produced certain errors, and in *LivelyStudio* we wanted connect the program and visual representation of *Behavior Properties*. In both, this was bi-directional communication. Selecting nodes or blocks could cause elements in the system to focus, or vice versa.

²<https://github.com/Wisc-HCI/open-vp>

³<https://Wisc-HCI.github.io/open-vp>

CoFrame and *LivelyStudio* differ significantly in their program representation, however. *CoFrame* features an imperative style of programming, where the program is a list of instructions that are executed in order. *LivelyStudio* on the other hand, features a flow-based programming style, where the program is a graph of nodes that are executed in parallel. *OpenVP* was designed to support both of these styles under a single umbrella. It should be noted that *OpenVP* is not itself a visual programming language itself, but rather a single architecture for representing the appearance and behavior of custom domain-specific languages, which themselves serve as high-level specifications for the underlying robot program and lower-level systems. For example, an interface like *LivelyStudio* would transfer its program to server or backend, which converts the nodes and *Behavior Properties* into a set of *Lively* objectives. These objectives would be used in the *Lively Solver* to determine the joint angles at a given time, which would in turn be sent to a control node on the robot, translating the joint angles into motor commands.

While *OpenVP* arguably transcends the specifics of collaborative robotics into general visual programming, the design decisions were motivated by these types of usage patterns. Features like durative execution feedback and program serialization were all motivated by the needs of these systems.

We hope that *OpenVP* can grow to become a robust ecosystem for the design of robot-specific visual programming environments. As part of this, we would like to further explore how an even greater set of state-based program representations might be supported, such as Petri Nets, much like that of the final chapter regarding *Allocobot*.

7 ALLOCOBOT

Cobots are made to provide assistance to human workers in a shared workspace as full coworkers (Fanuc, 2023; Kuka, 2023; Universal Robots, 2023). Frequently, these robots are distinguished by features such as sensing, compliance, and safety mechanisms. These features are intended to allow cobots to work alongside humans, rather than in isolation, and to provide assistance to humans in a way that is safe and comfortable. According to Grand View Research (2023), they are also growing in popularity, with the cobot market expected to expand at a compound annual rate of 29.9% in the U.S. and 32.0% between 2023 and 2030.

However, despite the growing popularity of cobots, there are still many challenges to be addressed before they can be widely adopted effectively. One such challenge is the allocation of cobots to tasks, and the resulting reorganization of work entailed. In this context, allocation refers to the process of assigning cobots to tasks in a way that is safe, efficient, and effective. This process is challenging because it requires consideration of many factors, including the capabilities of the cobots, the capabilities of the humans, the physical layout of the workspace, and the nature of the tasks themselves.

Even using a relatively high-level approach, it has been shown by Liu et al. (2022) that the impact of a cobot on the ergonomics of a task for the human is highly dependent on the nature and characteristics of the job itself. Using data from the O*NET database (National Center for O*NET Development, 2023), along with cobot expert-derived evaluation cobot capability, they predicted the impact that inserting a cobot into process would have on the human from an ergonomics perspective. This analysis showed that while some jobs did indeed predict improvements, other jobs showed either no improvement, or even detriment to the human workers. This work underscores the difficulty in introducing collaborative robots into workflows, since this variability in outcomes is likely more varied in individual cases, and more dependent on individual implementations. That fact, as well as practical challenges in the development of true interactive systems, helps to explain why cobots are

mostly used in isolation, rather than in true collaboration with humans (Michaelis et al., 2020).

During this growth of cobot interest we must develop tools and methods that can assist in the effective allocation of cobots to tasks. These tools and methods should be able to assist in the allocation process by providing a way to model the work that humans do, and to analyze the impact of cobots on that work. They should use a range of perspectives, including those of ergonomics, engineers, human factors, economists, and human-robot interaction. Importantly, this must simultaneously be done in a manner that is still accessible to a wide variety of users, including those with little to no experience in some of these domains, because the various stakeholders may not have all the expertise needed at hand Michaelis et al. (2020). This modeling and analysis should also be able to be performed in a way that is flexible and extensible, so that it can be applied to a wide variety of tasks and situations. Possible allocations should be easily inspected, but also flexible, so that they can adjust and adapt depending on minor deviations that workers may make from the "optimal" allocation. Finally, these tools should provide a clear set of guidelines for how a cobot program could be generated.

The focus of this ongoing research is the design of one such system, combining perspectives like ergonomists, economists, human factors engineers, and cobot experts to provide a pipeline that supports the analysis, restructuring, and analysis of human work as cobots are introduced. This pipeline, called *Allocobot*, will take as input a description of human-only work, translate this work into a representation capable of being reconfigured as a human-robot interaction, and then determine the set of choices (*e.g.*, the robot model selection, robot allocations, and arrangement of work) that would be most effective for the task. The result of this pipeline will be a policy indicating these choices, being flexible enough to handle certain deviations by the human workers, while also concrete enough to provide clear guidelines for implementation as a robot program.

Within this chapter, I will outline the current state of the art with regards to task allocation and specification, the status of the *Allocobot* project, and discuss the

future work that is planned¹.

Target Users

Allocobot, like the *Authr* and *CoFrame* systems, was meant to target users like automation experts and ergonomics specialists. This is evident in their aligned goals and perspectives, even if their methods and approaches differ. What we hope is that with *Allocobot*, we can provide a more holistic, principled approach than that of *Authr*, while producing a more usable output for practical development and decision-making. Of note, the set of stakeholders for *Allocobot* is also greater, given this interest in the decision-making process, which is useful for individuals in the business and management side of the organization. While such individuals may not be the users of a system like *Allocobot*, they are nevertheless involved in the process of considering whether and how to introduce collaborative robots, and therefore a key consideration in the design of the system.

Research Questions

Allocobot is ongoing work that asks important questions about how to facilitate productive exploration by automation experts and ergonomics specialists of the impact of cobots on human work. Specifically, we ask whether a novel primitive set may provide the benefits of existing ones like *Therbligs*, while also being more amenable to joint human-robot work. We also consider whether a Petri Net-based representation can provide a more flexible and extensible representation of human-robot collaborative work, and whether algorithms built around them can both generate reasonable, useful feedback, and also do so in a timeframe that is amenable to the types of workflows desired by these users and stakeholders. Finally, we will

¹The research discussed in this chapter represents unpublished work currently helmed by myself, Nathan White, Anna Konstant, Dr. Robert Radwin, and Dr. Bilge Mutlu. All authors contributed significantly to the conceptualization, design, implementation, evaluation, analysis, and/or the writing of the original manuscript, but this set of authors is subject to change. Additionally, we thank Dr. Aws Albarghouthi, Dr. Josiah Hannah, and Dr. Dieter van Malkebeek for early discussions about satisfiability, reinforcement learning, and complexity.

consider whether the algorithms, when provided with real descriptions of work, can correctly identify places for improvement through the introduction of collaborative robots and the reorganization of workflows.

7.1 Background

Interest in providing automated or semi-automated allocation methods are growing, using a mixture of approaches. One such approach was utilized by Pearce et al. (2018) to combine ergonomics and make-span estimates with optimization to generate concrete schedules of human and robot work, minimizing human worker physical stress along with overall task length. This method is broadly representative of similar work that uses optimization, along with a variety of metrics, to incorporate robots into human work (Huang et al., 2023; Calzavara et al., 2023; Monguzzi et al., 2022; Battini et al., 2016). Especially within the context of ergonomics analysis, these methods appear to offer a good fit, given that ergonomics metrics and computation techniques, such as the Strain Index (SI) (Garg et al., 2007, 2017), energy expenditure (EE) (Garg, 1976), and the Revised NIOSH Lifting Equation (Waters et al., 1994) generally feature some degree of non-summative computation, meaning that to understand the ergonomics of a task, the entire task - performed over the course of an entire day's of work - must be considered, rather than individual components. These holistic metrics can be represented as objectives, optimizing over complete variations or candidate allocations.

To understand how these allocations may work, we must first understand the fundamental activities that comprise jobs. One of the first formal representations of actions was *therbligs*, (Gilbreth and Gilbreth, 1924), used to model human work by reducing work activities to their most basic components. The combination of these basic motions represent a manual job, and can be utilized to study manual human work across a variety of tasks. Motion studies were then developed to analyze these primitive elements of manual work and create time estimates for these actions. Several predetermined time systems were created, but some of the most common include Methods-Time Measurement (MTM) (Maynard et al., 1948),

Modular Arrangement of Predetermined Time Standards (MODAPTS) (Carey et al., 2001), and Maynard Operation Sequence Time (MOST) (Zandin, 1990). These time systems estimate the time of a certain human work element based off common factors including distance, weight, and type of control. The tools are used prescriptively, helping to model the manual work in order to identify areas for improvement. In this way, primitives like *Therbligs* can be used to describe tasks, which can then be allocated to individual agents (either human or robot), and the results can be analyzed using a variety of ergonomics metrics. This is the approach used by Pearce et al. (2018), who used *Therbligs* and SI with an optimization-based approach to allocate various activities between humans and cobots.

However, the results of the work by Pearce et al. (2018) showed that the tasks most suitable for cobots were those that were conducive to parallel work, were repetitive, and featured activities that the cobot could easily do. This last point makes sense, given that the more capability the robot had, the greater it approximated a full worker. Furthermore, the rationale for the disproportionate benefit of parallel work follows from the fact that the addition of another, complete worker to a job with limited dependencies means at best a doubling of efficiency. However, this finding raises certain flags for the efficacy of cobots. As discussed previously, the promise of cobots, as opposed to automation in general, is that they can work to improve quality, performance, or the experience of workers through collaborative transformations of existing tasks. Simply having them work side-by-side without interaction avoids this type of collaborative potential, and the benefits that might be gained from it.

However, it is also possible that this finding is in part a logical consequence of the existing work representations and allocation approaches currently available, where entire primitives (*e.g., therbligs*) are allocated wholesale to individual humans and robots according to any dependencies between them. Dependencies effectively serve as constraints on this optimization, limiting how movable and flexible the primitives may be within the overall process, and therefore serving only as limits for how much benefit might be gained. To understand how true collaboration may benefit human workers, a novel method of considering action primitives and

allocation is needed.

Collaborative interactions are inherently concurrent, but many feature specific situations where orderings or dependencies are enforced. For this type of logic, a model commonly used is the *Petri Net*, a directed bipartite graph consisting of two node types (*Places* and *Transitions*) Peterson (1977). In a Petri Net, *tokens* travel between *Places* via *Transitions*, such that incoming *Transition* arcs indicate dependencies of that *Transition* for execution, and the outgoing edges indicate which tokens are produced in which *Places*. Petri Nets have been used frequently to model concurrent workflows, and multiple varieties have been developed that include representations of time and more nuanced dependencies (Zuberek, 1991; Van der Aalst, 1998). For example, Workflow (WF) Nets are a variety of Petri Net that features a more concise representation of logical dependencies, as well as the concept of sink and source *Places*, where resources appear or disappear (Van der Aalst, 1998). Petri Nets are also being used to model and even allocate tasks between combinations of humans and robots, due to their innate concurrent modeling (Casalino et al., 2019; Hu and Chen, 2017; Ziparo et al., 2011). However, many of these approaches are limited by their level of detail about the features of the processes themselves, and usually focus on a single aspect relevant to collaboration, such as deadlocking. Our goal is to provide a unified task and action representation informed by primitives such as *Therbligs*, *Energy Expenditure*, and *Methods-Time Measurement* with a Petri Net-based implementation to support a robust, straightforward approach to collaborative task allocations between humans and robots.

7.2 System Design & Implementation

In this section, we will outline both the high-level design of the *Allocobot* system, as well as the specific implementation details of the current prototype.

Approach

As discussed, a limitation of current primitive representations is that they are not conducive for reasoning about joint human-robot collaborative work. While drawing inspiration from these previous techniques, *Allocobot* departs from them in several key ways in order to better represent dependencies, joint actions, and the characteristics of collaborative work. To begin, work is structured hierarchically, but allocations can occur at various levels, resulting in varying levels of collaboration. For example, entire tasks may be allocated to one agent, while in other cases a single task may be restructured as a joint activity. Dependencies like resources also follow this hierarchy, but collaborative allocations may be cross-sections of this hierarchy, as opposed to strict sub-trees. In other words, dependencies in a true collaborative task describe a set of partial orderings, frequently involving the state or identities of various components or parts. So long as these partial orderings are respected within cycles, the absolute ordering of activities in a collaborative task can deviate from their order in the original specification. While embracing this partial order representation for dependencies offers flexibility and realism, it also presents challenges, given that any truly collaborative process has a nontrivial chance of deviating at least slightly from an original specification from cycle to cycle. Indeed, flexibility in aspects like timing are known to show benefits for the overall experience of human users in tasks like handovers (Huang et al., 2015). This, along with the inherent unpredictability that comes with collaborative human work, makes it is important to consider more than just fixed schedules. Failing to do this means that these static plans may not be representative of the task in practice, and may therefore be fragile to minor perturbations from the optimal result.

To address these characteristics, we propose a holistic method of collaborative task allocation, *Allocobot*, that includes both a novel primitive representation, specifically made for compatibility with both joint and singular activities by robots and human workers, as well as a method by which these primitives can be organized and allocated.

Data and Representations

First, we consider the top-level collection of actions to be the *Job*. The *Job* includes a set of *Tasks*, which themselves include a set of concurrent *Primitives* which are executed during the *Task*. Whereas *therbligs* are meant to be indivisible, yet complete representations of actions, our *Primitives* do not have the latter restriction. In other words, *Allocobot's* primitives recreate the functionality of other representations' primitives through composition. For example, *Allocobot's Force Primitive* could be combined with its *Position* and *Use Primitives* to produce what would be considered a *therblig Use*, where a worker positions and uses a tool while applying force. *Primitives* are organized into *Tasks*, but can still be independently allocated, with the caveat that they remain coordinated temporally and spatially. Thus, in the *Use* example, if the *Force* component were particularly problematic to the human, but possible by the robot, a joint action could occur where the robot provides force assistance while the human worker guides the tool.

Unlike other representations that organize dependencies as simple partial order relationships between entire sub-tasks, *Allocobot* aims for a more granular approach, considering the parts that an individual *Task* requires and produces, as well as spatial and agent availability considerations. These parts are called *Targets*, and include *Precursors* (parts coming in from prior processes), *Intermediates* (parts produced during, but which aren't outputs of the job), *Products* (outputs of the job), and *Reusables* (parts like tools that persist across cycles). *Allocobot* spatially situates activities, such that individual *Tasks* can specify specific areas where the worker or robot may use their hands or gripper, respectively. These areas, or *Points of Interest* (POIs), also include standing locations, so that reachability to these hand POIs can be approximated. Finally, the agents themselves are represented as classes of potential workers or robots. For example, it is possible to consider the relative impact of varying types of skilled workers or models of robots. For a breakdown of each of these concepts, and their properties, see Figure 7.1.

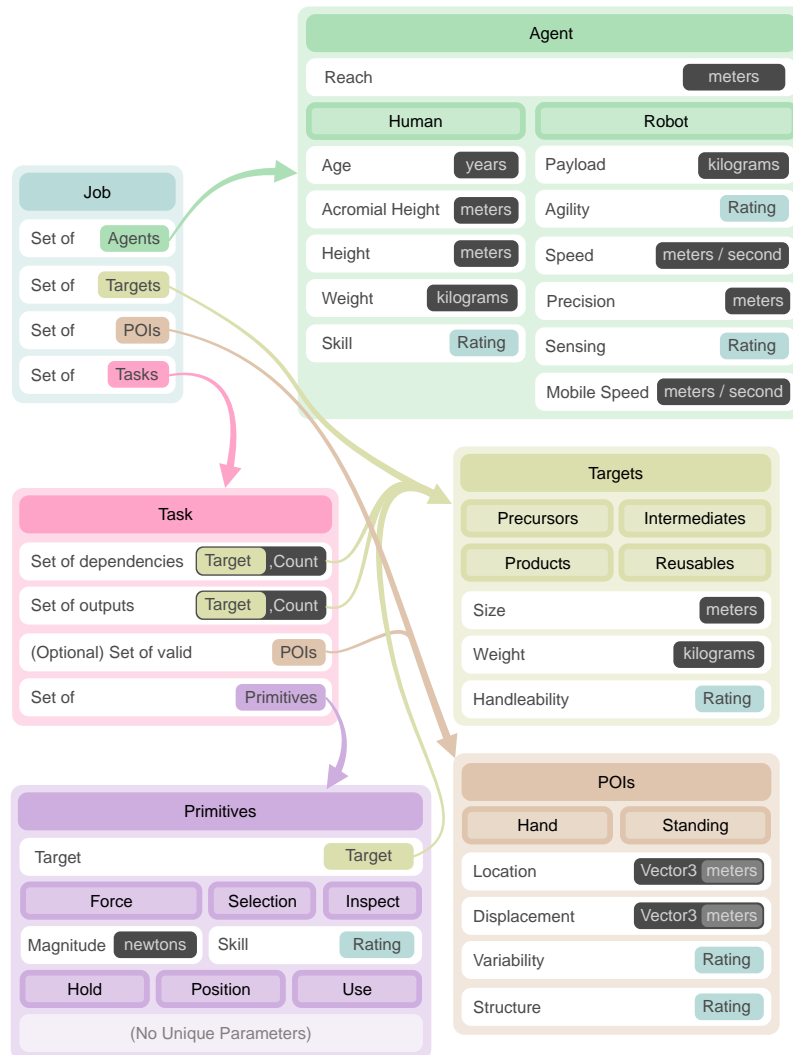


Figure 7.1: A mapping of the current components within *Allocobot*'s representation. Note, *Carry*, *Move*, *Travel*, and *Reach* Primitives are used internally within the algorithm, but are not specified explicitly, and therefore not shown. Overarching types are depicted as cards, where the solid header indicates the type. Any properties general across all types are listed first. Directly under outlined sub-type names are properties specific to that type or types. Rating is a simple Low/Medium/High categorical value.

Reformulation and Simulation

The above specification is performed by the user, after which the automated processing phase begins. For a complete overview of this workflow and the questions

asked in each phase, see Figure 7.2. The user-specified *Jobs* are then processed into an alternate Petri Net-based formulation Peterson (1977). Specifically, we use a modified Timed-transition Petri Net Zuberek (1991). This state machine representation is commonly used in representing manufacturing and workflow processes, and their emphasis on resource availability and timing is particularly suited for this type of modeling.

During this reformulation, the target dependencies are translated into tokens that must flow through the Petri Net. Similarly, *Agents* (both *human workers* and *robots*) are translated to tokens. Tasks are translated into *Transitions*, which consume and move these tokens around the state space, thereby representing the concurrent activities in the interaction. For example, if a given *Task* requires a certain *Precursor* and a given *Agent*, the *Transition* specifying that *Task* will have incoming arcs depicting those two requirements. If the *Task* produces an *Intermediate*, the outgoing arcs from the *Transition* would show both the *Agent* and the *Intermediate*, where they could be utilized in a later step.

This ability of tokens to both traverse the network while serving as constraints for actions allow for the network to encode two main decision types within the formulated Petri Nets, and both involve the execution of certain cost-inducing *Transitions*. The first represents the initial configuration or meta-parameters. These decisions include whether to add a given *Agent* type, which tasks certain agents can perform, and whether these are joint actions or solitary. These decisions are decided by the firing of certain irreversible *Transitions*, which serve to place *Agent* tokens in the simulation and enable other task *Transitions*. For example, hiring a worker, or purchasing a robot would be represented as a *Transition* which incurs a cost proportional to each option's monetary value, while depositing that *Agent's* token into the network for use. If the *Agent* is not added, the *Transitions* which would require it are effectively disabled, since the *Agent* resource token required to execute it is not present in the network. Conversely, if the *Agent* is added, those activities are enabled. These activities themselves constitute the second set of decisions, namely the timing and flow of the interaction itself. Based on the layout of the space, as well as the decisions within the aforementioned meta-parameter selection, *Agents*

and *Targets* will move throughout the space, producing the specified output.

The second set of *Transitions*, like those discussed before, also incur costs, but are more focused on the quality of the interaction, ergonomics, and production speed. A variety of methods have been combined to approximate and aggregate these ergonomics and collaborative-focused costs. The first, which we call one-time costs involves ones analogous to those mentioned previously. Each time a given *Task* is performed, some amount of penalty is applied. For humans, this cost is inspired by strain, and for robots, an assessment of robot capability matching. However, a great deal of ergonomic cost estimation involves the carryover effects of performing tasks repeatedly (e.g. fatigue), which these cost methods do not capture. To track how a given choice may impact these more temporally framed metrics, we introduce another method of cost estimation we call *exposure*. For these costs, we estimate the amount of strain induced for the hands, arms, and whole body. These estimates are translated to a corresponding number of tokens which are placed in designated *Agent*/type-specific *Place* bins. Each time the human *Agent* rests, or doesn't incur these costs, some number of tokens are removed from the bins. When assessed by the reinforcement learning algorithm, higher numbers of tokens in these bins are disincentivized. In this way, repeated activity by the human *Agent* without rest is avoided.

Once the Petri Net has been created, it serves as the scaffold for a process of Deep Reinforcement Learning (RL). Conceptually, this is achieved by simulating traces through the interaction, exploring various action possibilities and evaluating their impact on the overall performance of the agents in the task. Through this simulation, an RL agent representing the union of all user-specified *Agents* learns a policy that maps individual states to actions. In other words, at any moment in time, there exists a state vector equivalent to the marking, or layout, of tokens in the Petri Net. There are also a set of *Transition* actions that any of the included *Agents* can perform, based on the satisfiability of their input arcs. The combined set of actions that all *Agents* can perform at any time is the action vector, which are masked according to arc-token requirements. Concurrently executed actions are simply denoted by the presence of more than one threshold-satisfying entries in

this combined action vector.

The training of this deep network is done in two phases. The first shorter phase only serves to train the network to avoid deadlocks, as done by Hu et al. 2020. For a Petri Net, a deadlock occurs when for a given marking of tokens in the network, there are no valid *Transitions* that can fire. This initial training serves to train the RL agent to make choices that minimize these impasses. After this initial training phase, the second phase begins, in which the aforementioned costs specified by the Petri Net are utilized to refine the model to prioritize more effective, ergonomic, and efficient choices.

Analysis

After the learning phase is complete, the workflow returns to the user, where the model can be interrogated to answer questions like the overall cost (both monetary and ergonomic), and which allocations of work are most effective. Given the nature of the RL agent as the union of all *Agents*, the learned policy represents the preferred action for all *Agents* to perform given a current marking of the Petri Net. By executing the preferred actions at each time step, and feeding the result back into the RL agent, an "optimal" trace can be produced, documenting the state of the interaction at each moment, and the actions that each agent should perform. This trace can then be analyzed using conventional ergonomics approaches, or inspected for best-case efficiency. This is therefore analogous to the types of results an optimization-based approach might produce. However, the policy also allows for both greater realism since it inherently handles variation that might be caused by the human worker. By allowing the RL agent to select actions by the human *Agents* that are known to be less preferred, a variety of traces can be produced, other than the best-case scenario. Like the "optimal" trace, these too can be analyzed for ergonomic concerns, efficiency, and collaborative quality, therefore providing a distribution of outcomes.

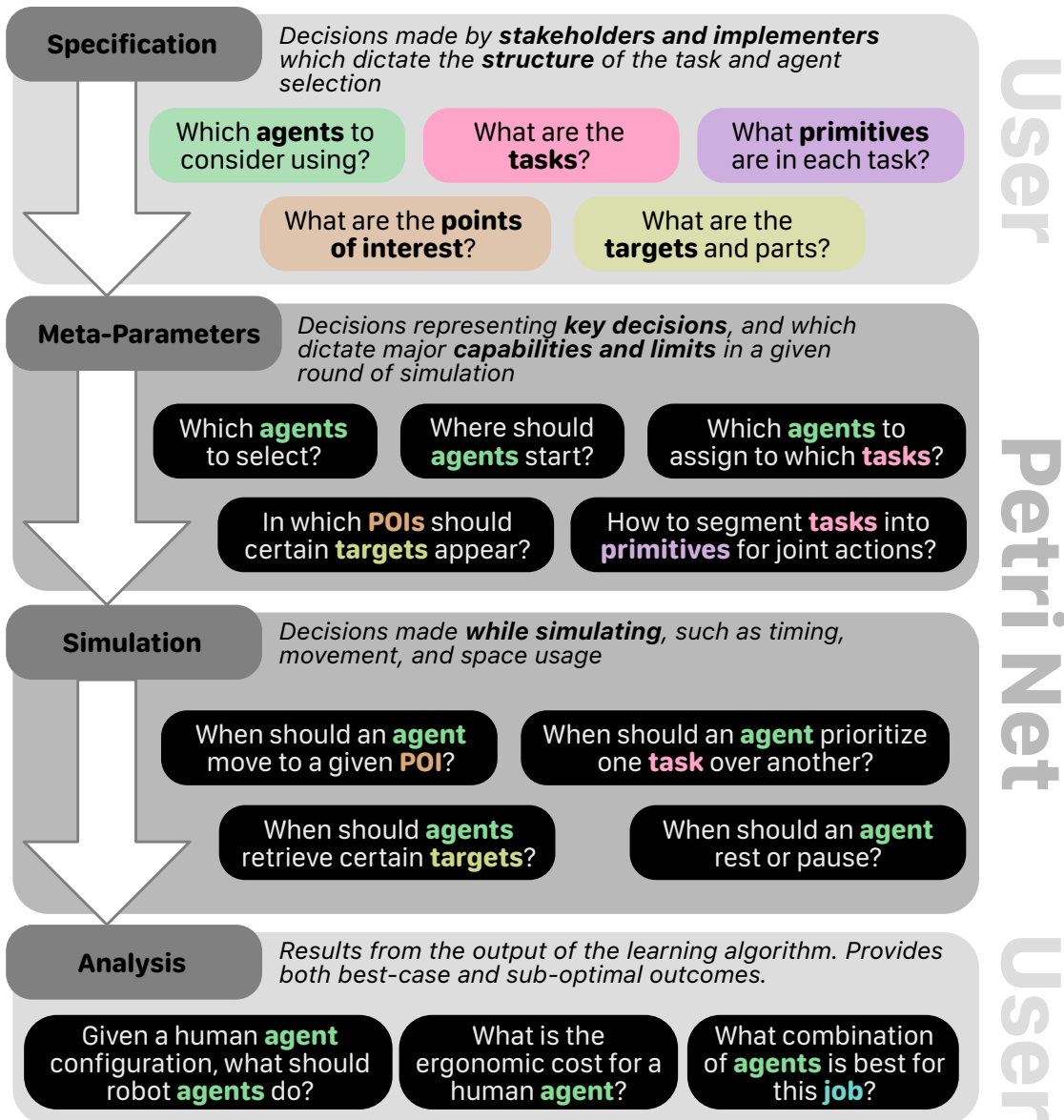


Figure 7.2: A graphic showing the flow of the *Allocobot* process, highlighting the questions specific to each phase. The first phase is specifications, and are inputs that the stakeholder or integrator might provide to detail the job. The second and third phases reflect the two decision types (meta-parameters and simulation). The fourth phase represents the types of questions that can be answered after the process.

Implementation

In this section, we describe in greater detail the specifics of our implementation of the aforementioned approach, with particular attention to the Petri Net, ergonomic estimation, and reinforcement learning.

Petri Net

A standard Petri Net can be defined theoretically as a four-tuple $N = (P, T, F, M_0)$, where N is the network, P and T are each disjoint finite sets representing the *Places* and *Transitions*, respectively, and $F \subseteq (P \times T) \cup (T \times P)$, or the set of directed, weighted arcs between P and T , and T and P Peterson (1977); Murata (1989). The initial marking, M_0 , or layout of tokens in P , is also commonly included, when present.

Petri nets have been extended in a variety of ways, including timing and alternative place representations. Timing can be implemented in one of two ways, *Timed-Transition Petri Nets*, and *Timed-Place Petri Nets*, which differ in the locus of time integration. In the former, each *Transition* takes a specified amount time, while in the latter, tokens must spend a certain amount of time in each *Place*. To better match the representation of the *Tasks* in the input representation, *Allocobot* utilizes a *Timed-Transition Petri Net*. Other extensions of Petri Nets include expanding the varieties and interpretations of various *Places* in the net Der Jeng (1997); Van der Aalst (1998). In *Allocobot*, we too specify a collection of *Places*, inspired by these varieties and the characteristics of our data representation:

- Source *Places* for *Agent* and *Precursor Target* tokens. For *Precursor Target* tokens, these effectively have a token count of infinity.
- Sink *Places* for *Agent* and *Product Target* tokens. These effectively have a token count of negative infinity.
- Exposure *Places* (Hand, Arm, and Whole Body variants), where exposure tokens are deposited following execution of human *Transitions* and consumed via rest or inactivity.

In addition to the above, we also utilize standard *Places* without unique functionality, which can encode various features, such as the presence of specific *Agents* and *Targets* at certain *POIs*, or the result of some previous choice (e.g., adding a robot *Agent*).

We also support a slight relaxation on some *Transition* guard functions, such that ranges of token consumption can vary within a specified range. These are useful in representing the ergonomic exposure tokens rest functionality. With this relaxed constraint, *Task Transitions* not contributing a certain type of exposure can serve as implicit rests by consuming those exposures from the exposure *Places*. This functionality is explored further in the *Ergonomic Cost Modeling* section.

Lastly, while the Petri Net model used by *Allocobot* is technically a non-colored Petri Net, each *Place* and *Transition* nevertheless encodes a rich amount of semantic information, such as the associated *Task*, *Targets*, *Primitive* assignments, *POIs*, and more. This semantic information is encoded as meta-data, which is used to assist in interpretation and training in later stages.

Ergonomic Cost Modeling

We utilized two methods of integrating human ergonomics costs into the *Allocobot* workflow. These two methods are a one-time ergonomic cost and an accumulating exposure cost.

Reinforcement learning conventionally enforces a cost on executing actions, such that the RL agent learns to be more efficient and choose efficient behaviors. This concept is used to provide the first method of integrating ergonomic considerations into the workflow. We call this a “one-time cost”, since it is incurred each time a given *Task Transition* is performed by an individual agent. This cost is estimated using a simplified fatigue method incorporating force and time. Worker characteristics such as strength are conventionally incorporated in these types of metrics by normalizing the cost to the amount of force needed, versus what can be supplied by the worker.

For the purpose of comparison, we consider how *Allocobot*'s primitives map

onto alternative representations. For a given model, some of these mappings are direct, while others are partial mappings. This mapping is shown in part within Figure 7.3.

Primitives	Definition	Therblig Representation	EE Representation	MTM Representation
Hold	Supporting the weight of a Target	<i>Grasp + Hold + Release</i>	<i>Hold</i> at arms length or at waist	NA
Tool Use	The process of using a tool (Reusable Target)	<i>Use</i>	Light or heavy <i>hand work</i>	NA
Position	Turning or rotating a Target	<i>Position</i>	Heavy/light <i>hand or arm work</i>	<i>Turn</i>
Force	Application of force	NA	<i>Pushing or Pulling</i>	<i>Apply Pressure (+) or Disengage (-)</i>
Carry	An Agent moves standing locations (POI) while holding a Target	<i>Transport Loaded</i>	<i>Carry</i> at arms length or at waist	<i>Grasp + Bend/Arise + Walk</i> obstructed or with weight + <i>Release</i>
Travel	An Agent moves from one standing location (POI) to another	<i>Transport Empty</i>	<i>Walking</i>	<i>Walk</i> unobstructed
Reach	An Agent moves their hands/gripper only	<i>Transport Empty</i>	<i>Forward movement</i> of arms, standing	<i>Reach</i> to fixed, slightly variable, or variable location
Move (horizontal displacement)	An Agent moves a Target in its hands/gripper	<i>Transport Loaded</i>	<i>Forward arm movement</i>	<i>Grasp + Move</i> (approximate or exact) + <i>Release</i>
Move (vertical displacement)	An Agent moves a Target in its hands/gripper	<i>Transport Loaded</i>	<i>Semi squat lift/lower</i>	<i>Grasp + Bend/Arise + Move</i> (approximate or exact) + <i>Release</i>

Figure 7.3: A description of the primitives utilized in the algorithm and the mapping onto different ergonomic models. Primitives in gray are used internally within the algorithm.

An important aspect for ergonomics analysis is the proportion of work to rest. This feature is why optimization-based approaches have historically been natural fits for allocation problems, since an assessment of the entire workflow is possible for a

given allocation. Petri Nets are memory-less, meaning that this type of computation is less directly translatable. The answer to this, however, is the creation of what we refer to as “exposure” cost *places*. Conceptually, these *places* serve as gauges on the combined magnitude and frequency of work that has been done recently by various parts of the body (hands, arms, and whole-body). The exposure *places* accumulate tokens from outgoing *Task Transitions* proportional to the fatigue that would be generated from that activity in that part of the body. When other *Task Transitions* are executed which don’t employ that part of the body, or the human *Agent* explicitly rests, these tokens are reduced. Therefore, if the human *Agent* repeatedly executes demanding tasks using some part of their body, the tokens in this *place* increase, while repeatedly resting or swapping the predominant body part will reduce them. To determine the appropriate body region breakdowns and corresponding token computation methods, traditional ergonomic models were considered, such as Strain Index (Garg et al., 2007, 2017), Garg Metabolic Prediction Model (EE) (Garg et al., 1978), NIOSH Lifting Index (Waters et al., 1994), Hand Activity Level (American Conference of Governmental Industrial Hygienists, 2018), Shoulder and Upper Extremity Threshold Limit Value (TLV) (American Conference of Governmental Industrial Hygienists, 2022), and Rohmert Curves (Rohmert, 1960). While the specific equations for computing each *Task Transition*’s contribution to the exposure *places* are not yet decided, it will likely focus on the usage of force both for its presence in many of these models, and as a complement to the one-time costs.

As mentioned, it is useful to segment which part of the body each exposure metric might be situated. A variety of different partition methods are possible, but given the fidelity of our spatial modeling, the segmentation which most effectively uses this geometry while providing useful distinctions is one of hands, arms/shoulders, and whole-body. More specifically, hands work refers to cases where the predominant activity involves small manipulation using the digits and precision work. Arm and Shoulder exertions include reaching in awkward postures, overhead work, carrying or manipulating heavy parts. Whole body exertions include lifting/lowering, carrying, holding objects, or moving objects. It should be stated

that this distinction is not exclusive, since individual *Tasks* can activate any number of these regions. This classification is based on the representation of MODAPTS Carey et al. (2001), incorporating information about the relationships between hand *POIs*, standing *POIs*, *Agent* metrics, and *Primitives*. After the exertions are classified, the token value is calculated based on the same values.

As the exposure *Places* can accumulate tokens, rest or localized inactivity can also reduce the tokens. Therefore, by giving rest to the human and reducing the tokens in the bins, the algorithm was rewarded. Rest was determined by using a scaled time metric.

Time Estimation

Given the timed nature of our representation, and the fact that many *Task Transitions* are not actually observed in the original workflow, a critical stage in the algorithm is the estimation of a given *Task Transition*, by an *Agent* or *Agents*. For this problem, we utilized Methods-Time Measurement (MTM) by Barnes (1980), a predetermined time system tool used to analyze the basic motions in a job, task, or subtask. In ergonomic analysis, MTM is used to determine where there are ergonomic strains based on time (Barnes, 1980). MTM-1 is the most basic MTM system and consists of 9 basic elements: reach, move, turn, apply pressure, grasp, position, release, disengage, eye times (focus and travel), and body, leg, and foot motions. Within each basic motion, there are different classes, types, and features that determine how much time a basic motion will take (Barnes, 1980). For example, a turn motion includes features such as weight, size of object, and degrees. MTM has been used in HRC to describe the basic motions of both the human and the robot Bänziger et al. (2017); Malik and Bilberg (2019); Teiwes et al. (2016). Teiwes et al. used MTM to describe the manual work and automation potential seen in a Volkswagen plant, where a score of the potential, duration, and movement was created to evaluate the different workstations. Bänziger et al. (2017) created skills based on the basic motions defined in Teiwes et al. (2016) to include physical interaction between the human and robot. However, no task features were included in the skills.

The primitives defined in Figure 7.3 are mapped to the MTM-1 motions, similar to the mapping of the primitives in the ergonomic cost modeling. The mapping allow standard times to be calculated for each task within a job, given the nature of their *Primitives*. MTM-1 times are presented as tables with different times based on task characteristics (i.e. class of motion, weight and size of object). In order to efficiently implement MTM-1 into our algorithm, we created regression equations based off the tables in Barnes (1980) when equations were not provided. For these regressions, we estimated power curve relationships for short distances and linear relationships for longer distances. The regression analysis was conducted in Microsoft Excel. All estimated regression equations had an R Square value above 0.99, which is expected because the MTM-1 tables themselves were created based off of linear regression equations.

Reinforcement Learning

To find a suitable policy for navigating through the Petri net according to the various time and cost metrics discussed, we use reinforcement learning (RL) to explore and reinforce optimal choices within the network. We accomplish this using Proximal Policy Optimization (PPO) Schulman et al. (2017) combined with action masking Huang and Ontañón (2020) to speed up the process of finding a viable solution. PPO is a policy gradient reinforcement learning method that makes small and iterative step sizes to learn the optimal policy for a given problem. Action masking is a technique that allows us to constrain the *Transition* space of the Petri Net at each time step to only those that are possible, i.e. if the prerequisites of a given *Transition* are not met, then it is not considered as a possible *Transition* for that state. This removes the requirement of first training the network to avoid invalid *Transitions*, thereby focusing on valid options for a given state.

While training the RL agent, each time step must produce a selection of *Transitions* for all *Agents*. Each selected *Transition* induces a negative reward (cost) to the overall RL agent. Producing the *Product(s)* associated with the *Job* results in a large positive reward, but failing to do so before the any cycle time limit induces a

large negative reward. The cost associated with each *Transition* is the combination of the one-time ergonomic cost associated with the *Transition* itself, the evaluation of exposure cost tokens in the exposure *Places*, as described in Section 7.2, as well as any cost-related metrics from the meta-parameter selection. Ergonomic costs are additive, such that the cost associated with exposure is added to the one-time cost of the *Transition*.

We utilized the implementation of PPO action masking from Stable-Baselines3 Raffin et al. (2021) contribution repository. Using this, we created two custom gym environments for the agent to train in. The first custom environment provides initial training against deadlock situations, similar to Hu et al. (2020), providing the agent with no reward for any *Transition* it takes, and severely penalizing *Transitions* that result in no viable *Transitions* in the subsequent state. Deadlock checking is checked through the action mask, determining if there are any possible *Transitions* that can be taken, ignoring silent actions such as rests which don't progress the state of the interaction. If no valid *Transitions* are present, the RL agent is penalized and the training session is ended.

The action mask is based on the outgoing arcs for each place in the Petri Net as well as the time associated with the *Transitions* they connect to. This is done to ensure a lightweight masking function, so as to not heavily impact training times. The first step of the action mask is to mark all currently firing *Transitions* as invalid, as they are in progress. From this reduced set of viable *Transitions*, we look at the outgoing arcs of each place that connects to these *Transitions*, as these are the arcs that reduce the token count for a given place. Since a place cannot have a negative amount of tokens, as this would mean, for example, we would have a negative amount of parts or individuals, we mark all *Transitions* that would create negative counts of tokens as invalid. The remaining set of *Transitions* is marked valid and considered for the current step of training.

After training the RL agent in deadlock avoidance, the second round of training begins, focusing on the full process. In this environment, each *Transition* taken incurs some cost to the RL agent based on the aforementioned factors, adding a one-to-one cost of the number of tokens in each ergo bin to the cost of executing the

Transition. This cost results in the RL agent accumulating an increasingly negative reward, which it's trying to minimize. The RL agent is only rewarded positively once it reaches the goal state, which is marked as the final product or state represented in the human-only process, *i.e.*, when the collaboration is considered complete. In both training environments, the RL agent records an observation of the current state of the Petri Net as a vector, *i.e.*, the number of tokens in each place, as well as the amount of time left for each *Transition* that is in progress.

To use the Petri Net within the PPO algorithm, we represent the change to the network as a matrix, with rows acting as the *Places* in the Petri Net and columns being the *Transitions*. Each index $_{ij}$ in the matrix represents the net change in tokens at place $_i$ for *Transition* $_j$. For example, if *Transition* $_j$ consumes 1 token from place $_i$ and then produces 1 token for place $_i$, index $_{ij}$ is 0. Because each *Transition* has a time associated with its execution, we split this matrix into two, one for token consumption input and one for token production output. This allows us to first consume tokens, wait for the *Transition* to finish, and then produce the associated tokens. At each step of the training process, we get a new observation of the Petri Net by multiplying this matrix with the previous observation.

However, as each *Transition* takes some amount of time t to fire, we produce the new observation by multiplying the input matrix by the old observation, to represent the consumption of tokens from the prerequisite *Places* in the Petri Net. Then after time t has passed, the new observation is the result of multiplying the output matrix by the current observation, which may be different than the observation produced by multiplying the input matrix. The step of applying the output matrix represents the change in the Petri Net that occurs from the *Transition* producing tokens to the outgoing *Places*. This time t is tracked in the observation vector, where each *Transition* has an associated timer indicating whether it is available or currently firing.

Assuming a maximum firing of the Petri Net at each time step, meaning that all possible agents are performing an action, we update the timer of the firing *Transitions* by the minimum amount between them. For example, if the max firing has the human agent engaged with *Transition* $_A$ for 25 seconds and the robot agent

with $Transition_B$ for 30 seconds, the simulation advances by 25 seconds for the next time step, leaving 5 seconds remaining on the robot's $Transition_B$. At this point, the human agent would be free to perform a new task, but the robot agent remains busy.

This second round of training produces a policy network that we apply to the problem space and determine the preferred sequence of tasks to complete the collaborative job. This network allows us to iteratively step through the problem space and identify the changes to the exposure costs over time, *Task* allocations, timing of activities, and hiring or purchasing decisions, and more. The result of this stepping process is a timeline or trace. Just like the preferred action set can be used, it is also possible to introduce variability into the trace by having the RL agent select human actions that are not the preferred action, likely proportional to their relative preference in the policy. This allows us to explore the space of possible solutions and identify the impact of different decisions on the overall cost of the process as a distribution.

7.3 Future Work

While most of the implementation of the above has been completed, a number of components still need to be finished. First, the ergonomic cost modeling, especially the exposure cost, is still in progress. Currently, a force-sensitive fatigue metric is being used in both one-time costs and exposure costs, but additional metrics may be incorporated, such as Hand Activity Level (HAL) (American Conference of Governmental Industrial Hygienists, 2018) and Shoulder and Upper Extremity Threshold Limit Value (TLV) (American Conference of Governmental Industrial Hygienists, 2022).

With regards to economic and business-related decisions, more work needs to be done to incorporate the cost of hiring and purchasing decisions. Currently, the cost of hiring a worker or purchasing a robot is a one-time cost, but a greater level of consideration needs to be taken to scale the relative benefits and costs of these decisions, as well as to events such as producing product and consuming

precursors. Likely, this will take the form of converting the ergonomic metrics into a monetary value, using estimation of the cost for injury and worker retention challenges. This will provide a singular metric for the scaling of relative costs and benefits, as well as a clear outcome value for stakeholders to understand.

The process of reinforcement learning using the Petri Net task models has been completed on example tasks, but more work needs to be done to evaluate the effectiveness and efficiency of the approach in fully realistic scenarios. This includes multiple levels of testing. The first of these is an efficiency consideration, focusing on the speed of the algorithm and the ability to scale to larger problems. Our goal in the system is to provide useful feedback about the process in a reasonable amount of time, so that stakeholders can make informed decisions about the process. Second is an modeling test, focusing on the ability of the algorithm as a whole to reason about the aspects like ergonomics and logic. This is relevant because despite using multiple ergonomic models and robust knowledge of collaborative robotics and business, the behavior of the system is highly dependent on the synergy between these models and the functionality of the Petri net model and RL method. This type of test therefore ensures that the indeed the algorithm appears to be prioritizing the correct aspects of the problem, such as low ergonomic burden and efficient production. This can be accomplished by comparing the trace results of the algorithm without additional agents to current workflows using standard domain metrics. If the algorithm is indeed prioritizing the correct aspects of the interaction, then the results should be similar or better than the current workflow.

Lastly, we want to evaluate the algorithm in the manner it is intended to be used. To do this, we would conduct a series of tests on real-world tasks, looking to determine the quality of the solutions provided when introducing collaborative robots, comparing the result of the algorithm to the current workflow. This evaluation could also include consultation with the organizations from which these real-world tasks are drawn, in order to better understand the quality of the feedback and the understandability and actionability of the results.

7.4 Chapter Summary

The *Allocobot* project grew directly out of interviews with and observations of individuals working in manufacturing, and the challenges they face in integrating collaborative robots into their processes. Specifically, we observed how even with a system like *CoFrame*, which provides a unique method of supporting the design of high-quality and safe collaborative robot programs, the stakeholders and engineers that would be tasked with using it lacked a prerequisite, namely a clear understanding of what they wanted to achieve. Compounding this issue was the multi-faceted nature of how to arrive at that answer, which includes considerations such as robot capability, human ergonomics, collaboration quality and safety, and efficiency. This complexity meant that it was difficult to find individuals who could provide all these perspectives, let alone integrate them into a single solution with clearly articulated metrics for relevant stakeholders tasked with the decision-making process. *Allocobot* also grew from our own motivation to improve on the *Authr* system. Despite the clear benefits of *Authr*'s approach, we desired a better match between the task representation for both agents, a more robust method of evaluating ergonomics and costs, and a more realistic output that respects the variable nature of human behavior.

Therefore, in this chapter, we presented *Allocobot*, a novel approach to the problem of collaborative robot allocation. We first presented the motivation for the work, specifically the need for a more holistic approach to the problem of integrating cobots into manufacturing processes that includes a process representation suited for the analysis and generation of robust human-robot collaborative work designs. We then presented the *Allocobot* workflow, which is a novel combination of a user-specified task representation, a Petri Net-based reformulation, and reinforcement learning. We then described the implementation of the approach, including the Petri Net representation, ergonomic cost modeling, and reinforcement learning. Lastly, we discussed the future work that needs to be done to complete the implementation and evaluate the approach.

8 GENERAL DISCUSSION

Summary and Significance of Work

The preceding work, including both technical and empirical contributions, has been in service of demonstrating partial support for my thesis statement stating that **tools and systems which support domain experts during the programming process through the use of task and program representations, transformation, and relevant feedback can support the design of collaborative robot behaviors.** As discussed, the nature of collaborative robot interaction design is multi-faceted and complex, and system design can be challenging with regards to time, effort, and resources, which means that a thorough proof of this thesis will involve the work of more than just a dissertation. Within the work described within this dissertation, we aim to provide this partial support in a two-pronged approach, by using this strategic combination of representation, transformation, and feedback to both elevate the understanding and knowledge of the individuals using them, as well as produce collaborative robot programs which incorporate beneficial improvements or increased collaborative potential. With *Authr*, we adapted a representation of human work called *Therbligs* into a set of action primitives that could be utilized in a linear sequence of processes, much like the engineers or work specialists would be familiar with. By combining these primitives with verification and limited synthesis, individuals were able to construct collaborative programs which better balanced ergonomic cost and time through allocation of work. *CoFrame* leaned more heavily into improving the knowledge of the users, focusing on the development of a joint programming-learning platform which through the use of visual feedback and guidance based on a model of cobot expertise aims to assist users in developing the skills needed to program cobots. The programming approach used within *CoFrame* is deliberately made to complement the approach familiar to roboticists, while introducing concepts such as synchronization and dependency. The optimization-based *Lively* framework and *LivelyStudio* system focused on the specification of robotic motion, which is a domain simultaneously visual and mathematical. With this

approach, we aim to provide a more intuitive and accessible way to specify robotic motion that supports both joint-based and Cartesian motion, as well as precise goals and procedural motion. By allowing the arbitration of these various behavior objectives to be handled within an optimization, the goal was to allow less technical, but highly knowledgeable individuals to focus on the qualitative aspects of motion design, as opposed to the technical mathematics. This approach was evaluated with a formative evaluation, which was used to improve the design approach to better support the both the current workflows of roboticists and animators, as well as the capabilities and requirements of real-time robot motion. Stepping back from the design of complete systems and tools, *OpenVP* was developed as a framework for the development of visual programming languages for robotics. Robotics presents certain unique challenges, such as the need for rich feedback and tight integration with external systems (like visualizations), logic and flow paradigms, and customizable primitives. The design of *OpenVP* was informed by the design of both *CoFrame* and *LivelyStudio*, and aims to allow tool creators to focus on the design of the representations used, instead of low-level details about interactivity, visualization, serialization, and implementation. The goal in such a system is that by abstracting away these details under a single framework, more research can be done which focuses on the varying affordances and capabilities of different representations at different levels of the cobot programming process. Finally, we discuss *Allocobot*, a project in progress which is highly informed by both our own observations at site visits, and a survey of current literature and industry trends. The challenge we have seen is that even in systems like *CoFrame*, which aim to make the cobot programming process more accessible, the individuals at these companies lack the ability to conceptualize the program objective. In other words, even if tools like *CoFrame* and *Authr* can assist programmers in developing programs, they still need to know what to program. *Allocobot* aims to address this by building on the concepts presented in *Authr* to develop a richer specification and representation for tasks and human work, leveraging Petri Nets and reinforcement learning to generate suggestions for task structure, robot selection, and allocation of work that can be used to inform both stakeholders and programmers in decision-making.

As stated, however, this work only represents a partial support for this thesis. This is due to many factors, which will be discussed in the following sections. However, broadly, this is due to two main factors. First, the work presented here represents solutions that address only a subset of the challenges and considerations that are relevant to the design of truly collaborative robot interactions. As discussed, the design of such systems is complex and highly interconnected, featuring systems for sensing, gesture, reasoning, planning, motion, and more. Given that this work primarily concerns itself with the challenges of motion design, task design, and work allocation, it represents only a portion of the required work to prove this thesis in full. Second, many of the projects outlined within this dissertation are still ongoing, with research being done by myself and colleagues, for example in the case of *CoFrame*, *Lively*, and *Allocobot*.

That being said, the work presented here together represent our approaches to begin bridging the current knowledge and skills gap preventing the effective use of collaborative robots. Demonstrating this, we have conducted summative and technical evaluations, formative evaluations, case studies, and research-motivated design. We have designed highly functional, standalone systems and tools, and we hope they represent the beginning of a larger body of work that will continue to address the challenges of collaborative robot interaction design.

Challenges and Limitations

While limitations of the various works have been discussed within their respective chapters, it is still useful to consider how these fit within the larger set of challenges and limitations of the body of work.

Evaluation Challenges

Systems for programming robots present challenges in designing effective means of evaluating them, since there are a multitude of metrics that are relevant. Like any system evaluation, this may include self-reported metrics such as usability or satisfaction. Performance-based metrics can also be used, such as the time it

takes to program something, or whether some goal was achieved. However, what matters more in many cases is not whether a goal was strictly achieved, but the quality with which it was done. Moreover, in the realm of collaborative robotics, the simple detection of errors is not sufficient, since the distinction between desirable behavior and undesirable behavior exists not as a discrete, consistent boundary, but rather a somewhat subjective and highly conditional spectrum. For example, a robot may be programmed to move to a location, but the quality of that motion may be poor, or the motion may be unsafe. In such a case, the robot may still be able to achieve the goal, but the quality of the motion may be poor. In such a case, the evaluation of the system should not be based on whether the robot was able to achieve the goal, but rather the quality of the motion. Furthermore, it is somewhat dependent on the expertise of the programmer. Suppose the individual is an expert in animation, and they have the benefit of knowing the type of motion desired in a specific situation. While the behavior may be suboptimal in certain circumstances, there may be good rationale for its creation, and thus the evaluation should consider not just the motion itself, but whether the creator was able to successfully execute on their vision.

Adding complexity to this is that in many systems, the ones in this dissertation being no exception, the goal of the system is not necessarily to create a full end-to-end robot infrastructure, but rather move the needle in some way by extending functionality. To what extent do evaluations focus on the specific contribution of the system, versus the system as a whole? Furthermore, given the extremely diverse range of approaches and layers of cobot programming, what is the right comparison for a given system? Is it one from research, or the one most commonly used in practice by the intended users? In *CoFrame*, is the comparison a current system like Polyscope (Universal Robots, 2023) that is used commonly, but which suffers from clear usability issues, their online video-based learning platform (Universal Robots, 2021), or some other research-focused robot programming system such as CoSTAR (Paxton et al., 2017) which was designed for a different purpose than *CoFrame*. What if the approach is novel, transcending architectural layers, like in the case of *Lively* and *LivelyStudio*? These are important questions for which there

are not always clear answers. In this work, we have attempted to balance the need to evaluate the systems as a whole, without losing sight of the design process and motivations that led to their creation. Combined with development challenges (discussed below), this has led to a staggered approach to evaluation, such that the evaluations of these systems sometimes take place some time after the development has concluded, with different individuals leading the charge. For example, an evaluation of the *CoFrame* system is ongoing, but being led by another student Nathan White, and is therefore not discussed explicitly in this body of work. As such, the *CoFrame* system must rest on the strength of its research-motivated design, the quality of its implementation and features, and the assessment of reviewers familiar with the domain. That is not to say that there are no evaluations of these systems. *Authr* was evaluated with both technical and user evaluations, while *Lively* was evaluated with a formative evaluation and benchmarks.

Development Challenges

As mentioned, systems for programming robots are complex and multifaceted. Any research-focused system exists in a balance between the demonstration of some novel concept and the development of a system that is usable and useful. Going too far in the direction of illustrating the concept, even a beneficial feature may be hidden by a lack of general usability of the system. Going too far in the other direction can result in highly usable systems, but which lack clear research contributions. This is a challenge that is not unique to robotics, but is especially challenging in robotics due to the complexity of the systems involved. For example, tools like *Authr*, *CoFrame*, and *LivelyStudio* includes the front-facing interfaces, as well as the infrastructure that supports it, including components like data storage, program verification and optimization, trajectory planning, and more. The interface itself represents significant work, requiring implementation of components for visualizing 3D scenes, data entry, navigation, drag-and-drop, and handling interactivity between components. This complexity requires a sizeable investment of time and effort, and consequently individuals that can contribute to the development of the

system. In practice, this means assembling a team of fellow graduate students, as well as highly capable undergraduate students. Producing the work demonstrated in this dissertation was by necessity a group effort, and the success we have had is a testament to the capabilities and tenacity of my fellow collaborators.

Even with a highly capable team, the development of these systems takes a considerable amount of time, requiring strategic reuse and generalization of various subsystems. For example, development of both *CoFrame* and *LivelyStudio* required significant development in terms of reusable components for 3D visualization and visual programming, the latter becoming the *OpenVP* library.

Research-related limitations served as another constraint on the design of these systems. Research-focused systems in general suffer when they cannot be executed and tested by other researchers, and this is particularly true when systems are tightly coupled to specific hardware or server behaviors. For example, if the functionality of the system is dependent on querying a third-party API, or even a locally run server, if the third party changes their API, or research funds dry up to maintain the server, the system may be completely broken. In the interest of producing systems which survive beyond the duration of a single research project, many of these systems were designed to be as self-contained as possible. While this assists in longevity, it also places certain constraints on the computational resources available, and the types of functionality that can be supported.

Breadth Limitations

The vast number of considerations for practical, fully functional cobot systems presents a vast breadth of potential improvement. Within the work presented here, attention is primarily devoted to the understanding of mutual dependencies, sub-task allocation, and motion specification, given their importance in the transition from standard manufacturing robotics to cobots. However, this reflects only a small part of the overall picture, which includes considerations such as object and gesture recognition, environment modeling, grasping and manipulation, communication, and error recovery. Any sufficiently complete method for supporting cobot pro-

gramming should consider these aspects in addition to the ones of focus within this dissertation. It is our hope that the approaches presented here can extend or remain compatible with future work in these areas. For example, the composability and extendability of *Lively* should allow for future expansion to handle improved specification that supports grasp-focused collision masking, or even higher-level abstractions of posture. *CoFrame*, partly driven by its usage of *OpenVP* has been built with extendability in mind, such that new actions and functionality can be supported.

Future Work

The limitations regarding breadth imply a certain type of future work, by which additional components and considerations of the cobot programming process can be addressed. How does a system like *CoFrame* function when modeling unpredictability of the human more explicitly? If object sensing is required, what is the most effective manner with which this physical and visual information is incorporated into the programming process?

Certain directions for future work focus more on digging deeper into the functionality and capabilities of the systems themselves, and I see a number of opportunities in this form. There is always a tradeoff in the usage of physical robots during the programming process itself. While it has clear benefits in grounding the movements of the robot in the real world, it comes with it the requirements of having that device available. This places constraints on the accessibility of the system to newcomers (such as in the education space) and care should be taken to ensure that educational resources are not tied explicitly to the use of sometimes costly equipment. Thus, a natural direction for future work is to explore instances where the benefits of such physical systems could be attained through alternative technologies, such as AR/VR and higher-fidelity simulation. In cases where the physical robot is truly necessary, how can that functionality best be incorporated into more explicitly programmed architectures?

The allocation of tasks, as discussed in the *Authr* and *Allocobot* chapters, is

a complex process that involves considerations about ergonomics, workforce requirements, cobot capability, the physical environment, and more. While *Allocobot* attempts to extend to more of these features than *Authr*, it is still limited in its ability to model them in a way that is both high-fidelity and sufficiently feasible computationally to be useful. In part this is by design; there is generally a certain degree of uncertainty in both the workspace and human behavior that makes attempting to over-fit to a specific model ill-advised. However, improvements to human modeling, as well as effective use of fuzzy logic and uncertainty – while managing the complexity of the system and computation – may represent ways of attaining both goals.

Given the emphasis on safety, predictability, and efficiency for companies looking to add cobots, it is important to consider also the tradeoffs between "creative" problem-solving processes, such as real-time planning and error recovery, and more explicit, hard-coded solutions that behave in predictable ways. In some ways, the ideal cobot sits at the intersection of being highly capable of recovering from error and assisting the human worker in ways that are adaptive, while also being well-vetted and predictable such that potential injuries are exceedingly rare. This is an exceedingly difficult balance to strike, especially given the need for high-fidelity input and sensing needed for the former. This is one of the reasons why at the current moment, despite recent advances in the field of Large Language Models (LLMs), there is very little interest from the industrial manufacturing community. For them, the risk of injury if the robot behaves in an unexpected way is just too high to justify the possible benefit of a more capable, adaptive robot. However, as the capabilities of these models improve, as well as the ability to regulate, constrain, and categorize their output, this balance may change.

The challenge then becomes how to best incorporate these models into the programming process itself. Do action primitives still make sense as a way of categorizing certain behaviors? Regarding the higher-level considerations of task allocation, ergonomics and capability, how do these emerging technologies fit into the picture? Ergonomics itself is a complex field that considers multiple factors in its models and estimates. It is unlikely that LLMs will immediately obtain the level of

fidelity needed to completely replace these methods completely within the program architecture, but there are still places where they may be useful. For example, in the *Allocobot* process, part of the success of this algorithm is on the ability to specify both the current setup, but also brainstorm potential future setups. This ideation, which provides inputs into these more computationally heavy systems, may be an effective way of incorporating these technologies into the programming process.

Conclusion

This final section serves as a final summary of the points within my dissertation. In these collected works by myself and my colleagues we have developed customized representations (*e.g.*, primitives, architectures, and domain-specific languages) and paired them with augmentation and feedback methods in order to reduce the gap between what current cobot programmers can currently achieve, and what they need to be able for them to become more effective and useful. *Authr* borrows from a set of human action definitions called *Therbligs*, combining them with a verification and synthesis approach which determines how allocations of those actions can produce programs that are efficient and ergonomic. *CoFrame* leans on an empirical model of cobot expertise to create a programming environment that provides feedback based on this model to assist users in developing their knowledge and understanding of the domain. *Lively* and *LivelyStudio* focus on the specification of robotic motion, taking a composable representation under an optimization-based framework that allows for a more design-focused approach. *OpenVP* provides a new architecture for visual programming languages in robotics, and *Allocobot* represents ongoing research meant to unify a diverse range of design considerations into a method for assisting in the allocation and decision-making process.

In many ways, I see this work as a stepping back from the current space of cobot programming and attempting to address why despite the existence of many highly capable subsystems and algorithms in the domain, the adoption and utility of cobots remains low. It is not enough for these new technologies to be more capable,

or pass certain benchmarks. This is certainly important, but not the entire picture. We need to consider not just the technology, but the individuals who have to piece them all together, and ask ourselves if we are handing them a challenge that is beyond their current capabilities. To truly produce meaningful societal impact, we must address the abstractions and representations for these methods, and how they exist within the larger program model and relate to the users' mental models and current understanding. Sometimes this involves considering their current approaches and workflows, and producing systems which incrementally move the needle towards more capable systems, but in a way that through feedback improves their understanding. Other times, it means creating novel representations which can be combined with various algorithms and methods to generate more comprehensive collaborative programs.

While the work presented here represents a large amount of time, it is by no means complete or comprehensive. There are a multitude of ways by which the current work can be further studied, extended, and modified. I hope that by presenting this material in this way, I motivate others to consider the ways in which they can contribute to this space, and help to bridge the gap between the current state of cobot programming and the future we envision.

REFERENCES

- Van der Aalst, Wil MP. 1998. The application of petri nets to workflow management. *Journal of circuits, systems, and computers* 8(01):21–66.
- Agility Robotics. 2023. Agility Robotics. <https://agilityrobotics.com>.
- Ajaykumar, Gopika, and Chien-Ming Huang. 2020. User needs and design opportunities in end-user robot programming. In *Companion of the 2020 acm/ieee international conference on human-robot interaction*, 93–95.
- Akgun, Baris, Maya Cakmak, Jae Wook Yoo, and Andrea Lockerd Thomaz. 2012. Trajectories and keyframes for kinesthetic teaching: A human-robot interaction perspective. In *Proceedings of the seventh annual acm/ieee international conference on human-robot interaction*, 391–398.
- Akrout, H., D. Anson, G. Bianchini, A. Neveur, C. Trinel, M. Farnsworth, and T. Tomiyama. 2013. Maintenance Task Classification: Towards Automated Robotic Maintenance for Industry. *Procedia CIRP* 11:367 – 372.
- Alexandrova, Sonya, Zachary Tatlock, and Maya Cakmak. 2015. Roboflow: A Flow-Based Visual Programming Language for Mobile Manipulation Tasks. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, 5537–5544. IEEE.
- American Conference of Governmental Industrial Hygienists, ed. 2018. *Hand activity: TLV physical agents 7th edition*.
- . 2022. *Upper limb localized fatigue: TLV physical agents 7th edition*.
- Andrew, Megan, Timothy Marler, Jesse Lastunen, Hannah Acheson-Field, and Steven W Popper. 2020. *An Analysis of Education and Training Programs in Advanced Manufacturing Using Robotics*. RAND.

Andrist, Sean, Wesley Collier, Michael Gleicher, Bilge Mutlu, and David Shaffer. 2015. Look together: Analyzing gaze coordination with epistemic network analysis. *Frontiers in psychology* 6:1016.

Andrist, Sean, Michael Gleicher, and Bilge Mutlu. 2017. Looking coordinated: Bidirectional gaze mechanisms for collaborative interaction with virtual characters. In *Proceedings of the 2017 chi conference on human factors in computing systems*, 2571–2582.

Andrist, Sean, Andrew R Ruis, and David Williamson Shaffer. 2018. A network analytic approach to gaze coordination during a collaborative task. *Computers in Human Behavior* 89:339–348. Publisher: Elsevier.

Andrist, Sean, Xiang Zhi Tan, Michael Gleicher, and Bilge Mutlu. 2014. Conversational gaze aversion for humanlike robots. In *2014 9th ACM/IEEE International Conf. on Human-Robot Interaction (HRI)*, 25–32. IEEE.

Asselborn, T., W. Johal, and P. Dillenbourg. 2017. Keep on moving! Exploring anthropomorphic effects of motion during idle moments. In *2017 26th IEEE International Symp. on Robot and Human Interactive Communication (RO-MAN)*, 897–902.

Autodesk. 2023. Autodesk Maya. <https://www.autodesk.com/products/maya/overview>.

Autor, David. 2021. Good News: There's a Labor Shortage. *The New York Times*.

Bangor, Aaron, Philip Kortum, and James Miller. 2008. An Empirical Evaluation of the System Usability Scale. *Intl. Journal of Human-Computer Interaction* 24(6): 574–594. Publisher: Taylor & Francis.

Bänziger, Timo, Andreas Kunz, and Konrad Wegener. 2017. A library of skills and behaviors for smart mobile assistant robots in automotive assembly lines. In *Proceedings of the companion of the 2017 acm/ieee international conference on human-robot interaction*, 77–78.

Barnes, R.M. 1980. *Motion and time study: Design and measurement of work*. 7th ed. John Wiley & Sons.

Battini, Daria, Xavier Delorme, Alexandre Dolgui, Alessandro Persona, and Fabio Sgarbossa. 2016. Ergonomics in assembly line balancing based on energy expenditure: a multi-objective model. *International Journal of Production Research* 54(3): 824–845.

Beck, Aryel, Lola Cañamero, Antoine Hiolle, Luisa Damiano, Piero Cosi, Fabio Tesser, and Giacomo Sommavilla. 2013a. Interpretation of emotional body language displayed by a humanoid robot: A case study with children. *International Journal of Social Robotics* 5(3):325–334. Publisher: Springer.

Beck, Aryel, Antoine Hiolle, and Lola Cañamero. 2013b. Using Perlin Noise to Generate Emotional Expressions in a Robot. *CogSci*.

Belpaeme, Tony, Paul E Baxter, Robin Read, Rachel Wood, Heriberto Cuayáhuatl, Bernd Kiefer, Stefania Racioppa, Ivana Kruijff-Korabayová, Georgios Athanasopoulos, Valentin Enescu, Rosemarijn Looije, Mark Neerincx, Yiannis Demiris, Raquel Ros-Espinoza, Aryel Beck, Lola Cañamero, Antione Hiolle, Matthew Lewis, Ilaria Baroni, Marco Nalin, Piero Cosi, Giulio Paci, Fabio Tesser, Giacomo Sommavilla, and Remi Humbert. 2013. Multimodal Child-Robot Interaction: Building Social Bonds. *Journal of Human-Robot Interaction* 1(2):1–21.

Berger, Suzanne, and Benjamin Armstrong. 2022. The puzzle of the missing robots.

Berk, Laura E, and Adam Winsler. 1995. *Scaffolding children's learning: Vygotsky and early childhood education. naeyc research into practice series. volume 7*. ERIC.

Billard, Aude, Sylvain Calinon, Ruediger Dillmann, and Stefan Schaal. 2008. Survey: Robot programming by demonstration. Tech. Rep., Springer.

Blender Foundation. 2023. Blender. <https://www.blender.org>.

- Bodenheimer, Bobby, Anna V Shleyfman, and Jessica K Hodgins. 1999. The effects of noise on the perception of animated human running. In *Computer Animation and Simulation'99*, 53–63. Springer.
- Bollini, Mario, Stefanie Tellex, Tyler Thompson, Nicholas Roy, and Daniela Rus. 2013. Interpreting and Executing Recipes with a Cooking Robot. In *Experimental Robotics*, 481–495. Springer.
- Boston Dynamics. 2023. Boston Dynamics. <https://bostondynamics.com>.
- Breazeal, Cynthia, Kerstin Dautenhahn, and Takayuki Kanda. 2016. Social robotics. *Springer handbook of robotics 1935–1972*. Publisher: Springer.
- Brohan, Anthony, Noah Brown, Justice Carbajal, Yevgen Chebotar, Xi Chen, Krzysztof Choromanski, Tianli Ding, Danny Driess, Avinava Dubey, Chelsea Finn, et al. 2023. Rt-2: Vision-language-action models transfer web knowledge to robotic control. *arXiv preprint arXiv:2307.15818*.
- Brooke, John. 1996. SUS-A Quick and Dirty Usability Scale. *Usability Evaluation in Industry* 189(194):4–7. Publisher: London.
- Bugmann, Guido, Ewan Klein, Stanislaw Lauria, and Theodor Kyriacou. 2004. Corpus-Based Robotics: A Route Instruction Example. In *Proceedings of Intelligent Autonomous Systems*, 96–103.
- Calzavara, Martina, Maurizio Faccio, and Irene Granata. 2023. Multi-objective task allocation for collaborative robot systems with an industry 5.0 human-centered perspective. *The International Journal of Advanced Manufacturing Technology* 128(1-2): 297–314.
- Carey, P, J Farrell, M Hui, and B Sullivan. 2001. *Heyde's modapts: A language of work*. Heyde Dynamics Pty Ltd.
- Casalino, Andrea, Andrea Maria Zanchettin, Luigi Piroddi, and Paolo Rocco. 2019. Optimal scheduling of human–robot collaborative assembly operations with time petri nets. *IEEE Transactions on Automation Science and Engineering* 18(1):70–84.

- Cha, Elizabeth, Maja Matarić, and Terrence Fong. 2016. Nonverbal signaling for non-humanoid robots during human-robot collaboration. In *2016 11th ACM/IEEE International Conference on Human-Robot Interaction (HRI)*, 601–602.
- Chen, Juo-Tung, and Chien-Ming Huang. 2023. Forgetful large language models: Lessons learned from using llms in robot programming. *arXiv preprint arXiv:2310.06646*.
- Chi, Diane, Monica Costa, Liwei Zhao, and Norman Badler. 2000. The EMOTE model for effort and shape. In *Proceedings of the 27th annual Conf. on Computer graphics and interactive techniques*, 173–182.
- Chi, Michelene TH, Paul J Feltovich, and Robert Glaser. 1981. Categorization and representation of physics problems by experts and novices. *Cognitive science* 5(2): 121–152. Publisher: Elsevier.
- Chitta, Sachin, Ioan Sucan, and Steve Cousins. 2012. Moveit![ros topics]. *IEEE Robotics & Automation Magazine* 19(1):18–19. Publisher: IEEE.
- Chrisinger, David. 2019. The solution lies in education: artificial intelligence & the skills gap. *On the Horizon*. Publisher: Emerald Publishing Limited.
- Christiernin, Linn Gustavsson. 2017. How to describe interaction with a collaborative robot. In *Proceedings of the Companion of the 2017 ACM/IEEE International Conference on Human-Robot Interaction*, 93–94.
- Chryssolouris, G, D Mavrikios, and L Rentzos. 2016. The teaching factory: A manufacturing education paradigm. *Procedia Cirp* 57:44–48. Publisher: Elsevier.
- Coumans, Erwin, and Yunfei Bai. 2016. PyBullet, a Python module for physics simulation for games, robotics and machine learning.
- Council, National Research. 2000. *How people learn: Brain, mind, experience, and school: Expanded edition*. Washington, DC: The National Academies Press.

Cuijpers, Raymond H, and Marco AMH Knops. 2015. Motions of robots matter! the social effects of idle and meaningful motions. In *International Conference on Social Robotics*, 174–183. Springer.

Dagdilelis, Vassilios, Maya Sartatzemi, and Katerina Kagani. 2005. Teaching (with) robots in secondary schools: some new and not-so-new pedagogical problems. In *Fifth IEEE International Conference on Advanced Learning Technologies (ICALT'05)*, 757–761. IEEE.

Datta, Chandan, Chandimal Jayawardena, I Han Kuo, and Bruce A MacDonald. 2012. RoboStudio: A visual programming environment for rapid authoring and customization of complex services on a personal service robot. In *2012 IEEE/RSJ International Conf. on Intelligent Robots and Systems*, 2352–2357. IEEE.

De Meijer, Marco. 1989. The contribution of general features of body movement to the attribution of emotions. *Journal of Nonverbal behavior* 13(4):247–268. Publisher: Springer.

De Moura, Leonardo, and Nikolaj Bjørner. 2008. Z3: An Efficient SMT Solver. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, 337–340. Springer.

Der Jeng, Mu. 1997. A petri net synthesis theory for modeling flexible manufacturing systems. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)* 27(2):169–183.

Desai, Ruta, Fraser Anderson, Justin Matejka, Stelian Coros, James McCann, George Fitzmaurice, and Tovi Grossman. 2019. Geppetto: Enabling semantic design of expressive robot behaviors. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*, 1–14.

diSessa, Andrea A. 1988. Knowledge in pieces. In *Constructivism in the computer age.*, 49–70. The Jean Piaget symposium series., Hillsdale, NJ, US: Lawrence Erlbaum Associates, Inc.

- Dréo, Johann, Pierre Savéant, Marc Schoenauer, and Vincent Vidal. 2011. Divide-and-Evolve: The Marriage of Descartes and Darwin. *Proceedings of the 7th International Planning Competition (IPC)*. Freiburg, Germany 91:155.
- Duffy, Brian R. 2003. Anthropomorphism and the social robot. *Robotics and Autonomous Systems* 42(3-4):177–190.
- . 2008. Fundamental Issues in Affective Intelligent Social Machiness. *The Open Artificial Intelligence Journal* 2(1).
- Duffy, Brian R, and Karolina Zawieska. 2012. Suspension of disbelief in social robotics. In *2012 RO-MAN: The 21st IEEE International Symp. on Robot and Human Interactive Communication*, 484–489. IEEE.
- Egges, A., T. Molet, and N. Magnenat-Thalmann. 2004. Personalised real-time idle motion synthesis. In *12th Pacific Conf. on Computer Graphics and Applications, 2004. PG 2004. Proceedings.*, 121–130.
- El Zaatari, Shirine, Mohamed Marei, Weidong Li, and Zahid Usman. 2019. Cobot Programming for Collaborative Industrial Tasks: An Overview. *Robotics and Autonomous Systems* 116:162–180. Publisher: Elsevier.
- Elprama, BVSA, Ilias El Makrini, and A Jacobs. 2016. Acceptance of collaborative robots by factory workers: a pilot study on the importance of social cues of anthropomorphic robots. In *International symposium on robot and human interactive communication*, vol. 7.
- Epic Games. 2023. Unreal Engine. <https://www.unrealengine.com/en-US>.
- Ericsson, K. Anders, and Herbert Simon. 1998. How to Study Thinking in Everyday Life: Contrasting Think-Aloud Protocols with Descriptions and Explanations of Thinking. *Mind, Culture, and Activity* 5(3):178–186. Publisher: Taylor & Francis.
- Ernst, Jette, and Charlotte Jonasson. 2020. Serving robots?—Exploring human and robot social dynamics in everyday hospital work. In *36th EGOS Colloquium 2020: Organizing for a Sustainable Future: Responsibility, Renewal & Resistance*.

Fantini, Paola, Marta Pinzone, Franco Sella, and Marco Taisch. 2017. Collaborative robots and new product introduction: capturing and transferring human expert knowledge to the operators. In *International Conference on Applied Human Factors and Ergonomics*, 259–268. Springer.

Fanuc. 2023. Fanuc. <https://www.fanucamerica.com>.

Fast-Berglund, Åsa, Filip Palmkvist, Per Nyqvist, Sven Ekered, and Magnus Åkerman. 2016. Evaluating cobots for final assembly. *Procedia CIRP* 44:175–180. Publisher: Elsevier.

Fischer, Kerstin. 2019. Why collaborative robots must be social (and even emotional) actors. *Techne: Research in Philosophy & Technology* 23(3).

Fogli, Daniela, Luigi Gargioni, Giovanni Guida, and Fabio Tampalini. 2022. A hybrid approach to user-oriented programming of collaborative robots. *Robotics and Computer-Integrated Manufacturing* 73:102234.

Fox, Maria, and Derek Long. 2003. Pddl2. 1: An extension to pddl for expressing temporal planning domains. *Journal of artificial intelligence research* 20:61–124.

Fraser, C. Ailie, Tovi Grossman, and George Fitzmaurice. 2017. WeBuild: Automatically Distributing Assembly Tasks Among Collocated Workers to Improve Coordination. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*, 1817–1830. CHI '17, New York, NY, USA: ACM. Event-place: Denver, Colorado, USA.

Fraser, Neil. 2015. Ten things we've learned from Blockly. In *2015 IEEE Blocks and Beyond Workshop (Blocks and Beyond)*, 49–50. IEEE.

Fryman, Jeff, and Bjoern Matthias. 2012. Safety of industrial robots: From conventional to collaborative applications. In *ROBOTIK 2012; 7th German Conference on Robotics*, 1–5. VDE.

Galín, Rinat, and Roman Meshcheryakov. 2019. Automation and robotics in the context of Industry 4.0: the shift to collaborative robots. In *IOP Conference Series: Materials Science and Engineering*, vol. 537, 032073. IOP Publishing. Issue: 3.

Gao, Yuxiang, and Chien-Ming Huang. 2019. PATI: a Projection-Based Augmented Table-Top Interface for Robot Programming. In *Proceedings of the 24th International Conference on Intelligent User Interfaces*, 345–355.

Garg, Arun. 1976. A metabolic rate prediction model for manual materials handling jobs. Ph.D. thesis, University of Michigan. Copyright - Database copyright ProQuest LLC; ProQuest does not claim copyright in the individual underlying works; Last updated - 2023-07-26.

Garg, Arun, Don B. Chaffin, and Gary D. Herrin. 1978. Prediction of metabolic rates for manual materials handling jobs. *American Industrial Hygiene Association Journal* 39(8):661–674.

Garg, Arun, J Steven Moore, and Jay M Kapellusch. 2007. The strain index to analyze jobs for risk of distal upper extremity disorders: Model validation. In *2007 IEEE International Conference on Industrial Engineering and Engineering Management*, 497–499. IEEE.

———. 2017. The revised strain index: an improved upper extremity exposure assessment model. *Ergonomics* 60(7):912–922.

Giannopoulou, Georgia, Elsi-Mari Borrelli, and Fiona McMaster. 2021. "Programming-It's not for Normal People": A Qualitative Study on User-Empowering Interfaces for Programming Collaborative Robots. In *2021 30th IEEE International Conference on Robot & Human Interactive Communication (RO-MAN)*, 37–44. IEEE.

Giffi, Craig, Paul Wellener, Ben Dollar, Heather Ashton Manolian, Luke Monck, and Chad Moutray. 2018. Deloitte and The Manufacturing Institute skills gap and future of work study. *Deloitte Insights*.

Gilbreth, Frank, and Lilian Gilbreth. 1924. Classifying the Elements of Work. *Management and Administration* 8(2):151–154.

Glas, Dylan F, Takayuki Kanda, and Hiroshi Ishiguro. 2016. Human-robot interaction design using interaction composer eight years of lessons learned. In *2016 11th ACM/IEEE International Conf. on Human-Robot Interaction (HRI)*, 303–310. IEEE.

Gleicher, Michael. 1998. Retargetting motion to new characters. In *Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, 33–42.

Gombolay, Matthew, Ronald Wilcox, and Julie Shah. 2018. Fast Scheduling of Robot Teams Performing Tasks With Temporospacial Constraints. *IEEE Transactions on Robotics* 34(1):220–239.

Gonulal, Talip, and Shawn Loewen. 2018. Scaffolding technique. *The TESOL encyclopedia of English language teaching* 1–5.

Google. 2019. Angular. Version Number: 7.2.0.

Grand View Research. 2023. Collaborative robots market size, share & trends analysis report by payload capacity, by application (assembly, handling, packaging, quality testing), by vertical, by region, and segment forecasts, 2023 - 2030. Tech. Rep. GVR-1-68038-371-3, Grand View Research.

Gualtieri, Luca, Erwin Rauch, and Renato Vidoni. 2021. Emerging research fields in safety and ergonomics in industrial collaborative robotics: A systematic literature review. *Robotics and Computer-Integrated Manufacturing* 67:101998. Publisher: Elsevier.

Guerin, Kelleher, Colin Lea, Chris Paxton, and Gregory Hager. 2015. A Framework for End-User Instruction of a Robot Assistant for Manufacturing. In *Robotics and Automation (ICRA), 2015 IEEE International Conference on*, 6167–6174. IEEE.

Haas, Andreas, Andreas Rossberg, Derek L Schuff, Ben L Titzer, Michael Holman, Dan Gohman, Luke Wagner, Alon Zakai, and JF Bastien. 2017. Bringing the web

up to speed with WebAssembly. In *Proceedings of the 38th ACM SIGPLAN Conf. on Programming Language Design and Implementation*, 185–200.

Harris, J., and E. Sharlin. 2011. Exploring the affect of abstract motion in social human-robot interaction. In *2011 RO-MAN*, 441–448.

Hart, Sandra, and Lowell Staveland. 1988. Development of NASA-TLX (Task Load Index): Results of Empirical and Theoretical Research. In *Advances in Psychology*, vol. 52, 139–183. Elsevier.

Heider, Fritz, and Marianne Simmel. 1944. An experimental study of apparent behavior. *The American journal of psychology* 57(2):243–259. Publisher: JSTOR.

Höller, Daniel, Gregor Behnke, Pascal Bercher, Susanne Biundo, Humbert Fiorino, Damien Pellier, and Ron Alford. 2020. Hddl: An extension to pddl for expressing hierarchical planning problems. In *Proceedings of the aaai conference on artificial intelligence*, vol. 34, 9883–9891.

Holm, Jacob Rubæk, Edward Lorenz, and Jørgen Stamhus. 2021. The impact of robots and AI/ML on skills and work organisation. In *Globalisation, New and Emerging Technologies, and Sustainable Development*, 149–168. Routledge.

Horst, John, Elena Messina, Jeremy Marvel, and others. 2021. Best Practices for the Integration of Collaborative Robots into Workcells Within Small and Medium-Sized Manufacturing Operations. *National Institute of Standards and Technology Advanced Manufacturing Series 100-41* 21 pages. Publisher: Advanced Manufacturing Series (NIST AMS), National Institute of Standards.

Hu, Bin, and Jing Chen. 2017. Optimal task allocation for human–machine collaborative manufacturing systems. *IEEE Robotics and Automation Letters* 2(4): 1933–1940.

Hu, Liang, Zhenyu Liu, Weifei Hu, Yueyang Wang, Jianrong Tan, and Fei Wu. 2020. Petri-net-based dynamic scheduling of flexible manufacturing system via deep

reinforcement learning with graph convolutional network. *Journal of Manufacturing Systems* 55:1–14.

Huang, Chien-Ming, Maya Cakmak, and Bilge Mutlu. 2015. Adaptive Coordination Strategies for Human-Robot Handovers. In *Robotics: science and systems*, vol. 11. Rome, Italy.

Huang, Chien-Ming, and Bilge Mutlu. 2016. Anticipatory robot control for efficient human-robot collaboration. In *2016 11th ACM/IEEE International Conference on Human-Robot Interaction (HRI)*, 83–90.

Huang, Chien-Ming, and Andrea L Thomaz. 2011. Effects of responding to, initiating and ensuring joint attention in human-robot interaction. In *2011 ro-man*, 65–71. IEEE.

Huang, Congfang, Shiyu Zhou, Jingshan Li, and Robert G Radwin. 2023. Allocating robots/cobots to production systems for productivity and ergonomics optimization. *IEEE Transactions on Automation Science and Engineering*.

Huang, Justin. 2017. Enabling Rapid End-to-End Programming of Mobile Manipulators. In *Proceedings of the Companion of the 2017 ACM/IEEE International Conference on Human-Robot Interaction*, 343–344. ACM.

Huang, Justin, and Maya Cakmak. 2017. Code3: A system for end-to-end programming of mobile manipulator robots for novices and experts. In *2017 12th ACM/IEEE International Conference on Human-Robot Interaction (HRI)*, 453–462. IEEE.

Huang, Justin, Tessa Lau, and Maya Cakmak. 2016. Design and evaluation of a rapid programming system for service robots. In *2016 11th acm/ieee international conference on human-robot interaction (hri)*, 295–302. IEEE.

Huang, Shengyi, and Santiago Ontañón. 2020. A closer look at invalid action masking in policy gradient algorithms. *arXiv preprint arXiv:2006.14171*.

Ishiguro, Hiroshi, and T Minato. 2005. Development of androids for studying on human-robot interaction. In *International Symp. on Robotics*, vol. 36, 5.

Jain, Eakta, Yaser Sheikh, Moshe Mahler, and Jessica Hodgins. 2010. Augmenting Hand Animation with Three-Dimensional Secondary Motion. In *Proceedings of the 2010 ACM SIGGRAPH/Eurographics Symp. on Computer Animation*, 93–102. SCA '10, Goslar, DEU: Eurographics Association. Event-place: Madrid, Spain.

Javaid, Mohd, Abid Haleem, Ravi Pratap Singh, Shanay Rab, and Rajiv Suman. 2022. Significant applications of cobots in the field of manufacturing. *Cognitive Robotics* 2:222–233.

Johnston, Ollie, and Frank Thomas. 1981. *The illusion of life: Disney animation*. Disney Editions New York.

Jun, Seung-kook, Pankaj Singhal, Madusudanan Sathianarayanan, Sudha Garimella, Abeer Eddib, and Venkat Krovi. 2012. Evaluation of Robotic Minimally Invasive Surgical Skills Using Motion Studies. In *Proceedings of the Workshop on Performance Metrics for Intelligent Systems*, 198–205. ACM.

Kakade, Siddhant, Bhmeshwar Patle, and Ashish Umbarkar. 2023. Applications of collaborative robots in agile manufacturing: a review. *Robotic Systems and Applications* 3(1):59–83.

Kato, Daishi, and Paul Henschel. 2019. Zustand: Bear necessities for state management in react. <https://docs.pmnd.rs/zustand/>.

Kato, Jun, Daisuke Sakamoto, Takeo Igarashi, and Masataka Goto. 2014. Sharedo: To-do List Interface for Human-agent Task Sharing. In *Proceedings of the Second International Conference on Human-agent Interaction*, 345–351. HAI '14, New York, NY, USA: ACM. Event-place: Tsukuba, Japan.

Khouadjia, Mostepha, Marc Schoenauer, Vincent Vidal, Johann Dréo, and Pierre Savéant. 2013. Pareto-Based Multiobjective AI Planning. In *International Joint Conference on Artificial Intelligence*. AAAI.

Kim, Seung Han, and Jae Wook Jeon. 2007. Programming LEGO Mindstorms NXT with Visual Programming. In *2007 International Conference on Control, Automation and Systems*, 2468–2472.

Klassner, Frank, and Scott Anderson. 2003. LEGO MindStorms: Not Just for K-12 Anymore. *IEEE Robotics Automation Magazine* 10(2):12–18.

Knight, Heather, and Reid Simmons. 2014. Expressive motion with x, y and theta: Laban effort features for mobile robots. In *The 23rd IEEE International Symp. on Robot and Human Interactive Communication*, 267–273. IEEE.

Ko, Andrew J, and Brad A Myers. 2005. Human factors affecting dependability in end-user programming. In *Proceedings of the first workshop on end-user software engineering*, 1–4.

Kopp, Tobias, Marco Baumgartner, and Steffen Kinkel. 2021. Success factors for introducing industrial human-robot interaction in practice: an empirically driven framework. *The International Journal of Advanced Manufacturing Technology* 112(3): 685–704. Publisher: Springer.

Kuka. 2023. Kuka. <https://www.kuka.com>.

Lappalainen, Inka. 2019. Logistics Robots as an enabler of hospital service system renewal? In *The 10 years Naples Forum on Service. Service Dominant Logic, Network and Systems Theory and Service Science: Integrating three Perspectives for a New Service Agenda. Ischia, Italy*.

Lasota, Przemyslaw A, and Julie A Shah. 2015. Analyzing the effects of human-aware motion planning on close-proximity human–robot collaboration. *Human factors* 57(1):21–33. Publisher: Sage Publications Sage CA: Los Angeles, CA.

Lasseter, John. 1987. Principles of traditional animation applied to 3D computer animation. In *Proceedings of the 14th annual conference on Computer graphics and interactive techniques*, 35–44.

Le, Chi Hieu, Dang Thang Le, Daniel Arey, Popan Gheorghe, Anh My Chu, Xuan Bien Duong, Trung Thanh Nguyen, Trong Toai Truong, Chander Prakash, Shi-Tian Zhao, et al. 2020. Challenges and conceptual framework to develop heavy-load manipulators for smart factories. *International Journal of Mechatronics and Applied Mechanics* 8(2):209–216.

Lieberman, Henry, Fabio Paternò, Markus Klann, and Volker Wulf. 2006. End-user development: An emerging paradigm. In *End user development*, 1–8. Springer.

Lin, Hsien-I, and YP Chiang. 2015. Understanding Human Hand Gestures for Learning Robot Pick-and-Place Tasks. *International Journal of Advanced Robotic Systems* 12(5):49. Publisher: SAGE Publications Sage UK: London, England.

Liu, Li, Andrew J Schoen, Curt Henrichs, Jingshan Li, Bilge Mutlu, Yajun Zhang, and Robert G Radwin. 2022. Human robot collaboration for enhancing work activities. *Human Factors* 00187208221077722.

Lund, Arnold. 2001. Measuring Usability with the USE Questionnaire. *Usability Interface* 8(2):3–6.

Malik, Ali Ahmad, and Arne Bilberg. 2019. Complexity-based task allocation in human-robot collaborative assembly. *Industrial Robot: the international journal of robotics research and application* 46(4):471–480.

Maloney, John, Mitchel Resnick, Natalie Rusk, Brian Silverman, and Evelyn Eastmond. 2010. The Scratch Programming Language and Environment. *ACM Transactions on Computing Education (TOCE)* 10(4):16. Publisher: ACM.

Marquart, C. L., C. Hinojosa, Z. Swiecki, B. Eagan, and D. W. Shaffer. 2018. Epistemic network analysis. <http://app.epistemicnetwork.org>.

Marturi, Naresh, Marek Kopicki, Alireza Rastegarpanah, Vijaykumar Rajasekaran, Maxime Adjigble, Rustam Stolkin, Aleš Leonardis, and Yasemin Bekiroglu. 2019. Dynamic grasp and trajectory planning for moving objects. *Autonomous Robots* 43(5):1241–1256. Publisher: Springer.

- Mateo, Carlos, Alberto Brunete, Ernesto Gamba, and Miguel Hernando. 2014. Hammer: An Android Based Application for End-User Industrial Robot Programming. In *2014 IEEE/ASME 10th International Conference on Mechatronic and Embedded Systems and Applications (MESA)*, 1–6. IEEE.
- Matsakis, Nicholas D, and Felix S Klock. 2014. The rust language. *ACM SIGAda Ada Letters* 34(3):103–104. Publisher: ACM New York, NY, USA.
- Matsas, Elias, and George-Christopher Vosniakos. 2017. Design of a virtual reality training system for human–robot collaboration in manufacturing tasks. *International Journal on Interactive Design and Manufacturing (IJIDeM)* 11(2):139–153. Publisher: Springer.
- Matthias, Bjoern, Soenke Kock, Henrik Jerregard, Mats Källman, and Ivan Lundberg. 2011. Safety of Collaborative Industrial Robots: Certification Possibilities for a Collaborative Assembly Robot Concept. In *Assembly and Manufacturing (ISAM), 2011 IEEE International Symposium on*, 1–6. IEEE.
- Mavrikios, Dimitris, Nikolaos Papakostas, Dimitris Mourtzis, and George Chrysosouris. 2013. On industrial learning and training for the factories of the future: a conceptual, cognitive and technology framework. *Journal of Intelligent Manufacturing* 24(3):473–485. Publisher: Springer.
- Mayer, Richard E. 1981. The psychology of how novices learn computer programming. *ACM Computing Surveys (CSUR)* 13(1):121–141.
- Maynard, Harold B, Gustave James Stegemerten, and John L Schwab. 1948. Methods-time measurement.
- Melzer, Ayelet, Tal Shafir, and Rachelle Palnick Tsachor. 2019. How Do We Recognize Emotion From Movement? Specific Motor Components Contribute to the Recognition of Each Emotion. *Frontiers in Psychology* 10:1389.
- Michaelis, Joseph, Amanda Siebert-Evenstone, David Shaffer, and Bilge Mutlu. 2020. Collaborative or Simply Uncaged? Understanding Human-Cobot Interac-

tions in Automation. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*, 1–12. CHI '20. Place: Honolulu, HI, USA.

Michalos, George, Sotiris Makris, Jason Spiliotopoulos, Ioannis Misios, Panagiota Tsarouchi, and George Chryssolouris. 2014. ROBO-PARTNER: Seamless Human-Robot Cooperation for Intelligent, Flexible and Safe Operations in the Assembly Factories of the Future. *Procedia CIRP* 23:71–76. Publisher: Elsevier.

Michalos, George, Sotiris Makris, Panagiota Tsarouchi, Toni Guasch, Dimitris Kontovrakis, and George Chryssolouris. 2015. Design Considerations for Safe Human-Robot Collaborative Workplaces. *Procedia CIRP* 37:248–253. Publisher: Elsevier.

Michalowski, Marek P, Selma Sabanovic, and Reid Simmons. 2006. A spatial model of engagement for a social robot. In *9th IEEE International Workshop on Advanced Motion Control, 2006.*, 762–767. IEEE.

Miller, David. 2021. Robotics Adoption Survey Finds Ups, Downs, and a Few Surprises. Publication Title: Automation World.

Monguzzi, Andrea, Mahmoud Badawi, Andrea Maria Zanchettin, and Paolo Rocco. 2022. A mixed capability-based and optimization methodology for human-robot task allocation and scheduling. In *2022 31st IEEE International Conference on Robot and Human Interactive Communication (ro-man)*, 1271–1276. IEEE.

Alvarez-de-los Mozos, Esther, and Arantxa Renteria. 2017. Collaborative robots in e-waste management. *Procedia Manufacturing* 11:55–62. Publisher: Elsevier.

Murata, Tadao. 1989. Petri nets: Properties, analysis and applications. *Proceedings of the IEEE* 77(4):541–580.

Mutlu, Bilge, Allison Terrell, and Chien-Ming Huang. 2013. Coordination mechanisms in human-robot collaboration. In *Proceedings of the Workshop on Collaborative Manipulation, 8th ACM/IEEE International Conference on Human-Robot Interaction*, 1–6. Citeseer.

Muxfeldt, Arne, Jan-Henrik Kluth, and Daniel Kubus. 2014. Kinesthetic Teaching in Assembly Operations—A User Study. In *International Conference on Simulation, Modeling, and Programming for Autonomous Robots*, 533–544. Springer.

National Center for O*NET Development. 2023. O*NET Online National Center for O*NET Development, Accessed at: www.onetonline.org/. Accessed 16 October 2023.

Nguyen, Hai, Matei Ciocarlie, Kaijen Hsiao, and Charles Kemp. 2013. ROS Commander (ROSCO): Behavior Creation for Home Robots. In *2013 IEEE International Conference on Robotics and Automation*, 467–474. IEEE.

Ortiz, Jesús Hamilton. 2020. Industry 4.0: Current status and future trends.

Palleschi, Alessandro, Mazin Hamad, Saeed Abdolshah, Manolo Garabini, Sami Haddadin, and Lucia Pallottino. 2021. Fast and safe trajectory planning: Solving the cobot performance/safety trade-off in human-robot shared environments. *IEEE Robotics and Automation Letters* 6(3):5445–5452.

Paxton, Chris, Andrew Hundt, Felix Jonathan, Kelleher Guerin, and Gregory Hager. 2017. CoSTAR: Instructing Collaborative Robots with Behavior Trees and Vision. In *Robotics and Automation (ICRA), 2017 IEEE International Conference on*, 564–571. IEEE.

Paxton, Chris, Felix Jonathan, Andrew Hundt, Bilge Mutlu, and Gregory D Hager. 2018. Evaluating Methods for End-User Creation of Robot Task Plans. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 6086–6092. IEEE.

Pearce, Margaret, Bilge Mutlu, Julie Shah, and Robert Radwin. 2018. Optimizing makespan and ergonomics in integrating collaborative robots into manufacturing processes. *IEEE transactions on automation science and engineering* 15(4):1772–1784.

Perlin, Ken. 1985. *An image synthesizer*, vol. 19. ACM.

- . 1995. Real time responsive animation with personality. *IEEE transactions on visualization and Computer Graphics* 1(1):5–15. Publisher: IEEE.
- . 2002. Improving noise. In *ACM Transactions on Graphics (TOG)*, vol. 21, 681–682. ACM. Issue: 3.
- Perlin, Ken, and Athomas Goldberg. 1996. Improv: A system for scripting interactive actors in virtual worlds. In *Proceedings of the 23rd annual Conf. on Computer graphics and interactive techniques*, 205–216.
- Perzylo, Alexander, Nikhil Somani, Stefan Profanter, Ingmar Kessler, Markus Rickert, and Alois Knoll. 2016. Intuitive instruction of industrial robots: Semantic process descriptions for small lot production. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2293–2300. IEEE.
- Peterson, James L. 1977. Petri nets. *ACM Computing Surveys (CSUR)* 9(3):223–252.
- Phillips, Elizabeth, Kristin E Schaefer, Deborah R Billings, Florian Jentsch, and Peter A Hancock. 2016. Human-animal teams as an analog for future human-robot teams: Influencing design and fostering trust. *Journal of Human-Robot Interaction* 5(1):100–125.
- Porfirio, David, Allison Sauppé, Aws Albarghouthi, and Bilge Mutlu. 2020. Transforming robot programs based on social context. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*, 1–12.
- Porfirio, David, Allison Sauppé, Aws Albarghouthi, and Bilge Mutlu. 2018. Authoring and verifying human-robot interactions. In *Proceedings of the 31st Annual ACM Symposium on User Interface Software and Technology*, 75–86.
- Porfirio, David, Laura Stegner, Maya Cakmak, Allison Sauppé, Aws Albarghouthi, and Bilge Mutlu. 2023. Sketching robot programs on the fly. In *Proceedings of the 2023 ACM/IEEE International Conference on Human-Robot Interaction*, 584–593.
- Pot, Emmanuel, Jérôme Monceaux, Rodolphe Gelin, and Bruno Maisonnier. 2009. Choregraphe: a graphical tool for humanoid robot programming. In *RO-MAN*

2009-*The 18th IEEE International Symposium on Robot and Human Interactive Communication*, 46–51. IEEE.

Quigley, Morgan, Ken Conley, Brian Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, and Andrew Y Ng. 2009. ROS: An Open-Source Robot Operating System. In *ICRA Workshop on Open Source Software*, vol. 3, 5. Kobe, Japan. Issue: 3.2.

Raffin, Antonin, Ashley Hill, Adam Gleave, Anssi Kanervisto, Maximilian Ernestus, and Noah Dormann. 2021. Stable-baselines3: Reliable reinforcement learning implementations. *Journal of Machine Learning Research* 22(268):1–8.

Rakita, Daniel, Bilge Mutlu, and Michael Gleicher. 2017. A motion retargeting method for effective mimicry-based teleoperation of robot arms. In *Proceedings of the 2017 ACM/IEEE International Conf. on Human-Robot Interaction*, 361–370. ACM.

———. 2018. RelaxedIK: Real-time synthesis of accurate and feasible robot arm motion. In *Robotics: Science and systems*, vol. 14, 26–30. Pittsburgh, PA.

———. 2022. PROXIMA: An Approach for Time or Accuracy Budgeted Collision Proximity Queries. In *Proceedings of Robotics: Science and Systems (RSS)*.

Ras, Eric, Fridolin Wild, Christoph Stahl, and Alexandre Baudet. 2017. Bridging the skills gap of workers in Industry 4.0 by human performance augmentation tools: Challenges and roadmap. In *Proceedings of the 10th International Conference on PErvasive Technologies Related to Assistive Environments*, 428–432.

Ravichandar, Harish, Athanasios S Polydoros, Sonia Chernova, and Aude Billard. 2020. Recent advances in robot learning from demonstration. *Annual review of control, robotics, and autonomous systems* 3:297–330.

Resnick, Mitchel, John Maloney, Andrés Monroy-Hernández, Natalie Rusk, Evelyn Eastmond, Karen Brennan, Amon Millner, Eric Rosenbaum, Jay Silver, Brian Silverman, and others. 2009. Scratch: programming for all. *Communications of the ACM* 52(11):60–67. Publisher: ACM New York, NY, USA.

- Rethink Robotics. 2023. Rethink Robotics. <https://www.rethinkrobotics.com>.
- Ribeiro, Tiago, and Ana Paiva. 2017. Animating the Adelino Robot with ERIK: The Expressive Robotics Inverse Kinematics. In *Proceedings of the 19th ACM International Conf. on Multimodal Interaction*, 388–396. ICMI '17, New York, NY, USA: Association for Computing Machinery. Event-place: Glasgow, UK.
- Riedelbauch, Dominik, and Dominik Henrich. 2018. Fast Graphical Task Modelling for Flexible Human-Robot Teaming. In *ISR 2018; 50th International Symposium on Robotics*, 1–6.
- Riek, Laurel D, Tal-Chen Rabinowitch, Paul Bremner, Anthony G Pipe, Mike Fraser, and Peter Robinson. 2010. Cooperative gestures: Effective signaling for humanoid robots. In *2010 5th ACM/IEEE International Conference on Human-Robot Interaction (HRI)*, 61–68. IEEE.
- Rohmert, W. 1960. Determination of the recovery pause for static work of man. *Internationale Zeitschrift Fur Angewandte Physiologie, Einschliesslich Arbeitsphysiologie* 18:123–164.
- Sakr, Maram, Zexi Jesse Li, HF Machiel Van der Loos, Dana Kulić, and Elizabeth A Croft. 2022. Quantifying demonstration quality for robot learning and generalization. *IEEE Robotics and Automation Letters* 7(4):9659–9666.
- Sanctuary AI. 2023. Sanctuary AI. <https://sanctuary.ai>.
- Sanner, Michel F, and others. 1999. Python: a programming language for software integration and development. *J Mol Graph Model* 17(1):57–61.
- Sauer, Vanessa, Axel Sauer, and Alexander Mertens. 2021. Zoomorphic gestures for communicating cobot states. *IEEE Robotics and Automation Letters* 6(2):2179–2185. Publisher: IEEE.
- Sauppé, Allison, and Bilge Mutlu. 2015. The social impact of a robot co-worker in industrial settings. In *Proceedings of the 33rd annual acm conference on human factors in computing systems*, 3613–3622.

Sauppé, Allison, and Bilge Mutlu. 2014. Design Patterns for Exploring and Prototyping Human-robot Interactions. In *Proceedings of the 32nd Annual ACM Conference on Human Factors in Computing Systems*, 1439–1448. CHI '14, New York, NY, USA: ACM. Event-place: Toronto, Ontario, Canada.

Schnell, Marie. 2021. Challenges when introducing collaborative robots in sme manufacturing industry.

Schoen, Andrew, Curt Henrichs, Mathias Strohkirch, and Bilge Mutlu. 2020. Authr: A Task Authoring Environment for Human-Robot Teams. In *Proceedings of the 33rd Annual ACM Symposium on User Interface Software and Technology*, 1194–1208.

Schoen, Andrew, and Bilge Mutlu. 2024. OpenVP: A Customizable Visual Programming Environment for Robotics Applications. In *Proceedings of the 2024 ACM/IEEE International Conference on Human-Robot Interaction (in press)*.

Schoen, Andrew, Dakota Sullivan, Ze Dong Zhang, Daniel Rakita, and Bilge Mutlu. 2023. Lively: Enabling multimodal, lifelike, and extensible real-time robot motion. In *Proceedings of the 2023 ACM/IEEE International Conference on Human-Robot Interaction*, 594–602.

Schoen, Andrew, Nathan White, Curt Henrichs, Amanda Siebert-Evenstone, David Shaffer, and Bilge Mutlu. 2022. Coframe: A system for training novice cobot programmers. In *2022 17th acm/ieee international conference on human-robot interaction (hri)*, 185–194. IEEE.

Schulman, John, Jonathan Ho, Alex X Lee, Ibrahim Awwal, Henry Bradlow, and Pieter Abbeel. 2013. Finding locally optimal, collision-free trajectories with sequential convex optimization. In *Robotics: science and systems*, vol. 9, 1–10. Berlin, Germany.

Schulman, John, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*.

- Senft, Emmanuel, Michael Hagenow, Robert Radwin, Michael Zinn, Michael Gleicher, and Bilge Mutlu. 2021a. Situated Live Programming for Human-Robot Collaboration. In *ACM Symposium on User Interface and Software Technology*.
- Senft, Emmanuel, Michael Hagenow, Kevin Welsh, Robert Radwin, Michael Zinn, Michael Gleicher, and Bilge Mutlu. 2021b. Task-Level Authoring for Remote Robot Teleoperation. *Frontiers in Robotics & AI*.
- Shaffer, D, and A Ruis. 2017. Epistemic network analysis: A worked example of theory-based learning analytics. *Handbook of learning analytics*.
- Shaffer, David Williamson. 2017. *Quantitative ethnography*. Cathcart Press.
- Shaffer, David Williamson, Wesley Collier, and Andrew R Ruis. 2016. A tutorial on epistemic network analysis: Analyzing the structure of connections in cognitive, social, and interaction data. *Journal of Learning Analytics* 3(3):9–45.
- Shaffer, DW, F Borden, A Srinivasan, J Saucerman, G Arastoopour, W Collier, AR Ruis, and KA Frank. 2015. The ncoder: A technique for improving the utility of inter-rater reliability statistics. *Games and Professional Simulations Technical Report 1*.
- Shi, Jane, Glenn Jimmerson, Tom Pearson, and Roland Menassa. 2012. Levels of human and robot collaboration for automotive manufacturing. In *Proceedings of the Workshop on Performance Metrics for Intelligent Systems*, 95–100.
- Shmatko, Natalia, and Galina Volkova. 2020. Bridging the Skill Gap in Robotics: Global and National Environment. *SAGE Open* 10(3):2158244020958736. Publisher: SAGE Publications Sage CA: Los Angeles, CA.
- Sial, Sara Baber, Muhammad Baber Sial, Yasar Ayaz, Syed Irtiza Ali Shah, and Aleksandar Zivanovic. 2016. Interaction of robot with humans by communicating simulated emotional states through expressive movements. *Intelligent Service Robotics* 9(3):231–255. Publisher: Springer.

Siebert-Evenstone, Amanda, Joseph E Michaelis, David Williamson Shaffer, and Bilge Mutlu. 2021. Safety First: Developing a Model of Expertise in Collaborative Robotics. In *International Conference on Quantitative Ethnography*, 304–318. Springer.

Siebert-Evenstone, Amanda L, Golnaz Arastoopour Irgens, Wesley Collier, Zachari Swiecki, Andrew R Ruis, and David Williamson Shaffer. 2017. In search of conversational grain size: Modelling semantic structure using moving stanza windows. *Journal of Learning Analytics* 4(3):123–139.

Silver, Tom, Varun Hariprasad, Reece S Shuttleworth, Nishanth Kumar, Tomás Lozano-Pérez, and Leslie Pack Kaelbling. 2022. Pddl planning with pretrained large language models. In *Neurips 2022 foundation models for decision making workshop*.

Simões, Ana Correia, António Lucas Soares, and Ana Cristina Barros. 2020. Factors influencing the intention of managers to adopt collaborative robots (cobots) in manufacturing organizations. *Journal of Engineering and Technology Management* 57: 101574. Publisher: Elsevier.

Skoglund, Alexander, Boyko Iliev, Bourhane Kadmiry, and Rainer Palm. 2007. Programming by Demonstration of Pick-and-Place Tasks for Industrial Manipulators using Task Primitives. In *2007 International Symposium on Computational Intelligence in Robotics and Automation*, 368–373. IEEE.

Słowikowski, Marcin, Zbigniew Pilat, Michał Smater, and Jacek Zieliński. 2018. Collaborative learning environment in vocational education. In *AIP Conference Proceedings*, vol. 2029, 020070. AIP Publishing LLC. Issue: 1.

Snibbe, S, M Scheeff, and K Rahardja. 1999. A layered architecture for lifelike robotic motion. *Proceedings of the International Conf. on Advanced Robotics*.

Softbank Robotics. 2022. Autonomous Life. http://doc.aldebaran.com/2-8/family/nao_user_guide/nao_life.html.

- Song, H., M. J. Kim, S. Jeong, H. Suk, and D. Kwon. 2009. Design of idle motions for service robot via video ethnography. In *RO-MAN 2009 - The 18th IEEE International Symp. on Robot and Human Interactive Communication*, 195–199.
- Stanton, Neville. 2006. Hierarchical Task Analysis: Developments, Applications, and Extensions. *Applied Ergonomics* 37(1):55–79. Publisher: Elsevier.
- Steinmetz, Frank, Verena Nitsch, and Freerk Stulp. 2019. Intuitive Task-Level Programming by Demonstration Through Semantic Skill Recognition. *IEEE Robotics and Automation Letters* 4(4):3742–3749.
- Steinmetz, Frank, Annika Wollschläger, and Roman Weitschat. 2018. RAZER-A HRI for Visual Task-Level Programming and Intuitive Skill Parameterization. *IEEE Robotics and Automation Letters* 3(3):1362–1369.
- Strabala, Kyle, Min Kyung Lee, Anca Dragan, Jodi Forlizzi, Siddhartha S Srinivasa, Maya Cakmak, and Vincenzo Micelli. 2013. Toward seamless human-robot handovers. *Journal of Human-Robot Interaction* 2(1):112–132. Publisher: Journal of Human-Robot Interaction Steering Committee.
- Şucan, Ioan A., Mark Moll, and Lydia E. Kavraki. 2012. The Open Motion Planning Library. *IEEE Robotics & Automation Magazine* 19(4):72–82. <https://ompl.kavrakilab.org>.
- Sullivan, Sarah, Charles Warner-Hillard, Brendan Eagan, Ryan J Thompson, Andrew R Ruis, Krista Haines, Carla M Pugh, David Williamson Shaffer, and Hee Soo Jung. 2018. Using epistemic network analysis to identify targets for educational interventions in trauma team communication. *Surgery* 163(4):938–943.
- Teiwes, Johannes, Timo Bänziger, Andreas Kunz, and Konrad Wegener. 2016. Identifying the potential of human-robot collaboration in automotive assembly lines using a standardised work description. In *2016 22nd international conference on automation and computing (icac)*, 78–83. IEEE.

Terzioğlu, Yunus, Bilge Mutlu, and Erol Şahin. 2020. Designing social cues for collaborative robots: the role of gaze and breathing in human-robot collaboration. In *Proceedings of the 2020 ACM/IEEE International Conference on Human-Robot Interaction*, 343–357.

Tian, Bo, Mukund Janardhanan, and Marina Marinelli. 2023. A systematic investigation of the barriers to effective implementation of human-robot assembly line: an integrated multi-criteria decision-making approach. *International Journal of Computer Integrated Manufacturing* 1–26.

Toris, Russell, Julius Kammerl, David Lu, Jihoon Lee, Odest Chadwicke Jenkins, Sarah Osentoski, Mitchell Wills, and Sonia Chernova. 2015. Robot Web Tools: Efficient Messaging for Cloud Robotics. In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 4530–4537. IEEE.

Truong, Arthur, Hugo Boujut, and Titus Zaharia. 2016. Laban descriptors for gesture recognition and emotional analysis. *The visual computer* 32(1):83–98. Publisher: Springer.

Universal Robots. 2021. Universal Robots Academy. <https://academy.universal-robots.com>.

———. 2023. Universal Robots. <https://www.universal-robots.com>.

Van Someren, Maartin, Yvonne Barnard, and Jacobijn Sandberg. 1994. The Think Aloud Method : A Practical Guide to Modelling Cognitive Processes. In *London: AcademicPress*.

Vicentini, Federico. 2021. Collaborative robotics: a survey. *Journal of Mechanical Design* 143(4):040802. Publisher: American Society of Mechanical Engineers.

Von Laban, Rudolf, and Roderyk Lange. 1975. *Laban's principles of dance and movement notation*. Princeton Book Co Pub.

Wang, Yeping, Gopika Ajaykumar, and Chien-Ming Huang. 2020. See what i see: Enabling user-centric robotic assistance using first-person demonstrations. In

Proceedings of the 2020 acm/ieee international conference on human-robot interaction, 639–648.

Waters, TR, V Putz-Anderson, and A Garg. 1994. Applications manual for the revised niosh lifting equation. Tech. Rep., National Institute for Occupational Safety and Health, DHHS (NIOSH).

Webkid GmbH. 2019. Reactflow: Build better node-based uis with react flow. <https://reactflow.dev>.

Weintrop, David, Afsoon Afzal, Jean Salac, Patrick Francis, Boyang Li, David C Shepherd, and Diana Franklin. 2018. Evaluating CoBlox: A comparative study of robotics programming environments for adult novices. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, 1–12.

Weintrop, David, David C Shepherd, Patrick Francis, and Diana Franklin. 2017. Blockly goes to work: Block-based programming for industrial robots. In *2017 IEEE Blocks and Beyond Workshop (B&B)*, 29–36. IEEE.

Wilson, Garrett, Christopher Pereyda, Nisha Raghunath, Gabriel de la Cruz, Shivam Goel, Sepehr Nesaei, Bryan Minor, Maureen Schmitter-Edgecombe, Matthew E Taylor, and Diane J Cook. 2019. Robot-enabled support of daily activities in smart home environments. *Cognitive Systems Research* 54:258–272. Publisher: Elsevier.

Wingard, Jason, and Christine Farrugia. 2021. *The Great Skills Gap: Optimizing Talent for the Future of Work*. Stanford University Press.

Wingrave, Chadwick, and Joseph LaViola. 2010. Reflecting on the Design and Implementation Issues of Virtual Environments. *Presence* 19(2):179–195.

Witkin, Andrew, and Zoran Popovic. 1995. Motion warping. In *Proceedings of the 22nd annual Conf. on Computer graphics and interactive techniques*, 105–108.

- Wojtynek, Michael, Jochen Jakob Steil, and Sebastian Wrede. 2019. Plug, plan and produce as enabler for easy workcell setup and collaborative robot programming in smart factories. *KI-Künstliche Intelligenz* 33(2):151–161. Publisher: Springer.
- Young, Richard M. 1981. The machine inside the machine: Users' models of pocket calculators. *International Journal of Man-Machine Studies* 15(1):51–85.
- Zandin, Kjell. 1990. *Most work measurement systems: Basic most, mini most, maxi most*. Marcel Dekker Inc.
- Zhang, Chongjie, and Julie Shah. 2016. Co-optimizing Multi-agent Placement with Task Assignment and Scheduling. In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence*, 3308–3314. IJCAI'16, AAAI Press. Place: New York, New York, USA.
- Zhao, Fangyun, Curt Henrichs, and Bilge Mutlu. 2020. Task Interdependence in Human-Robot Teaming. In *2020 29th IEEE International Conference on Robot and Human Interactive Communication (RO-MAN)*, 1143–1149. IEEE.
- Ziaeeafard, Saeedeh, Michele H Miller, Mo Rastgaar, and Nina Mahmoudian. 2017. Co-robotics hands-on activities: A gateway to engineering design and STEM learning. *Robotics and Autonomous Systems* 97:40–50. Publisher: Elsevier.
- Ziparo, Vittorio A, Luca Iocchi, Pedro U Lima, Daniele Nardi, and Pier Francesco Palamara. 2011. Petri net plans: A framework for collaboration and coordination in multi-robot systems. *Autonomous Agents and Multi-Agent Systems* 23:344–383.
- Zuberek, Wlodek M. 1991. Timed petri nets definitions, properties, and applications. *Microelectronics Reliability* 31(4):627–644.
- Şucan, Ioan, Mark Moll, and Lydia Kavraki. 2012. The Open Motion Planning Library. *IEEE Robotics & Automation Magazine* 19(4):72–82.